aws

User Guide

Amazon Relational Database Service



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Relational Database Service: User Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon RDS? 1
Advantages of Amazon RDS 1
Comparison of responsibilities 2
Amazon RDS shared responsibility model 3
DB instances 4
Amazon RDS application architecture: example 5
DB engines 6
DB instance classes 7
DB instance storage 8
DB instances in an Amazon Virtual Private Cloud (Amazon VPC) 8
AWS Regions and Availability Zones 9
Availability Zones 9
Multi-AZ deployments 10
Access control with security groups 12
Amazon RDS monitoring 13
User interfaces to Amazon RDS 14
AWS Management Console 15
Command line interface 15
Amazon RDS APIs 15
How you are charged for Amazon RDS 16
AWS Free Tier on Amazon RDS 16
What's next? 17
Getting started 17
Topics specific to database engines 17
DB instances 18
DB instance classes 21
DB instance class types 21
Supported DB engines 29
Determining DB instance class support in AWS Regions 103
Changing your DB instance class 109
Configuring the processor for RDS for Oracle 109
Hardware specifications 132
DB instance storage 175
Storage types 175

Provisioned IOPS storage ... 176
General Purpose storage ... 180
SSD performance characteristics ... 184
Magnetic storage (legacy, not recommended) ... 189
Additional storage volumes ... 189
Dedicated log volume (DLV) ... 190
Monitoring database performance ... 191
Factors that affect database performance ... 193
Regions, Availability Zones, and Local Zones ... 196
AWS Regions ... 197
Availability Zones ... 202
Local Zones ... 203
Supported Amazon RDS features by Region and engine ... 205
Table conventions ... 205
Feature quick reference ... 206
Blue/Green Deployments ... 208
Cross-Region automated backups ... 209
Cross-Region read replicas ... 210
Database activity streams ... 213
Dual-stack mode ... 220
Export snapshots to S3 ... 241
IAM database authentication ... 254
Kerberos authentication ... 260
Multi-AZ DB clusters ... 275
Performance Insights ... 286
RDS Custom ... 287
Amazon RDS Proxy ... 308
Secrets Manager integration ... 323
Zero-ETL integrations ... 324
Engine-native features ... 336
DB instance billing for Amazon RDS ... 337
On-Demand DB instances ... 339
Reserved DB instances ... 341
Setting up ... 355
Sign up for an AWS account ... 355
Create a user with administrative access ... 356

Grant programmatic access ... 357
Determine requirements ... 359
Provide access to your DB instance ... 361
Getting started ... 365
Creating and connecting to a MariaDB DB instance ... 366
Prerequisites ... 367
Create an EC2 instance ... 367
Create a MariaDB DB instance ... 373
(Optional) Create VPC, EC2 instance, and MariaDB instance using CloudFormation ... 378
Connect to a MariaDB DB instance ... 380
Delete the EC2 instance and DB instance ... 383
(Optional) Delete the EC2 instance and DB instance created with CloudFormation ... 384
(Optional) Connect your DB instance to a Lambda function ... 384
Creating and connecting to a Microsoft SQL Server DB instance ... 386
Prerequisites ... 387
Create an EC2 instance ... 387
Create a SQL Server DB instance ... 392
(Optional) Create VPC, EC2 instance, and SQL Server instance using CloudFormation ... 398
Connecting to your SQL Server DB instance ... 400
Exploring your sample DB instance ... 402
Delete the EC2 instance and DB instance ... 404
(Optional) Delete the EC2 instance and DB instance created with CloudFormation ... 405
(Optional) Connect your DB instance to a Lambda function ... 405
Creating and connecting to a MySQL DB instance ... 406
Prerequisites ... 407
Create an EC2 instance ... 407
Create a MySQL DB instance ... 413
(Optional) Create VPC, EC2 instance, and MySQL instance using CloudFormation ... 418
Connect to a MySQL DB instance ... 420
Delete the EC2 instance and DB instance ... 423
(Optional) Delete the EC2 instance and DB instance created with CloudFormation ... 424
(Optional) Connect your DB instance to a Lambda function ... 424
Creating and connecting to an Oracle DB instance ... 426
Prerequisites ... 427
Step 1: Create an EC2 instance ... 427
Step 2: Create an Oracle DB instance ... 433

(Optional) Create VPC, EC2 instance, and Oracle DB instance using CloudFormation 438
Step 3: Connect your SQL client to an Oracle DB instance 440
Step 4: Delete the EC2 instance and DB instance 444
(Optional) Delete the EC2 instance and DB instance created with CloudFormation 444
(Optional) Connect your DB instance to a Lambda function 445
Creating and connecting to a PostgreSQL DB instance 446
Prerequisites 447
Create an EC2 instance 447
Create a PostgreSQL DB instance 453
(Optional) Create VPC, EC2 instance, and PostgreSQL instance using CloudFormation 458
Connect to a PostgreSQL DB instance 460
Delete the EC2 instance and DB instance 463
(Optional) Delete the EC2 instance and DB instance created with CloudFormation 464
(Optional) Connect your DB instance to a Lambda function 464
Tutorial: Create a web server and an Amazon RDS DB instance 466
Launch an EC2 instance to connect with your DB instance 467
Create a DB instance 473
Install a web server 494
Tutorial: Create a Lambda function to access your Amazon RDS DB instance 506
Prerequisites 507
Create an Amazon RDS DB instance 507
Create Lambda function and proxy 508
Create a function execution role 509
Create a Lambda deployment package 511
Update the Lambda function 513
Test your Lambda function in the console 515
Create an Amazon SQS queue 516
Create an event source mapping to invoke your Lambda function 517
Test and monitor your setup 517
Clean up your resources 519
Tutorials and sample code 521
Tutorials in this guide 521
Tutorials in other AWS guides 522
Tutorials and sample code in GitHub 523
AWS Database Cookbook 523
AWS workshop - Amazon RDS PostgreSQL 524

AWS workshop - Amazon RDS MySQL ... 525
Working with AWS SDKs ... 525
Best practices for Amazon RDS ... 527
Amazon RDS basic operational guidelines ... 527
DB instance RAM recommendations ... 528
Keeping database engine versions up to date ... 529
AWS database drivers ... 529
Using Enhanced Monitoring ... 530
Using metrics ... 530
Viewing performance metrics ... 530
Evaluating performance metrics ... 533
Tuning queries ... 535
Best practices for working with MySQL ... 536
Table size ... 536
Number of tables ... 537
Storage engine ... 537
Best practices for working with MariaDB ... 538
Table size ... 538
Number of tables ... 539
Storage engine ... 539
Best practices for working with Oracle ... 540
Best practices for working with PostgreSQL ... 540
Loading data into a PostgreSQL DB instance ... 540
Working with the PostgreSQL autovacuum feature ... 541
Amazon RDS for PostgreSQL best practices video ... 542
Best practices for working with SQL Server ... 542
Amazon RDS for SQL Server best practices video ... 543
Working with DB parameter groups ... 544
Automating DB instance creation ... 544
New features video ... 545
Programmatic access to Amazon RDS ... 546
Console-to-Code ... 547
Configuring a DB instance ... 548
Creating a DB instance ... 549
Prerequisites ... 549
Creating a DB instance ... 555

Available settings ... 562
Creating resources with AWS CloudFormation ... 599
RDS and CloudFormation templates ... 599
Learn more about CloudFormation ... 599
Connecting to a DB instance ... 600
Finding the connection information ... 600
Scenarios for accessing a DB instance ... 604
Connecting to DB instances with the AWS drivers ... 605
Connecting to a DB instance running a specific DB engine ... 606
Managing connections with RDS Proxy ... 606
Database authentication options ... 607
Encrypted connections ... 607
Working with option groups ... 608
Option groups overview ... 608
Creating an option group ... 610
Copying an option group ... 613
Adding an option to an option group ... 614
Listing the options and option settings for an option group ... 620
Modifying an option setting ... 621
Removing an option from an option group ... 624
Deleting an option group ... 627
Parameter groups ... 630
Overview of parameter groups ... 630
DB parameter groups ... 634
DB cluster parameter groups ... 650
Comparing DB parameter groups ... 664
Specifying DB parameters ... 664
Creating an ElastiCache cache from Amazon RDS ... 672
Overview of ElastiCache cache creation with RDS DB instance settings ... 672
Creating an ElastiCache cache with settings from an RDS DB instance ... 673
Auto-migrating EC2 databases ... 677
Overview ... 677
Prerequisites ... 678
Limitations ... 679
Creating IAM resources ... 680
Set up data migration ... 687

Managing migrations ... 689
Monitoring ... 692
Tutorial: Creating a MySQL DB instance with a custom parameter and new option group ... 693
Prerequisites ... 694
Create a custom parameter group ... 694
Modify parameter values ... 695
Create option group ... 695
Add option ... 696
Create your DB DB instance ... 696
Managing a DB instance ... 698
Stopping a DB instance ... 699
Use cases ... 699
Time considerations ... 700
Supported DB engines, classes, and Regions ... 701
Support for Multi-AZ ... 701
How it works ... 701
Limitations ... 703
Option and parameter groups ... 703
Public IP addresses ... 704
Stopping a DB instance ... 704
Starting a DB instance ... 706
Rebooting a DB instance ... 708
Use cases for rebooting a DB instance ... 708
How rebooting works ... 709
Rebooting in Multi-AZ ... 709
Considerations ... 710
Prerequisites ... 710
Rebooting a DB instance: basic steps ... 711
Connecting an EC2 instance ... 713
Overview ... 714
Connecting an EC2 instance ... 719
Viewing connected compute resources ... 722
Connecting to a DB instance running a specific DB engine ... 723
Connecting a Lambda function ... 724
Overview ... 726
Connecting a Lambda function ... 737

Viewing connected compute resources ... 739
Modifying a DB instance ... 740
Scheduling modifications ... 742
Available settings ... 743
Maintaining a DB instance ... 779
Overview of DB instance maintenance updates ... 779
Viewing pending maintenance ... 780
Maintenance for Multi-AZ deployments ... 783
Maintenance window ... 783
Applying updates ... 789
Operating system updates ... 791
AWS Organizations upgrade rollout ... 794
Upgrading the engine version ... 799
Manually upgrading the engine version ... 800
Automatically upgrading the minor engine version ... 802
Renaming a DB instance ... 807
Renaming to replace an existing DB instance ... 808
Working with DB instance read replicas ... 810
Overview ... 812
Differences between DB engines ... 816
Creating a read replica ... 819
Promoting a read replica ... 822
Monitoring read replication ... 827
Cross-Region read replicas ... 830
Tagging RDS resources ... 844
Why use RDS tags? ... 845
How RDS tags work ... 846
Best practices ... 849
Copying tags to DB snapshots ... 850
Tagging automated backups ... 851
Adding and deleting tags in Amazon RDS ... 852
Tutorial: Specify which DB instances to stop by using tags ... 856
ARNs in Amazon RDS ... 860
Constructing an ARN ... 860
Getting an existing ARN ... 868
Working with storage ... 871

Viewing storage volume details for your DB instance ... 871
Increasing DB instance storage capacity ... 872
Removing additional storage volumes ... 878
Managing capacity automatically with storage autoscaling ... 880
Upgrading the storage file system ... 888
Modifying Provisioned IOPS settings ... 889
I/O-intensive storage modifications ... 891
Modifying General Purpose (gp3) settings ... 893
Using a dedicated log volume (DLV) ... 895
Deleting a DB instance ... 902
Prerequisites for deleting a DB instance ... 902
Considerations when deleting a DB instance ... 902
Deleting a DB instance ... 904
Tutorial: Managing a MySQL DB instance ... 907
Introduction ... 907
Prerequisites ... 907
Add tags to your DB instance ... 908
Increase DB storage ... 908
Create read replicas ... 909
Update tags ... 910
Configuring and managing a Multi-AZ deployment ... 913
Multi-AZ DB instance deployments ... 915
Converting a DB instance to a Multi-AZ deployment ... 917
Failing over a Multi-AZ DB instance ... 919
Multi-AZ failover with additional storage volumes ... 923
Multi-AZ DB cluster deployments ... 925
Instance class availability for Multi-AZ DB clusters ... 926
Multi-AZ DB cluster architecture ... 926
Parameter groups for Multi-AZ DB clusters ... 928
RDS Proxy with Multi-AZ DB clusters ... 928
Replica lag and Multi-AZ DB clusters ... 928
Multi-AZ DB cluster snapshots ... 931
Creating a Multi-AZ DB cluster ... 932
Connecting to a Multi-AZ DB cluster ... 957
Connecting an AWS compute resource and a Multi-AZ DB cluster ... 963
Modifying a Multi-AZ DB cluster ... 989

Upgrading a Multi-AZ DB cluster 1007
Renaming a Multi-AZ DB cluster 1010
Rebooting a Multi-AZ DB cluster 1013
Failing over a Multi-AZ DB cluster 1015
PostgreSQL logical replication with Multi-AZ DB clusters 1019
Working with Multi-AZ DB cluster read replicas 1024
Setting up external replication from Multi-AZ DB clusters 1035
Deleting a Multi-AZ DB cluster 1036
Limitations of Multi-AZ DB clusters 1038
RDS Extended Support 1039
RDS Extended Support overview 1040
RDS Extended Support charges 1041
Avoiding charges for RDS Extended Support 1041
Versions with RDS Extended Support 1042
RDS Extended Support version naming 1042
Responsibilities with RDS Extended Support 1043
Amazon RDS responsibilities 1043
Your responsibilities 1044
Creating a DB instance or a Multi-AZ DB cluster 1044
RDS Extended Support behavior 1045
Considerations for RDS Extended Support 1045
Create a DB instance or a Multi-AZ DB cluster with RDS Extended Support 1046
Viewing RDS Extended Support enrollment 1047
Viewing support dates 1050
Restoring a DB instance or a Multi-AZ DB cluster 1053
RDS Extended Support behavior 1053
Considerations for RDS Extended Support 1054
Restore a DB instance or a Multi-AZ DB cluster with RDS Extended Support 1055
Using Blue/Green Deployments for database updates 1057
Overview of Blue/Green Deployments 1058
Region and version availability 1059
Benefits 1059
Workflow 1059
Authorizing access 1064
Limitations and considerations 1065
Best practices 1074

Creating a blue/green deployment 1080
Preparing for a blue/green deployment 1080
Specifying changes 1083
Lazy loading and storage initialization 1086
Creating a blue/green deployment 1088
Available settings 1090
Viewing a blue/green deployment 1095
Switching a blue/green deployment 1099
Switchover timeout 1100
Switchover guardrails 1100
Switchover actions 1101
Switchover best practices 1102
Verifying CloudWatch metrics before switchover 1103
Monitoring replica lag prior to switchover 1104
Switching over a blue/green deployment 1105
After switchover 1107
Deleting a blue/green deployment 1109
Backing up, restoring, and exporting data 1113
Introduction to backups 1114
Backup storage 1114
Managing automated backups 1116
Backup window 1116
Backup retention period 1119
Enabling automated backups 1120
Retaining automated backups 1122
Deleting retained automated backups 1124
Unsupported MySQL storage engines 1128
Unsupported MariaDB storage engines 1129
Cross-Region automated backups 1130
Managing manual backups 1149
Creating a DB snapshot for a Single-AZ DB instance 1150
Creating a Multi-AZ DB cluster snapshot 1153
Deleting a DB snapshot 1155
Restoring to a DB instance 1158
Restoring from a snapshot 1159
Considerations 1161

Point-in-time recovery ... 1165
Restoring a Multi-AZ DB cluster to a specified time ... 1171
Restoring from a snapshot to a Multi-AZ DB cluster ... 1175
Restoring from a Multi-AZ DB cluster snapshot to a DB instance ... 1178
Tutorial: Restore a DB instance from a DB snapshot ... 1181
Copying a DB snapshot ... 1185
Copying a DB snapshot ... 1185
Limitations ... 1195
Considerations ... 1196
Sharing a DB snapshot ... 1206
Sharing a snapshot ... 1207
Sharing public snapshots ... 1211
Sharing encrypted snapshots ... 1213
Stopping snapshot sharing ... 1217
Exporting DB snapshot data to Amazon S3 ... 1219
Overview of exporting snapshot data ... 1220
Setting up access to an S3 bucket ... 1221
Exporting a DB snapshot ... 1226
Region and version availability ... 1230
Limitations ... 1230
Monitoring snapshot exports ... 1231
Canceling a snapshot export ... 1234
Failure messages ... 1235
Troubleshooting PostgreSQL permissions errors ... 1237
File naming conventions ... 1237
Data conversion ... 1239
Using AWS Backup ... 1249
Monitoring metrics in a DB instance ... 1250
Monitoring plan ... 1250
Performance baseline ... 1251
Performance guidelines ... 1251
Monitoring tools ... 1253
Automated monitoring tools ... 1253
Manual monitoring tools ... 1255
Viewing instance status ... 1257
Viewing Amazon RDSDB instance status ... 1258

Recommendations from Amazon RDS ... 1265
Viewing recommendations ... 1267
Applying recommendations ... 1275
Dismissing recommendations ... 1281
Modifying dismissed recommendations to active ... 1283
Recommendations reference ... 1284
Viewing metrics in the Amazon RDS console ... 1309
Viewing the Performance Insights dashboard ... 1310
Choosing the new monitoring view from the Monitoring tab ... 1311
Choosing the new monitoring view from the Performance Insights page ... 1313
Creating a custom dashboard ... 1314
Choosing the preconfigured dashboard ... 1317
Monitoring RDS with CloudWatch ... 1319
Overview of Amazon RDS and Amazon CloudWatch ... 1320
Viewing CloudWatch metrics ... 1322
Exporting Performance Insights metrics to CloudWatch ... 1327
Creating CloudWatch alarms ... 1333
Tutorial: Creating a CloudWatch alarm for DB cluster replica lag ... 1333
Monitoring with Database Insights ... 1341
Pricing ... 1341
Engine, Region, and instance class support ... 1341
Turning on the Advanced mode ... 1347
Turning on the Standard mode ... 1351
Monitor slow queries ... 1357
Considerations ... 1359
Monitoring DB load with Performance Insights ... 1360
Overview of Performance Insights ... 1361
Turning Performance Insights on and off ... 1374
Performance Schema for MariaDB or MySQL ... 1379
Performance Insights policies ... 1384
Analyzing metrics with the Performance Insights dashboard ... 1399
Viewing Performance Insights proactive recommendations ... 1448
Retrieving metrics with the Performance Insights API ... 1451
Logging Performance Insights calls using AWS CloudTrail ... 1475
VPC endpoints (AWS PrivateLink) ... 1478
Analyzing performance with DevOps Guru for RDS ... 1482

Benefits of DevOps Guru for RDS ... 1482
How DevOps Guru for RDS works ... 1483
Setting up DevOps Guru for RDS ... 1485
Monitoring the OS with Enhanced Monitoring ... 1493
Overview of Enhanced Monitoring ... 1493
Setting up and enabling Enhanced Monitoring ... 1495
Viewing OS metrics in the RDS console ... 1501
Viewing OS metrics using CloudWatch Logs ... 1505
RDS metrics reference ... 1506
CloudWatch metrics for RDS ... 1506
CloudWatch dimensions for RDS ... 1527
CloudWatch metrics for Performance Insights ... 1528
Counter metrics for Performance Insights ... 1531
SQL statistics for Performance Insights ... 1564
OS metrics in Enhanced Monitoring ... 1581
Monitoring events, logs, and database activity streams ... 1595
Viewing logs, events, and streams in the Amazon RDS console ... 1596
Monitoring RDS events ... 1599
Overview of events for Amazon RDS ... 1599
Viewing Amazon RDS events ... 1601
Working with Amazon RDS event notification ... 1605
Creating a rule that triggers on an Amazon RDS event ... 1631
Amazon RDS event categories and event messages ... 1636
Monitoring RDS logs ... 1684
Viewing and listing database log files ... 1684
Downloading a database log file ... 1685
Watching a database log file ... 1687
Publishing to CloudWatch Logs ... 1688
Reading log file contents using REST ... 1691
Db2 database log files ... 1693
MariaDB database log files ... 1698
Microsoft SQL Server database log files ... 1712
MySQL database log files ... 1717
Oracle database log files ... 1733
PostgreSQL database log files ... 1744
Monitoring RDS API calls in CloudTrail ... 1758

CloudTrail integration with Amazon RDS ... 1758
Amazon RDS log file entries ... 1759
Monitoring RDS with Database Activity Streams ... 1763
Overview ... 1763
Configuring Oracle unified auditing ... 1770
Configuring SQL Server auditing ... 1771
Starting a database activity stream ... 1772
Modifying a database activity stream ... 1775
Getting the activity stream status ... 1778
Stopping a database activity stream ... 1779
Monitoring activity streams ... 1780
IAM policy examples for activity streams ... 1822
Monitoring threats with GuardDuty RDS Protection ... 1825
Amazon RDS Custom ... 1826
Database customization challenge ... 1826
RDS Custom management model and benefits ... 1828
Shared responsibility model in RDS Custom ... 1828
Support perimeter and unsupported configurations in RDS Custom ... 1831
Key benefits of RDS Custom ... 1831
RDS Custom architecture ... 1832
VPC ... 1832
RDS Custom automation and monitoring ... 1833
Amazon S3 ... 1837
AWS CloudTrail ... 1838
RDS Custom security ... 1840
How RDS Custom securely manages tasks on your behalf ... 1840
SSL certificates ... 1841
Secure your Amazon S3 bucket against the confused deputy problem ... 1841
Rotating credentials for RDS Custom for Oracle ... 1842
Working with RDS Custom for Oracle ... 1848
RDS Custom for Oracle workflow ... 1848
Database architecture for Amazon RDS Custom for Oracle ... 1853
Feature availability and support for RDS Custom for Oracle ... 1855
RDS Custom for Oracle requirements and limitations ... 1858
Setting up your RDS Custom for Oracle environment ... 1862
Working with CEVs for RDS Custom for Oracle ... 1882

Configuring an RDS Custom for Oracle DB instance ... 1914
Managing an RDS Custom for Oracle DB instance ... 1933
Managing a Multi-AZ deployment for RDS Custom for Oracle ... 1951
Working with RDS Custom for Oracle replicas ... 1972
Backing up and restoring an RDS Custom for Oracle DB instance ... 1982
Working with option groups in RDS Custom for Oracle ... 1993
Migrating to RDS Custom for Oracle ... 2002
Upgrading an RDS Custom for Oracle DB instance ... 2003
Troubleshooting RDS Custom for Oracle ... 2016
Known issues for RDS Custom for Oracle ... 2042
Working with RDS Custom for SQL Server ... 2046
RDS Custom for SQL Server workflow ... 2046
RDS Custom for SQL Server requirements and limitations ... 2049
Setting up your RDS Custom for SQL Server environment ... 2129
Bring Your Own Media with RDS Custom for SQL Server ... 2154
Working with CEVs for RDS Custom for SQL Server ... 2156
Creating and connecting to an RDS Custom for SQL Server DB instance ... 2178
Managing an RDS Custom for SQL Server DB instance ... 2190
Working with Microsoft Active Directory with RDS Custom for SQL Server ... 2204
Managing a Multi-AZ deployment for RDS Custom for SQL Server ... 2230
Backing up and restoring an RDS Custom for SQL Server DB instance ... 2246
Copying an RDS Custom for SQL Server DB snapshot ... 2263
Migrating an on-premises database to RDS Custom for SQL Server ... 2274
Operating system updates ... 2277
Upgrading an RDS Custom for SQL Server DB instance ... 2280
Troubleshooting Amazon RDS Custom for SQL Server ... 2284
Amazon RDS on AWS Outposts ... 2324
Prerequisites ... 2325
Support for Amazon RDS features ... 2326
Supported DB instance classes ... 2333
Customer-owned IP addresses ... 2335
Using CoIPs ... 2335
Limitations ... 2337
Multi-AZ deployments ... 2338
Working with the shared responsibility model ... 2338
Improving availability ... 2338

Prerequisites ... 2339
Working with API operations for Amazon EC2 permissions ... 2340
Creating DB instances for RDS on Outposts ... 2342
Creating read replicas for RDS on Outposts ... 2352
Considerations for restoring DB instances ... 2355
Amazon RDS Proxy ... 2356
Region and version availability ... 2357
Quotas and limitations ... 2358
RDS for MariaDB limitations ... 2359
RDS for SQL Server limitations ... 2360
MySQL limitations ... 2361
PostgreSQL limitations ... 2362
Planning where to use RDS Proxy ... 2363
RDS Proxy concepts and terminology ... 2364
Overview of RDS Proxy concepts ... 2365
Connection pooling ... 2366
Security ... 2367
Failover ... 2369
Transactions ... 2370
Getting started with RDS Proxy ... 2371
Set up a proxy network ... 2371
Setting up database credentials ... 2374
Configuring IAM authentication ... 2378
Creating a proxy ... 2385
Viewing a proxy ... 2398
Connecting through RDS Proxy ... 2399
Managing an RDS Proxy ... 2404
Modifying an RDS Proxy ... 2404
Adding a database user ... 2411
Moving to end-to-end IAM authentication ... 2413
RDS Proxy connection considerations ... 2416
Avoid pinning RDS Proxy ... 2420
Deleting an RDS Proxy ... 2426
Working with RDS Proxy endpoints ... 2427
Overview of proxy endpoints ... 2428
Limitations for proxy endpoints ... 2428

Proxy endpoints for Multi-AZ DB clusters ... 2429
Accessing RDS databases across VPCs ... 2430
Creating a proxy endpoint ... 2432
Viewing proxy endpoints ... 2435
Modifying a proxy endpoint ... 2436
Deleting a proxy endpoint ... 2437
Monitoring RDS Proxy with CloudWatch ... 2439
Working with RDS Proxy events ... 2447
RDS Proxy events ... 2447
Troubleshooting RDS Proxy ... 2450
Verifying connectivity for a proxy ... 2450
Common issues and solutions ... 2452
Troubleshooting for RDS for MySQL ... 2454
Troubleshooting for RDS for PostgreSQL ... 2455
Using RDS Proxy with AWS CloudFormation ... 2462
Zero-ETL integrations ... 2464
Benefits ... 2466
Key concepts ... 2466
Limitations ... 2467
General limitations ... 2467
RDS for MySQL limitations ... 2468
RDS for PostgreSQL limitations ... 2468
RDS for Oracle limitations ... 2469
Amazon Redshift limitations ... 2469
Amazon SageMaker AI lakehouse limitations ... 2469
Quotas ... 2470
Supported Regions ... 2470
Getting started with zero-ETL integrations ... 2470
Step 1: Create a custom DB parameter group ... 2471
Step 2: Select or create a source database ... 2473
Step 3a: Create a target data warehouse ... 2473
Set up an integration using the AWS SDKs ... 2474
Step 3b: Create an AWS Glue catalog for Amazon SageMaker Lakehouse zero-ETL integration ... 2479
Next steps ... 2483
Creating zero-ETL integrations with Amazon Redshift ... 2483

Prerequisites ... 2483
Required permissions ... 2483
Creating zero-ETL integrations ... 2486
Encrypting integrations ... 2491
Next steps ... 2493
Creating zero-ETL integrations with an Amazon SageMaker lakehouse ... 2493
Prerequisites ... 2493
Required permissions ... 2493
Creating zero-ETL integrations with an Amazon SageMaker lakehouse ... 2496
Encrypting integrations ... 2501
Next steps ... 2503
Data filtering for zero-ETL integrations ... 2504
Format of a data filter ... 2505
Filter logic ... 2507
Filter precedence ... 2508
RDS for MySQL examples ... 2508
RDS for PostgreSQL examples ... 2509
RDS for Oracle examples ... 2510
Adding data filters ... 2514
Removing data filters ... 2516
Adding and querying data ... 2516
Creating a target database ... 2517
Adding data to the source database ... 2517
Querying your Amazon RDS data in Amazon Redshift ... 2519
Data type differences ... 2520
DDL operations for RDS for PostgreSQL ... 2537
Viewing and monitoring zero-ETL integrations ... 2540
Viewing integrations ... 2540
Monitoring using system tables ... 2542
Monitoring with EventBridge ... 2543
Modifying zero-ETL integrations ... 2543
Deleting zero-ETL integrations ... 2544
Troubleshooting zero-ETL integrations ... 2545
I can't create a zero-ETL integration ... 2546
My integration is stuck in a state of `Syncing` ... 2547
My tables aren't replicating to Amazon Redshift ... 2547

One or more of my Amazon Redshift tables requires a resync ... 2547
Integration failed issues for Amazon SageMaker AI lakehouse zero-ETL integrations ... 2550
Db2 on Amazon RDS ... 2551
Db2 overview ... 2552
Db2 features ... 2553
Db2 versions ... 2556
Db2 licensing ... 2561
Db2 instance classes ... 2577
Db2 default roles ... 2580
Db2 parameters ... 2580
EBCDIC collation ... 2587
Db2 local time zone ... 2587
DB instance prerequisites ... 2590
Administrator account ... 2590
Additional considerations ... 2590
Multiple Db2 databases ... 2592
Connecting to your Db2 DB instance ... 2594
Finding the endpoint ... 2594
IBM Db2 CLP ... 2596
IBM CLPPlus ... 2601
DBeaver ... 2604
IBM Db2 Data Management Console ... 2608
Security group considerations ... 2617
Securing Db2 connections ... 2618
Encrypting with SSL/TLS ... 2618
Using Kerberos authentication ... 2625
Administering your RDS for Db2 DB instance ... 2642
System tasks ... 2644
Database tasks ... 2656
Integrating with S3 ... 2684
Create an IAM policy ... 2684
Create an IAM role and attach your IAM policy ... 2689
Add your IAM role to your DB instance ... 2691
Migrating data to RDS for Db2 ... 2694
Migrating data with AWS services ... 2694
Migrating data with native Db2 tools ... 2706

Federation ... 2719
Homogeneous federation ... 2719
Heterogeneous federation ... 2724
Working with Db2 replicas ... 2729
Read-only and standby replicas ... 2729
Database activations ... 2730
HADR configurations ... 2730
Archive log retention ... 2730
Outages during Db2 replication ... 2731
Requirements and considerations for Db2 replicas ... 2731
Preparing to create a Db2 replica ... 2734
Creating a standby Db2 replica ... 2735
Modifying the replica mode ... 2737
Working with Db2 replica backups ... 2740
Troubleshooting Db2 replication issues ... 2743
Options for RDS for Db2 DB instances ... 2747
Db2 audit logging ... 2748
External stored procedures ... 2763
Java-based external stored procedures ... 2763
Known issues and limitations ... 2772
Authentication limitation ... 2772
Non-fenced routines ... 2772
Non-automatic storage tablespaces during migration ... 2772
Setting the db2_compatibility_vector parameter ... 2773
Migrating databases that contain INVALID packages ... 2773
RDS for Db2 stored procedures ... 2775
Considerations for stored procedures ... 2784
Granting and revoking privileges ... 2785
Audit policies ... 2803
Buffer pools ... 2808
Databases ... 2814
Storage access ... 2857
Tablespaces ... 2860
RDS for Db2 user-defined functions ... 2871
rdsadmin.get_task_status ... 2871
rdsadmin.list_databases ... 2876

rdsadmin.list_modifiable_db_params ... 2878
Troubleshooting ... 2880
Database connection error ... 2880
File I/O error ... 2880
Stored procedure errors ... 2884
MariaDB on Amazon RDS ... 2897
MariaDB feature support ... 2899
MariaDB major versions ... 2899
Supported storage engines ... 2908
Cache warming ... 2910
Features not supported ... 2911
MariaDB versions ... 2913
Supported MariaDB minor versions ... 2913
Supported MariaDB major versions ... 2922
The Database Preview environment ... 2923
MariaDB version 11.7 in the Database Preview environment ... 2927
Deprecated MariaDB versions ... 2927
Connecting to a DB instance running MariaDB ... 2928
Finding the connection information ... 2929
Connecting from the command-line client ... 2932
Connecting with the AWS drivers ... 2933
Troubleshooting ... 2934
Securing MariaDB connections ... 2935
MariaDB security ... 2935
Password validation plugins ... 2937
Encrypting with SSL/TLS ... 2938
Using new SSL/TLS certificates ... 2942
Improving query performance with RDS Optimized Reads ... 2947
Overview ... 2947
Use cases ... 2948
Best practices ... 2948
Using ... 2949
Monitoring ... 2950
Limitations ... 2950
Improving write performance with RDS Optimized Writes for MariaDB ... 2952
Overview ... 2952

Using with a new database ... 2953
Enabling on an existing database ... 2958
Limitations ... 2959
Upgrades of the MariaDB DB engine ... 2960
Considerations ... 2961
Finding valid upgrade targets ... 2962
MariaDB version numbers ... 2963
RDS version numbers ... 2965
Major version upgrades ... 2966
Upgrading a MariaDB DB instance ... 2967
Automatic minor version upgrades ... 2967
Upgrading with reduced downtime ... 2970
Monitoring with events ... 2973
Upgrading a MariaDB DB snapshot engine version ... 2974
Upgrade options for unsupported engine versions ... 2976
Importing data into an RDS for MariaDB DB instance ... 2978
Importing data considerations ... 2982
Importing data from an external database ... 2987
Importing data with reduced downtime ... 2991
Importing data from any source ... 3009
MariaDB replication ... 3016
MariaDB read replicas ... 3017
Configuring GTID-based replication with an external source instance ... 3031
Configuring binary log file position replication with an external source instance ... 3035
Options for MariaDB ... 3041
MariaDB Audit Plugin support ... 3041
Parameters for MariaDB ... 3047
Viewing MariaDB parameters ... 3047
MySQL parameters that aren't available ... 3049
Migrating data from a MySQL DB snapshot to a MariaDB DB instance ... 3052
Performing the migration ... 3052
Incompatibilities between MariaDB and MySQL ... 3054
MariaDB on Amazon RDS SQL reference ... 3056
mysql.rds_replica_status ... 3056
mysql.rds_set_external_master_gtid ... 3058
mysql.rds_kill_query_id ... 3061

mysql.rds_execute_operation ... 3062
Local time zone ... 3065
Known issues and limitations for MariaDB ... 3067
File size limits ... 3067
InnoDB reserved word ... 3069
Custom ports ... 3069
Performance Insights ... 3069
Microsoft SQL Server on Amazon RDS ... 3070
Common management tasks ... 3072
Limitations ... 3074
DB instance class support ... 3078
Optimize CPU ... 3083
Limitations ... 3085
DB instance class support ... 3086
Set CPU cores and threads ... 3088
Security ... 3092
SSL support ... 3093
Using SSL with a SQL Server DB instance ... 3094
Configuring SQL Server security protocols and ciphers ... 3100
Updating applications for new SSL/TLS certificates ... 3107
Compliance programs ... 3111
HIPAA ... 3111
Version support ... 3112
Version management ... 3118
Feature support ... 3121
SQL Server 2022 features ... 3121
SQL Server 2019 features ... 3122
SQL Server 2017 features ... 3123
SQL Server 2016 features ... 3123
SQL Server 2014 features ... 3123
SQL Server 2012 end of support on Amazon RDS ... 3124
SQL Server 2008 R2 end of support on Amazon RDS ... 3124
CDC support ... 3124
Unsupported and limited feature support ... 3125
Multi-AZ deployments ... 3126
Using TDE ... 3127

Functions and stored procedures ... 3127
Local time zone ... 3134
Supported time zones ... 3135
Licensing SQL Server on Amazon RDS ... 3149
Restoring license-terminated DB instances ... 3149
SQL Server Developer Edition ... 3150
Connecting to a DB instance running SQL Server ... 3151
Before you connect ... 3151
Finding the DB instance endpoint and port number ... 3152
Connecting to your DB instance with SSMS ... 3153
Connecting to your DB instance with SQL Workbench/J ... 3156
Security group considerations ... 3158
Troubleshooting ... 3158
SQL Server Developer Edition ... 3161
Benefits ... 3161
Regions ... 3161
Licensing and usage ... 3162
Prerequisites ... 3162
Limitations ... 3163
Preparing a CEV ... 3164
Creating a custom engine version ... 3165
Creating a Developer Edition DB instance ... 3170
Applying database minor version upgrades ... 3171
Viewing and managing custom engine versions ... 3172
Troubleshooting SQL Server Developer Edition for RDS for SQL Server ... 3174
Working with Active Directory with RDS for SQL Server ... 3176
Working with self-managed Active Directory ... 3177
Working with AWS Managed Active Directory with RDS for SQL Server ... 3200
Upgrades of the SQL Server DB engine ... 3216
Major version upgrades ... 3217
Upgrade considerations ... 3219
Testing an upgrade ... 3222
Upgrading a SQL server DB instance ... 3223
Upgrading deprecated DB instances before support ends ... 3224
Working with SQL Server storage ... 3225
Considerations ... 3225

Add, remove, modify storage ... 3226
Restore operations ... 3228
Use cases ... 3229
Importing and exporting SQL Server databases ... 3233
Limitations and recommendations ... 3235
Setting up ... 3237
Using native backup and restore ... 3245
Compressing backup files ... 3264
Troubleshooting ... 3265
Importing and exporting SQL Server data using other methods ... 3268
Using BCP from Linux ... 3281
SQL Server read replicas ... 3287
Configuring read replicas for SQL Server ... 3287
Read replica limitations with SQL Server ... 3288
Option considerations ... 3289
Synchronizing database users and objects ... 3290
Troubleshooting ... 3292
Multi-AZ for RDS for SQL Server ... 3294
Adding Multi-AZ to a SQL Server DB instance ... 3296
Removing Multi-AZ from a SQL Server DB instance ... 3296
Limitations, notes, and recommendations ... 3297
Determining the location of the secondary ... 3301
Migrating to Always On AGs ... 3301
Additional features for SQL Server ... 3303
Using password policy with a SQL Server DB instance ... 3304
Amazon S3 integration ... 3311
Using Database Mail ... 3332
Instance store support for tempdb ... 3348
Using extended events ... 3351
Access to transaction log backups ... 3355
Options for SQL Server ... 3391
Listing the available options for SQL Server versions and editions ... 3393
Linked Servers with Oracle OLEDB ... 3396
Linked Servers with Teradata ODBC ... 3409
Native backup and restore ... 3418
Transparent Data Encryption ... 3423

SQL Server Audit ... 3436
SQL Server Analysis Services ... 3446
SQL Server Integration Services ... 3475
SQL Server Reporting Services ... 3498
Microsoft Distributed Transaction Coordinator ... 3518
SQL Server resource governor ... 3536
Common DBA tasks ... 3565
Accessing the tempdb database ... 3567
Analyzing database workload with Database Engine Tuning Advisor ... 3571
Changing the db_owner to the rdsa account for your database ... 3575
Managing collations and character sets ... 3576
Creating a database user ... 3583
Determining a recovery model ... 3584
Determining the last failover time ... 3585
Troubleshoot PITR failures due to LSN gaps ... 3586
Deny or allow viewing database names ... 3587
Disabling fast inserts ... 3588
Dropping a SQL Server database ... 3589
Renaming a Multi-AZ database ... 3589
Resetting the db_owner role membership for master user ... 3590
Restoring license-terminated DB instances ... 3591
Transitioning a database from OFFLINE to ONLINE ... 3592
Using CDC ... 3592
Using SQL Server Agent ... 3595
Working with SQL Server logs ... 3602
Working with trace and dump files ... 3604
MySQL on Amazon RDS ... 3606
MySQL feature support ... 3609
MySQL major versions ... 3609
Supported storage engines ... 3611
Using memcached and other options ... 3612
InnoDB cache warming ... 3612
Inclusive language changes for MySQL 8.4 ... 3614
Features not supported ... 3617
MySQL versions ... 3618
Minor versions ... 3618

Major versions ... 3626
RDS Extended Support versions ... 3627
Database Preview environment ... 3634
9.5 (preview) ... 3637
9.4 (preview) ... 3637
9.3 (preview) ... 3637
Deprecated versions ... 3637
Connecting to a DB instance running MySQL ... 3638
Finding the connection information ... 3639
Installing the command-line client ... 3643
Connecting from the command-line client ... 3643
Connecting from MySQL Workbench ... 3644
Connecting with the AWS drivers ... 3646
Troubleshooting ... 3647
Securing MySQL connections ... 3648
Password validation ... 3650
Encrypting with SSL/TLS ... 3652
Using new SSL/TLS certificates ... 3656
Using Kerberos authentication for MySQL ... 3661
Improving query performance with RDS Optimized Reads ... 3676
Overview ... 3676
Use cases ... 3677
Best practices ... 3677
Using ... 3678
Monitoring ... 3679
Limitations ... 3679
Improving write performance with RDS Optimized Writes for MySQL ... 3681
Overview ... 2952
Using with a new database ... 3682
Enabling on an existing database ... 3687
Limitations ... 3688
Upgrades of the MySQL DB engine ... 3689
Considerations ... 3690
Finding valid upgrade targets ... 3691
MySQL version numbers ... 3692
RDS version numbers ... 3694

Major version upgrades ... 3694
Testing an upgrade ... 3701
Upgrading a MySQL DB instance ... 3702
Automatic minor version upgrades ... 3702
Upgrading with reduced downtime ... 3705
Monitoring with events ... 3708
Upgrading a MySQL DB snapshot engine version ... 3709
Upgrade options for unsupported engine versions ... 3711
Importing data into an RDS for MySQL DB instance ... 3714
Importing data considerations ... 3718
Restoring a backup into a MySQL DB instance ... 3723
Importing data from an external database ... 3739
Importing data with reduced downtime ... 3743
Importing data from any source ... 3761
MySQL replication ... 3768
MySQL read replicas ... 3769
GTID-based replication ... 3785
Configuring binary log file position replication with an external source instance ... 3793
Configuring multi-source replication ... 3798
Configuring active-active clusters ... 3806
Use cases ... 3806
Limitations and considerations for active-active clusters ... 3807
Preparing for a cross-VPC active-active cluster ... 3810
Required parameter settings for active-active clusters ... 3812
Converting a DB instance to an active-active cluster ... 3815
Setting up a new active-active cluster ... 3821
Adding a DB instance to an active-active cluster ... 3827
Monitoring active-active clusters ... 3830
Stopping Group Replication in an active-active cluster ... 3831
Renaming a DB instance in an active-active cluster ... 3832
Removing a DB instance from an active-active cluster ... 3832
Exporting data from a MySQL DB instance ... 3834
Prepare an external MySQL database ... 3834
Prepare the source MySQL DB instance ... 3835
Copy the database ... 3837
Complete the export ... 3838

Options for MySQL ... 3840
MariaDB Audit Plugin ... 3841
memcached ... 3848
Parameters for MySQL ... 3854
Common DBA tasks for MySQL ... 3856
Understanding predefined users ... 3856
Role-based privilege model ... 3856
Dynamic privileges ... 3860
Ending a session or query ... 3864
Skipping the current replication error ... 3864
Improve crash recovery times ... 3866
Managing the Global Status History ... 3869
Configuring buffer pool size and redo log capacity ... 3871
Local time zone ... 3873
Known issues and limitations ... 3875
InnoDB reserved word ... 3875
Storage-full behavior ... 3875
Inconsistent InnoDB buffer pool size ... 3876
Index merge optimization returns incorrect results ... 3877
MySQL parameter exceptions for Amazon RDS DB instances ... 3878
MySQL file size limits in Amazon RDS ... 3879
MySQL Keyring Plugin not supported ... 3881
Custom ports ... 3881
MySQL stored procedure limitations ... 3881
GTID-based replication with an external source instance ... 3882
MySQL default authentication plugin ... 3882
Overriding innodb_buffer_pool_size ... 3882
Upgrading from MySQL 5.7 to MySQL 8.4 ... 3883
InnoDB page compression ... 3883
RDS for MySQL stored procedures ... 3884
Collecting and maintaining the Global Status History ... 3885
Configuring, starting, and stopping binary log (binlog) replication ... 3888
Ending a session or query ... 3930
Managing active-active clusters ... 3932
Managing multi-source replication ... 3937
Replicating transactions using GTIDs ... 3959

Rotating the query logs ... 3962
Setting and showing binary log configuration ... 3964
Warming the InnoDB cache ... 3968
Oracle on Amazon RDS ... 3970
Oracle overview ... 3971
Oracle features ... 3972
Oracle versions ... 3976
Oracle licensing ... 3982
Oracle users and privileges ... 3986
Oracle instance classes ... 3987
Oracle database architecture ... 3998
Oracle parameters ... 3999
Oracle character sets ... 4001
Oracle limitations ... 4006
Connecting to your Oracle DB instance ... 4009
Finding the endpoint ... 4009
SQL developer ... 4011
SQL*Plus ... 4014
Security group considerations ... 4015
Dedicated and shared server processes ... 4016
Troubleshooting ... 4016
Modifying Oracle sqlnet.ora parameters ... 4018
Securing Oracle connections ... 4023
Encrypting with SSL ... 4023
Using new SSL/TLS certificates ... 4024
Encrypting with NNE ... 4028
Configuring Kerberos authentication ... 4028
Configuring UTL_HTTP access ... 4047
Working with CDBs ... 4060
Overview of CDBs ... 4060
Configuring a CDB ... 4066
Backing up and restoring a CDB ... 4073
Converting a non-CDB to a CDB ... 4074
Converting single-tenant to multi-tenant ... 4076
Adding an RDS for Oracle tenant database to your CDB instance ... 4079
Modifying an RDS for Oracle tenant database ... 4082

Deleting an RDS for Oracle tenant database from your CDB 4085
Viewing tenant database details 4087
Upgrading your CDB 4092
Administering your Oracle DB instance 4093
System tasks 4108
Database tasks 4133
Checkpointing a database 4151
Setting distributed recovery 4151
Setting the database time zone 4151
Working with AWR 4153
Adjusting database links for use with DB instances in a VPC 4158
Setting the default edition 4159
Enabling auditing for the SYS.AUD$ table 4160
Disabling auditing for the SYS.AUD$ table 4161
Cleaning up interrupted online index builds 4161
Skipping corrupt blocks 4162
Purging the recycle bin 4165
Setting the default displayed values for full redaction 4166
Log tasks 4168
RMAN tasks 4181
Oracle Scheduler tasks 4215
Diagnosing problems 4224
Other tasks 4234
Working with Oracle storage 4249
Considerations 4250
Limitations 4251
Database management operations 4251
Modify storage volumes 4252
Backup and restore storage volumes 4254
Use cases 4256
Configuring advanced RDS for Oracle features 4259
Configuring the instance store 4259
Turning on HugePages 4270
Turning on extended data types 4273
Importing data into Oracle 4277
Importing using Oracle SQL Developer 4278

Migrating using Oracle transportable tablespaces ... 4278
Importing using Oracle Data Pump ... 4295
Importing using Oracle Export/Import ... 4311
Importing using Oracle SQL*Loader ... 4312
Migrating with Oracle materialized views ... 4314
Working with Oracle replicas ... 4317
Overview of Oracle replicas ... 4317
Requirements and considerations for Oracle replicas ... 4319
Preparing to create an Oracle replica ... 4324
Creating a mounted Oracle replica ... 4326
Modifying the replica mode ... 4327
Working with Oracle replica backups ... 4328
Performing an Oracle Data Guard switchover ... 4331
Troubleshooting Oracle replicas ... 4339
Redo transport compression ... 4340
Options for Oracle ... 4342
Overview of Oracle DB options ... 4342
Amazon S3 integration ... 4345
Oracle Application Express (APEX) ... 4372
Amazon EFS integration ... 4396
Java virtual machine (JVM) ... 4412
Enterprise Manager ... 4416
Label security ... 4442
Locator ... 4445
Native network encryption (NNE) ... 4449
OLAP ... 4464
Secure Sockets Layer (SSL) ... 4467
Spatial ... 4481
SQLT ... 4485
Statspack ... 4494
Time zone ... 4498
Time zone file autoupgrade ... 4503
Transparent Data Encryption (TDE) ... 4513
UTL_MAIL ... 4518
XML DB ... 4522
Upgrading the Oracle DB engine ... 4523

Overview of Oracle upgrades ... 4523
Major version upgrades ... 4529
Minor version upgrades ... 4530
Upgrade considerations ... 4539
Testing an upgrade ... 4542
Upgrading an RDS for Oracle DB instance ... 4544
Upgrading an Oracle DB snapshot ... 4546
Tools and third-party software for Oracle ... 4549
Using Oracle GoldenGate ... 4550
Using the Oracle Repository Creation Utility ... 4568
Configuring CMAN ... 4576
Installing a Siebel database on Oracle on Amazon RDS ... 4579
Oracle Database engine releases ... 4584
PostgreSQL on Amazon RDS ... 4585
Common management tasks ... 4587
Working with the Database Preview environment ... 4591
Features not supported in the Database Preview environment ... 4592
PostgreSQL version 17 in the Database Preview environment ... 4592
Creating a new DB instance in the Database Preview environment ... 4592
PostgreSQL versions ... 4595
Deprecated PostgreSQL versions ... 4595
RDS for PostgreSQL release process ... 4596
Advantages of RDS for PostgreSQL incremental release process ... 4596
Managing release updates ... 4596
PostgreSQL extension versions ... 4598
Restricting installation of PostgreSQL extensions ... 4598
PostgreSQL trusted extensions ... 4600
PostgreSQL features ... 4602
Custom data types and enumerations ... 4603
Event triggers ... 4603
Huge pages ... 4604
Performing logical replication ... 4605
Configuring IAM authentication for logical replication ... 4612
RAM disk for the stats_temp_directory ... 4615
Tablespaces ... 4616
Collations for EBCDIC and other mainframe migrations ... 4617

Managing logical slot synchronization ... 4622
Connecting to a PostgreSQL instance ... 4624
Installing the psql client ... 4625
Finding the connection information ... 4625
Using pgAdmin to connect to a RDS for PostgreSQL DB instance ... 4627
Using psql to connect to your RDS for PostgreSQL DB instance ... 4629
Connecting to RDS for PostgreSQL with the AWS JDBC Driver ... 4631
Connecting to RDS for PostgreSQL with the AWS Python Driver ... 4631
Troubleshooting connections to your RDS for PostgreSQL instance ... 4631
Securing connections with SSL/TLS ... 4634
Using SSL with a PostgreSQL DB instance ... 4634
Updating applications to use new SSL/TLS certificates ... 4642
Using Kerberos authentication ... 4647
Region and version availability ... 4648
Overview of Kerberos authentication ... 4648
Setting up ... 4649
Managing an RDS for PostgreSQL DB instance in an Active Directory domain ... 4663
Connecting with Kerberos authentication ... 4664
Using a custom DNS server for outbound network access ... 4668
Turning on custom DNS resolution ... 4668
Turning off custom DNS resolution ... 4668
Setting up a custom DNS server ... 4668
Upgrades of the PostgreSQL DB engine ... 4671
Considerations ... 4673
Finding valid upgrade targets ... 4674
PostgreSQL version numbers ... 4675
RDS version numbers ... 4675
Choosing a major version upgrade ... 4676
How to perform a major version upgrade ... 4685
Automatic minor version upgrades ... 4695
Upgrading PostgreSQL extensions ... 4698
Monitoring with events ... 4698
Upgrading a PostgreSQL DB snapshot engine version ... 4700
RDS for PostgreSQL read replicas ... 4703
Read replica limitations with PostgreSQL ... 4703
Read replica configuration with PostgreSQL ... 4704

Logical decoding on a read replica ... 4706
Using cascading read replicas ... 4710
Creating cross-Region cascading read replicas ... 4711
How replication works for different RDS for PostgreSQL versions ... 4713
Monitoring and tuning the replication process ... 4717
Configuring delayed replication with RDS for PostgreSQL ... 4720
Troubleshooting for RDS for PostgreSQL read replica ... 4724
Improving query performance with RDS Optimized Reads ... 4726
Overview of RDS Optimized Reads in PostgreSQL ... 4726
Use cases ... 4727
Best practices ... 4728
Using ... 4728
Monitoring ... 4729
Limitations ... 4729
Importing data into PostgreSQL ... 4730
Importing a PostgreSQL database from an Amazon EC2 instance ... 4732
Using the \copy command to import data to a table on a PostgreSQL DB instance ... 4734
Importing data from Amazon S3 into RDS for PostgreSQL ... 4736
Transporting PostgreSQL databases between DB instances ... 4755
Exporting PostgreSQL data to Amazon S3 ... 4764
Installing the extension ... 4765
Overview of exporting to S3 ... 4766
Specifying the Amazon S3 file path to export to ... 4767
Setting up access to an Amazon S3 bucket ... 4768
Exporting query data using the aws_s3.query_export_to_s3 function ... 4773
Function reference ... 4776
Troubleshooting access to Amazon S3 ... 4780
Invoking a Lambda function from RDS for PostgreSQL ... 4781
Step 1: Configure outbound connections ... 4782
Step 2: Configure IAM for your instance and Lambda ... 4783
Step 3: Install the extension ... 4784
Step 4: Use Lambda helper functions ... 4785
Step 5: Invoke a Lambda function ... 4786
Step 6: Grant users permissions ... 4788
Examples: Invoking Lambda functions ... 4788
Lambda function error messages ... 4791

Lambda function and parameter reference ... 4792
Common DBA tasks for RDS for PostgreSQL ... 4798
Collations supported in RDS for PostgreSQL ... 4799
Understanding PostgreSQL roles and permissions ... 4799
Dead connection handling in PostgreSQL ... 4815
Working with PostgreSQL autovacuum ... 4817
Managing TOAST OID contention ... 4864
Logging mechanisms ... 4872
Managing temporary files with PostgreSQL ... 4873
Using pgBadger for log analysis with PostgreSQL ... 4879
Using PGSnapper for monitoring PostgreSQL ... 4879
Managing custom casts ... 4879
Working with parameters ... 4883
Tuning with wait events for RDS for PostgreSQL ... 4904
Essential concepts for RDS for PostgreSQL tuning ... 4905
RDS for PostgreSQL wait events ... 4910
Client:ClientRead ... 4912
Client:ClientWrite ... 4915
CPU ... 4917
IO:BufFileRead and IO:BufFileWrite ... 4923
IO:DataFileRead ... 4932
IO:WALWrite ... 4940
IPC:parallel wait events ... 4943
IPC:ProcArrayGroupUpdate ... 4949
Lock:advisory ... 4951
Lock:extend ... 4954
Lock:Relation ... 4957
Lock:transactionid ... 4960
Lock:tuple ... 4963
LWLock:BufferMapping (LWLock:buffer_mapping) ... 4967
LWLock:BufferIO (IPC:BufferIO) ... 4969
LWLock:buffer_content (BufferContent) ... 4971
LWLock:lock_manager (LWLock:lockmanager) ... 4973
LWLock:pg_stat_statements ... 4979
LWLock:SubtransSLRU (LWLock:SubtransControlLock) ... 4983
Timeout:PgSleep ... 4990

Timeout:VacuumDelay 4990
Tuning RDS for PostgreSQL with Amazon DevOps Guru proactive insights 4994
Database has long running idle in transaction connection 4994
Using PostgreSQL extensions 4998
Using functions from orafce 4999
Using Amazon RDS delegated extension support for PostgreSQL 5001
Managing partitions with the pg_partman extension 5015
Using pgAudit to log database activity 5021
Scheduling maintenance with the pg_cron extension 5034
Using pglogical to synchronize data 5044
Using pgactive to create active-active replication 5058
Reducing bloat with the pg_repack extension 5124
Upgrading and using PLV8 5130
Using PL/Rust to write functions in the Rust language 5132
Managing spatial data with PostGIS 5138
Supported foreign data wrappers in Amazon RDS for PostgreSQL 5148
Using the log_fdw extension 5148
Using postgres_fdw to access external data 5150
Working with a MySQL database 5151
Working with an Oracle database 5155
Working with a SQL Server database 5159
Working with Trusted Language Extensions for PostgreSQL 5163
Terminology 5164
Requirements for using Trusted Language Extensions 5165
Setting up Trusted Language Extensions 5168
Overview of Trusted Language Extensions 5172
Creating TLE extensions 5173
Dropping your TLE extensions from a database 5178
Uninstalling Trusted Language Extensions 5179
Using PostgreSQL hooks with your TLE extensions 5180
Using Custom Data Types in Trusted Language Extensions 5186
Function reference for Trusted Language Extensions 5187
Hooks reference for Trusted Language Extensions 5200
Code examples 5204
Basics 5205
Hello Amazon RDS 5206

Learn the basics 5216
Actions 5335
Scenarios 5472
Create an Aurora Serverless work item tracker 5472
Serverless examples 5476
Connecting to an Amazon RDS database in a Lambda function 5477
Security 5497
Database authentication 5499
Password authentication 5500
IAM database authentication 5500
Kerberos authentication 5500
Password management with RDS and Secrets Manager 5502
Limitations 5502
Overview 5503
Benefits 5504
Permissions 5504
Enforcing RDS management 5505
Managing password for DB instance 5506
Managing password for tenant database 5510
Managing password for Multi-AZ DB cluster 5514
Rotating secrets for DB instance 5518
Rotating secrets for Multi-AZ DB clusters 5520
Viewing secret details 5522
Viewing secrets details for Multi-AZ DB clusters 5525
Viewing secrets details for tenant databases 5528
Region and version availability 5531
Data protection 5531
Data encryption 5532
Internetwork traffic privacy 5563
Identity and access management 5565
Audience 5565
Authenticating with identities 5566
Managing access using policies 5569
How Amazon RDS works with IAM 5570
Identity-based policy examples 5577
AWS managed policies 5600

Policy updates ... 5608
Cross-service confused deputy prevention ... 5627
IAM database authentication ... 5629
Troubleshooting ... 5680
Logging and monitoring ... 5682
Compliance validation ... 5684
Resilience ... 5685
Backup and restore ... 5685
Replication ... 5685
Failover ... 5686
Infrastructure security ... 5687
Security groups ... 5687
Public accessibility ... 5687
VPC endpoints (AWS PrivateLink) ... 5689
Considerations ... 1478
Availability ... 1479
Creating an interface VPC endpoint ... 1479
Creating a VPC endpoint policy ... 1479
Security best practices ... 5692
Controlling access with security groups ... 5693
Overview of VPC security groups ... 5694
Security group scenario ... 5694
Creating a VPC security group ... 5696
Associating with a DB instance ... 5697
Master user account privileges ... 5697
Service-linked roles ... 5702
Service-linked role permissions for Amazon RDS ... 5702
Service-linked role permissions for Amazon RDS Custom ... 5705
Service-linked role permissions for Amazon RDS Beta ... 5706
Service-linked role for Amazon RDS Preview ... 5707
Using Amazon RDS with Amazon VPC ... 5709
Working with a DB instance in a VPC ... 5709
Updating the VPC for a DB instance ... 5727
Scenarios for accessing a DB instance in a VPC ... 5727
Tutorial: Create a VPC for use with a DB instance (IPv4 only) ... 5734
Tutorial: Create a VPC for use with a DB instance (dual-stack mode) ... 5742

Moving a DB instance into a VPC ... 5753
Quotas and constraints ... 5756
Quotas in Amazon RDS ... 5756
Naming constraints in Amazon RDS ... 5762
Maximum number of database connections ... 5763
File size limits in Amazon RDS ... 5766
Troubleshooting ... 5767
Can't connect to DB instance ... 5767
Testing the DB instance connection ... 5770
Troubleshooting connection authentication ... 5770
Security issues ... 5771
Error message "failed to retrieve account attributes, certain console functions may be impaired." ... 5771
Troubleshooting incompatible-network state ... 5771
Causes ... 5771
Resolution ... 5772
Resetting the DB instance owner password ... 5773
DB instance outage or reboot ... 5774
Parameter changes not taking effect ... 5774
DB instance out of storage ... 5775
Insufficient DB instances available ... 5777
RDS freeable memory issues ... 5777
MySQL and MariaDB issues ... 5778
Maximum MySQL and MariaDB connections ... 5778
Diagnosing and resolving incompatible parameters status for a memory limit ... 5779
Diagnosing and resolving lag between read replicas ... 5781
Diagnosing and resolving a MySQLor MariaDB read replication failure ... 5783
Creating triggers with binary logging enabled requires SUPER privilege ... 5785
Diagnosing and resolving point-in-time restore failures ... 5786
Replication stopped error ... 5787
Read replica create fails or replication breaks with fatal error 1236 ... 5788
Read replica replication fails to initialize metadata structure ... 5788
Can't set backup retention period to 0 ... 5789
Amazon RDS API reference ... 5790
Using the Query API ... 5790
Query parameters ... 5790

Query request authentication 5791
Troubleshooting applications 5791
Retrieving errors 5791
Troubleshooting tips 5792
Document history 5793
Earlier updates 5962
AWS Glossary 5992

What is Amazon Relational Database Service (Amazon RDS)?

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the AWS Cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

> **Note**
>
> This guide covers Amazon RDS database engines other than Amazon Aurora. For information about using Amazon Aurora, see the *Amazon Aurora User Guide*.

If you are new to AWS products and services, begin learning more with the following resources:

- For an overview of all AWS products, see What is cloud computing?
- Amazon Web Services provides a number of database services. To learn more about the variety of database options available on AWS, see Choosing an AWS database service and Running databases on AWS.

Advantages of Amazon RDS

Amazon RDS is a managed database service. It's responsible for most management tasks. By eliminating tedious manual processes, Amazon RDS frees you to focus on your application and your users.

Amazon RDS provides the following principal advantages over database deployments that aren't fully managed:

- You can use database engines that you are already familiar with: IBM Db2, MariaDB, Microsoft SQL Server, MySQL, Oracle Database, and PostgreSQL.
- Amazon RDS manages backups, software patching, automatic failure detection, and recovery.
- You can turn on automated backups, or manually create your own backup snapshots. You can use these backups to restore a database. The Amazon RDS restore process works reliably and efficiently.

- You can get high availability with a primary DB instance and a synchronous secondary DB instance that you can fail over to when problems occur. You can also use read replicas to increase read scaling.
- In addition to the security in your database package, you can control access by using AWS Identity and Access Management (IAM) to define users and permissions. You can also help protect your databases by putting them in a virtual private cloud (VPC).

Comparison of responsibilities with Amazon EC2 and on-premises deployments

We recommend Amazon RDS as your default choice for most relational database deployments. The following alternatives have the disadvantage of making you spend more time managing software and hardware:

On-premises deployment

When you buy an on-premises server, you get CPU, memory, storage, and IOPS, all bundled together. You assume full responsibility for the server, operating system, and database software.

Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the AWS Cloud. Unlike in an on-premises server, CPU, memory, storage, and IOPS are separated so that you can scale them independently. AWS manages the hardware layers, which eliminates some of the burden of managing an on-premises database server.

The disadvantage to running a database on Amazon EC2 is that you're more prone to user errors. For example, when you update the operating system or database software manually, you might accidentally cause application downtime. You might spend hours checking every change to identify and fix an issue.

The following table compares the management models for on-premises databases, Amazon EC2, and Amazon RDS.

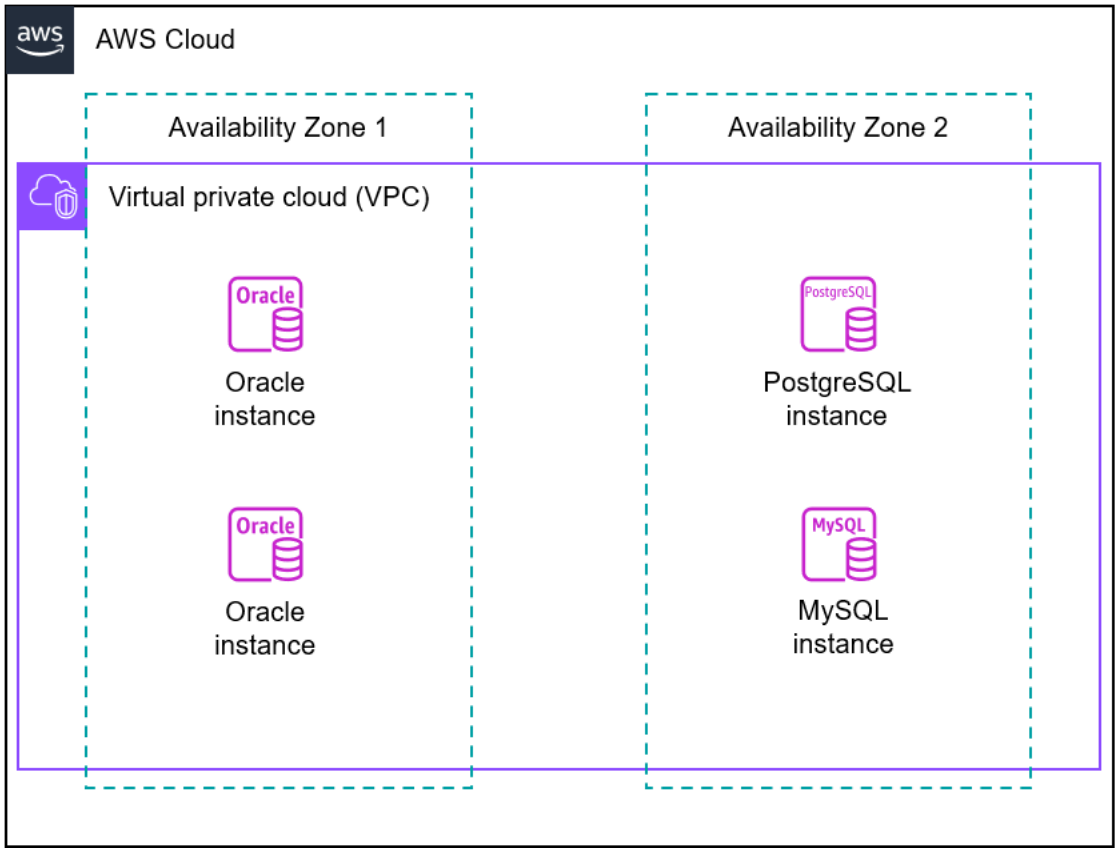
Feature	On-premises management	Amazon EC2 management	Amazon RDS management
Application optimization	Customer	Customer	Customer
Scaling	Customer	Customer	AWS
High availability	Customer	Customer	AWS
Database backups	Customer	Customer	AWS
Database software patching	Customer	Customer	AWS
Database software install	Customer	Customer	AWS
Operating system (OS) patching	Customer	Customer	AWS
OS installation	Customer	Customer	AWS
Server maintenance	Customer	AWS	AWS
Hardware lifecycle	Customer	AWS	AWS
Power, network, and cooling	Customer	AWS	AWS

Amazon RDS shared responsibility model

Amazon RDS is responsible for hosting the software components and infrastructure of DB instances and DB clusters. You are responsible for query tuning, which is the process of adjusting SQL queries to improve performance. Query performance is highly dependent on database design, data size, data distribution, application workload, and query patterns, which can vary greatly. Monitoring and tuning are highly individualized processes that you own for your RDS databases. You can use Amazon RDS Performance Insights and other tools to identify problematic queries.

Amazon RDS DB instances

A *DB instance* is an isolated database environment in the AWS Cloud. The basic building block of Amazon RDS is the DB instance. Your DB instance can contain one or more user-created databases. The following diagram shows a virtual private cloud (VPC) that contains two Availability Zones, with each AZ containing two DB instances.



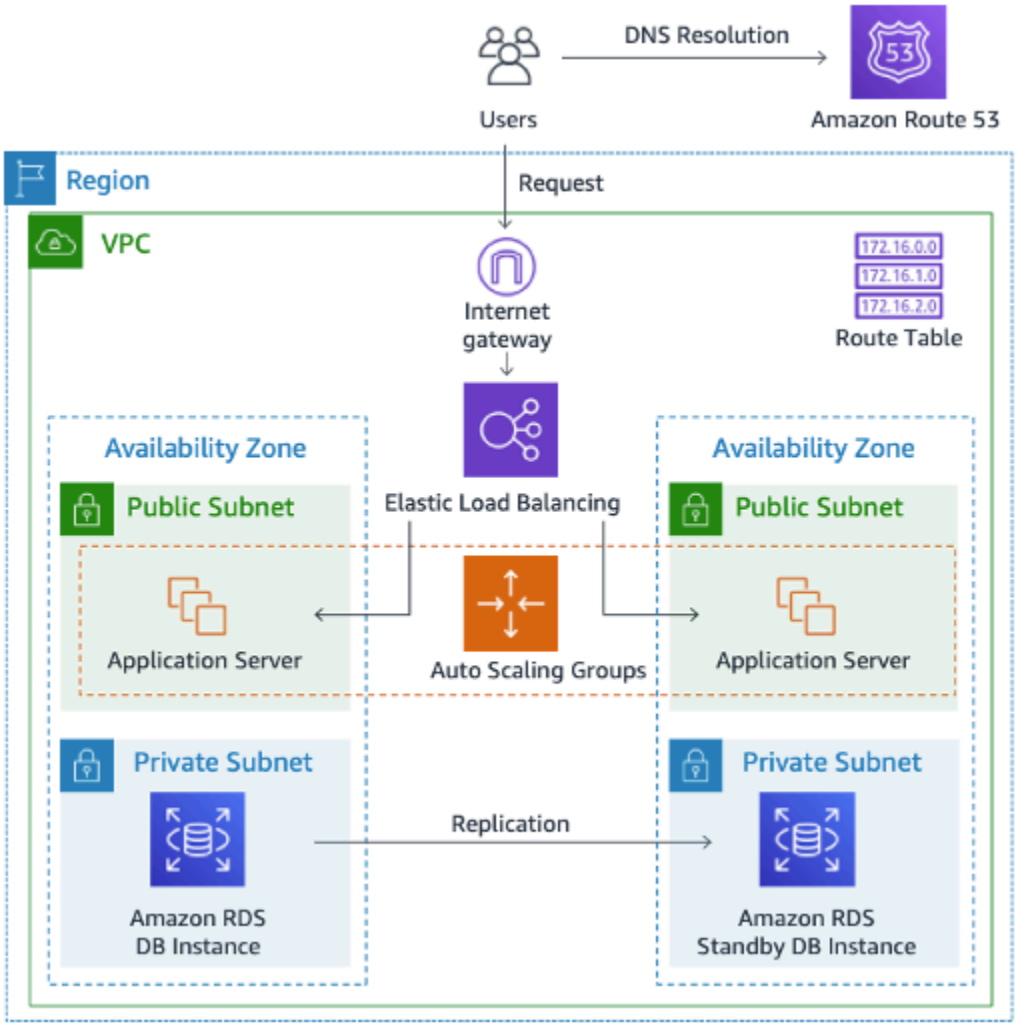
You can access your DB instances by using the same tools and applications that you use with a standalone database instance. You can create and modify a DB instance by using the AWS Command Line Interface (AWS CLI), the Amazon RDS API, or the AWS Management Console.

Topics

- Amazon RDS application architecture: example
- DB engines
- DB instance classes
- DB instance storage
- DB instances in an Amazon Virtual Private Cloud (Amazon VPC)

Amazon RDS application architecture: example

The following image shows a typical use case of a dynamic website that uses Amazon RDS DB instances for database storage:



The primary components of the preceding architecture are as follows:

Elastic Load Balancing

AWS routes user traffic through Elastic Load Balancing. A load balancer distributes workloads across multiple compute resources, such as virtual servers. In this sample use case, the Elastic Load Balancer forwards client requests to application servers.

Application servers

Application servers interact with RDS DB instances. An application server in AWS is typically hosted on EC2 instances, which provide scalable computing capacity. The application servers

reside in public subnets with different Availability Zones (AZs) within the same Virtual Private Cloud (VPC).

RDS DB instances

The EC2 application servers interact with RDS DB instances. The DB instances reside in private subnets within different Availability Zones (AZs) within the same Virtual Private Cloud (VPC). Because the subnets are private, no requests from the internet are permitted.

The primary DB instance replicates to another DB instance, called a *read replica*. Both DB instances are in private subnets within the VPC, which means that Internet users can't access them directly.

DB engines

A *DB engine* is the specific relational database software that runs on your DB instance. Amazon RDS supports the following database engines:

- IBM Db2

  For more information, see Amazon RDS for Db2.
- MariaDB

  For more information, see Amazon RDS for MariaDB.
- Microsoft SQL Server

  For more information, see Amazon RDS for Microsoft SQL Server.
- MySQL

  For more information, see Amazon RDS for MySQL.
- Oracle Database

  For more information, see Amazon RDS for Oracle.
- PostgreSQL

  For more information, see Amazon RDS for PostgreSQL.

Each DB engine has its own supported features, and each version of a DB engine can include specific features. Support for Amazon RDS features varies across AWS Regions and specific

versions of each DB engine. To check feature support in different engine versions and Regions, see Supported features in Amazon RDS by AWS Region and DB engine.

Additionally, each DB engine has a set of parameters in a DB parameter group that control the behavior of the databases that it manages. For more information about parameter groups, see Parameter groups for Amazon RDS.

DB instance classes

A *DB instance class* determines the computation and memory capacity of a DB instance. A DB instance class consists of both the DB instance class type and the size. Amazon RDS supports the following instance class types, where the asterisk (*) represents the generation, optional attribute, and size:

- General purpose – db.m*
- Memory optimized – db.z*, db.x*, db.r*
- Compute optimized – db.c*
- Burstable performance – db.t*

Each instance class offers different compute, memory, and storage capabilities. For example, db.m7g is a 7th-generation, general-purpose DB instance class type powered by AWS Graviton3 processors. When you create a DB instance, you specify a DB instance class such as db.m7g.2xlarge, where 2xlarge is the size. For more information about the hardware specifications for the different instance classes, see Hardware specifications for DB instance classes.

You can select the DB instance class that best meets your requirements. If your requirements change over time, you can change your DB instance class. For example, you might scale up your db.m7g.2xlarge instance to db.m7g.4xlarge. For more information, see DB instance classes.

> **Note**
>
> For pricing information on DB instance classes, see the Pricing section of the Amazon RDS product page.

DB instance storage

Amazon EBS provides durable, block-level storage volumes that you can attach to a running instance. DB instance storage comes in the following types:

- General Purpose (SSD)

  This cost-effective storage type is ideal for a broad range of workloads running on medium-sized DB instances. General Purpose storage is best suited for development and testing environments.
- Provisioned IOPS (PIOPS)

  This storage type is designed to meet the needs of I/O-intensive workloads, particularly database workloads, that require low I/O latency and consistent I/O throughput. Provisioned IOPS storage is best suited for production environments.
- Magnetic

  Amazon RDS supports magnetic storage for backward compatibility. We recommend that you use General Purpose SSD or Provisioned IOPS SSD for any new storage needs.

The storage types differ in performance characteristics and price. You can tailor your storage performance and cost to the requirements of your database.

Each DB instance has minimum and maximum storage requirements depending on the storage type and the database engine it supports. It's important to have sufficient storage so that your databases have room to grow. Also, sufficient storage makes sure that features for the DB engine have room to write content or log entries. For more information, see Amazon RDS DB instance storage.

DB instances in an Amazon Virtual Private Cloud (Amazon VPC)

You can run a DB instance on a virtual private cloud (VPC) using the Amazon Virtual Private Cloud (Amazon VPC) service. When you use a VPC, you have control over your virtual networking environment. You can choose your own IP address range, create subnets, and configure routing and access control lists.

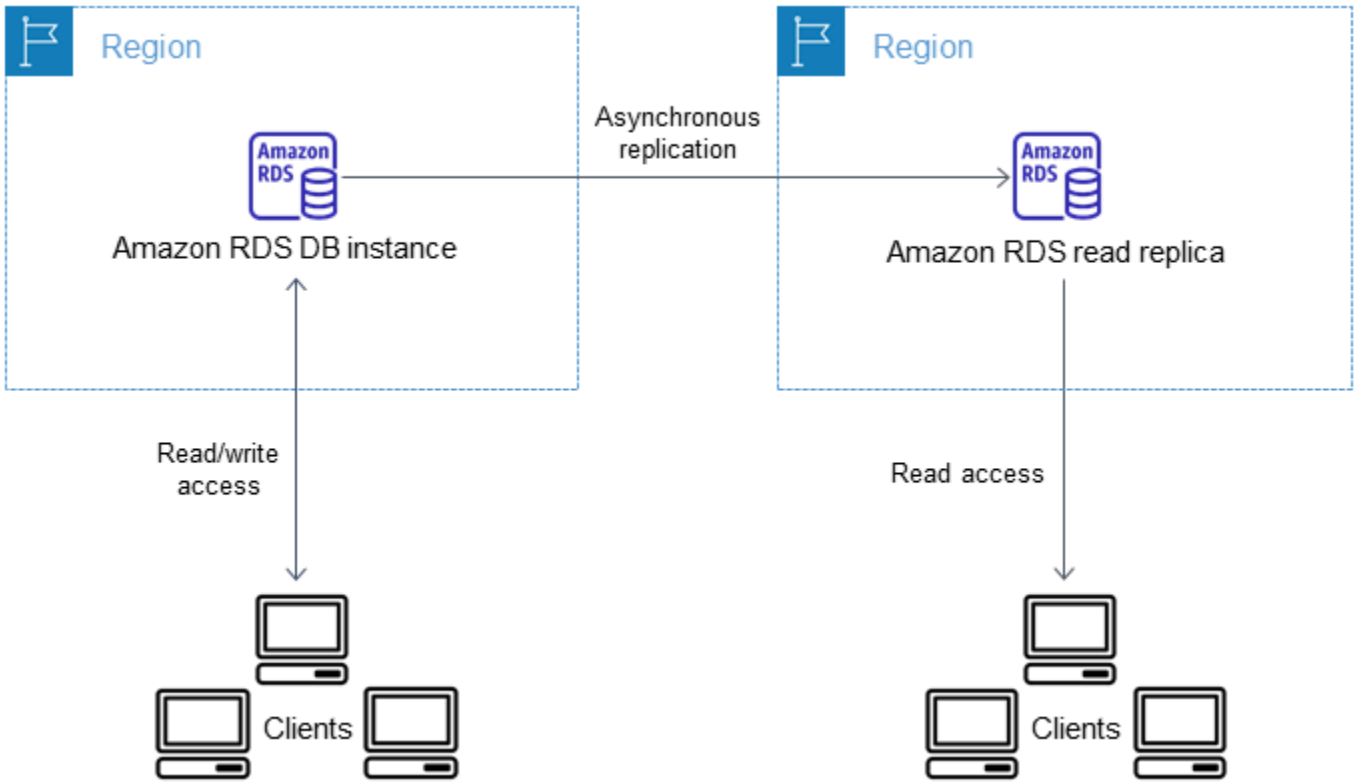
The basic functionality of Amazon RDS is the same whether it's running in a VPC or not. Amazon RDS manages backups, software patching, automatic failure detection, and recovery. There's no additional cost to run your DB instance in a VPC. For more information on using Amazon VPC with RDS, see Amazon VPC and Amazon RDS.

Amazon RDS uses Network Time Protocol (NTP) to synchronize the time on DB instances.

AWS Regions and Availability Zones

Amazon cloud computing resources are housed in highly available data center facilities in different areas of the world (for example, North America, Europe, or Asia). Each data center location is called an AWS Region. With Amazon RDS, you can create your DB instances in multiple Regions.

The following scenario shows an RDS DB instance in one Region that replicates asynchronously to a standby DB instance in a different Region. If one Region becomes unavailable, the instance in the other Region is still available.



Availability Zones

Each AWS Region contains multiple distinct locations called Availability Zones, or AZs. Each Availability Zone is engineered to be isolated from failures in other Availability Zones. Each is engineered to provide inexpensive, low-latency network connectivity to other Availability Zones in the same AWS Region. By launching DB instances in separate Availability Zones, you can protect your applications from the failure of a single location. For more information, see Regions, Availability Zones, and Local Zones.

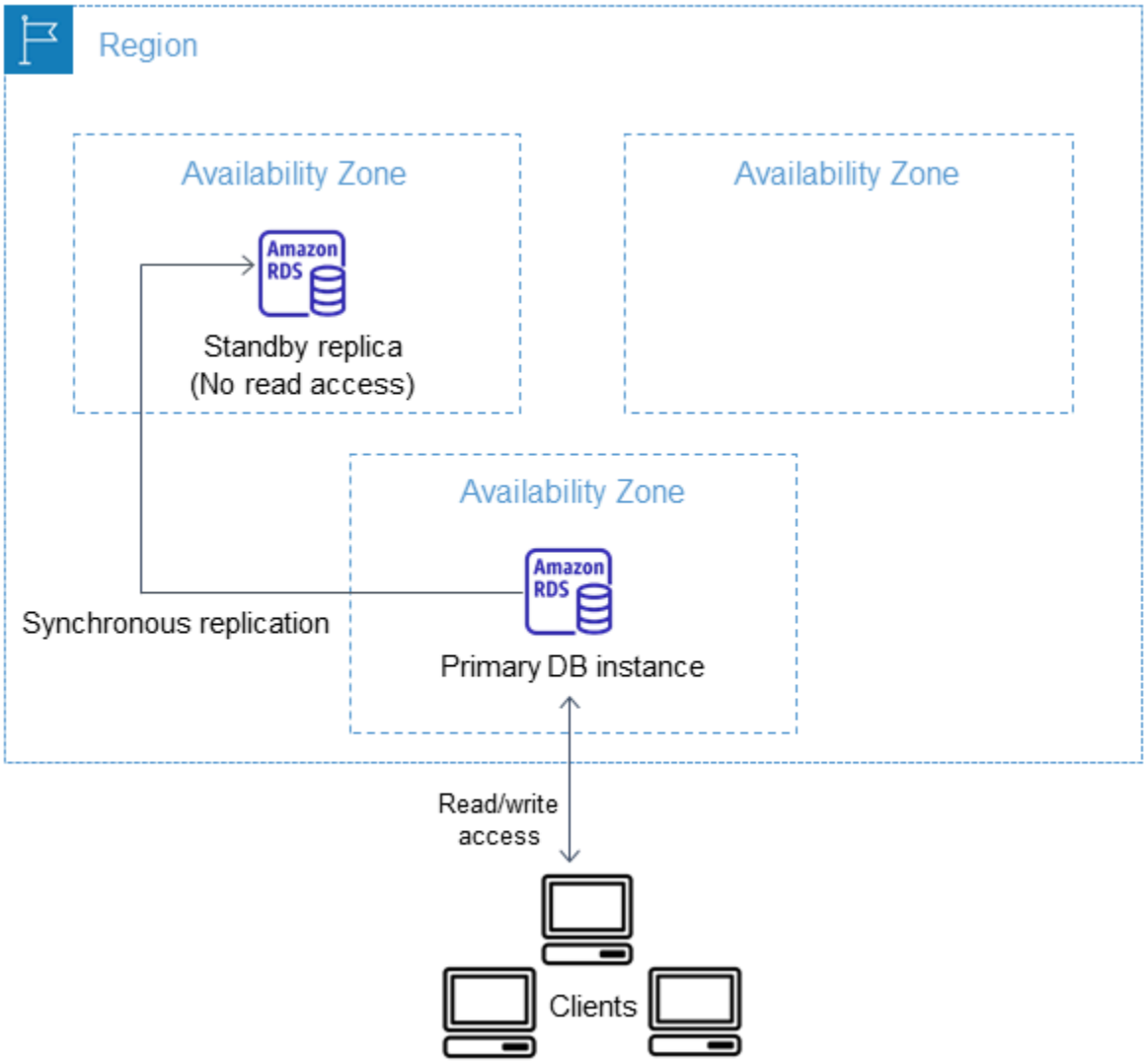
Multi-AZ deployments

You can run your DB instance in several Availability Zones, an option called a Multi-AZ deployment. When you choose this option, Amazon automatically provisions and maintains one or more secondary standby DB instances in a different AZ. Your primary DB instance is replicated across Availability Zones to each secondary DB instance.

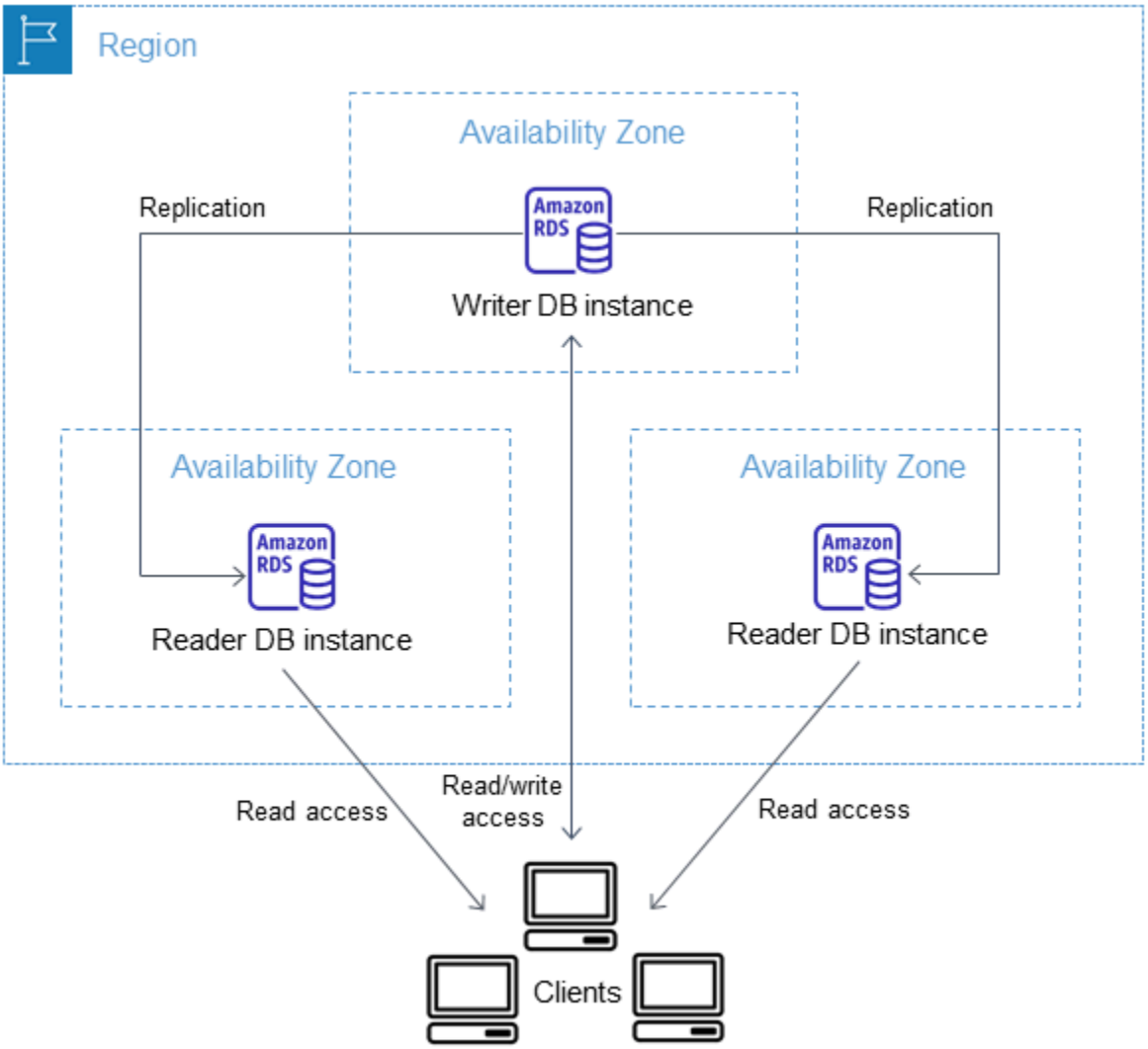
A Multi-AZ deployment provides the following advantages:

- Providing data redundancy and failover support
- Eliminating I/O freezes
- Minimizing latency spikes during system backups
- Serving read traffic on secondary DB instances (Multi-AZ DB clusters deployment only)

The following diagram depicts a Multi-AZ DB instance deployment, where Amazon RDS automatically provisions and maintains a synchronous standby replica in a different Availability Zone. The replica database doesn't serve read traffic.



The following diagram depicts a Multi-AZ DB cluster deployment, which has a writer DB instance and two reader DB instances in three separate Availability Zones in the same AWS Region. All three DB instances can serve read traffic.

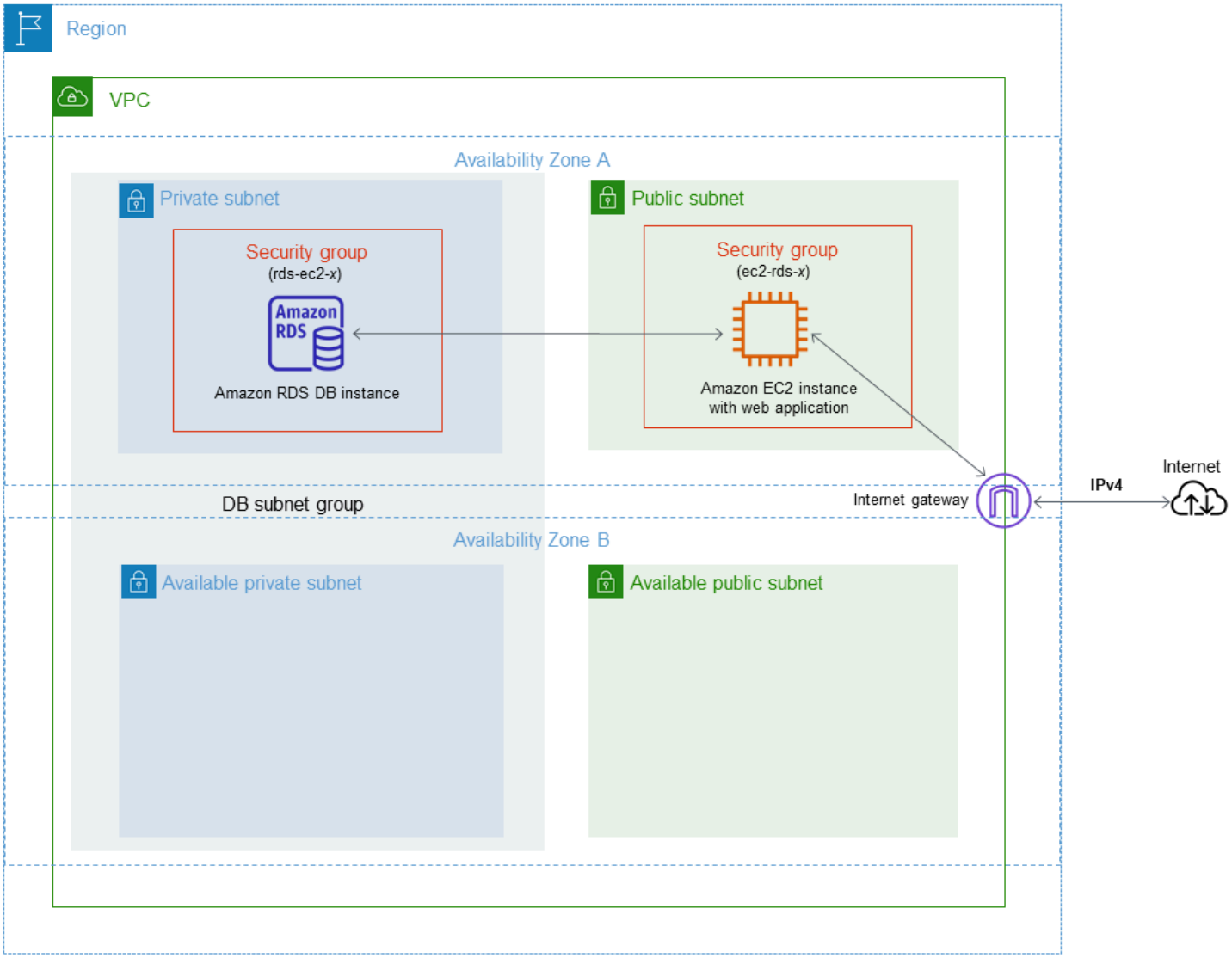


For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

Access control with security groups

A *security group* controls the access to a DB instance by allowing access to IP address ranges or Amazon EC2 instances that you specify. You can apply a security group to one or more DB instances.

A common use of a DB instance in a VPC is to share data with an application server in the same VPC. The following example uses VPC security group `ec2-rds-`*x* to define inbound rules that use the IP addresses of the client application as the source. The application server belongs to this security group. A second security group named `rds-ec2-`*x* specifies `ec2-rds-`*x* as the source and attaches to an RDS DB instance. According to the security group rules, client applications can't directly access the DB instance, but the EC2 instance can access the DB instance.



For more information about security groups, see Security in Amazon RDS .

Amazon RDS monitoring

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon RDS and your other AWS solutions. AWS provides various monitoring tools to watch Amazon RDS, report when something is wrong, and take automatic actions when appropriate.

You can track the performance and health of your DB instances using various automated and manual tools:

Amazon RDS DB instance status and recommendations

View details about the current status of your instance by using the Amazon RDS console, AWS CLI, or RDS API. You can also respond to automated recommendations for database resources, such as DB instances, read replicas, and DB parameter groups. For more information, see Recommendations from Amazon RDS.

Amazon CloudWatch metrics for Amazon RDS

You can use the Amazon CloudWatch service to monitor the performance and health of a DB instance. CloudWatch performance charts are shown in the Amazon RDS console. Amazon RDS automatically sends metrics to CloudWatch every minute for each active database. You don't get additional charges for Amazon RDS metrics in CloudWatch.

Using Amazon CloudWatch alarms, you can watch a single Amazon RDS metric over a specific time period. You can then perform one or more actions based on the value of the metric relative to a threshold that you set. For more information, see Monitoring Amazon RDS metrics with Amazon CloudWatch.

Amazon RDS Performance Insights and operating-system monitoring

Performance Insights assesses the load on your database, and determine when and where to take action. For more information, see Monitoring DB load with Performance Insights on Amazon RDS. Amazon RDS Enhanced Monitoring looks at metrics in real time for the operating system. For more information, see Monitoring OS metrics with Enhanced Monitoring.

Integrated AWS services

Amazon RDS is integrated with Amazon EventBridge, Amazon CloudWatch Logs, and Amazon DevOps Guru. For more information, see Monitoring metrics in an Amazon RDS instance.

User interfaces to Amazon RDS

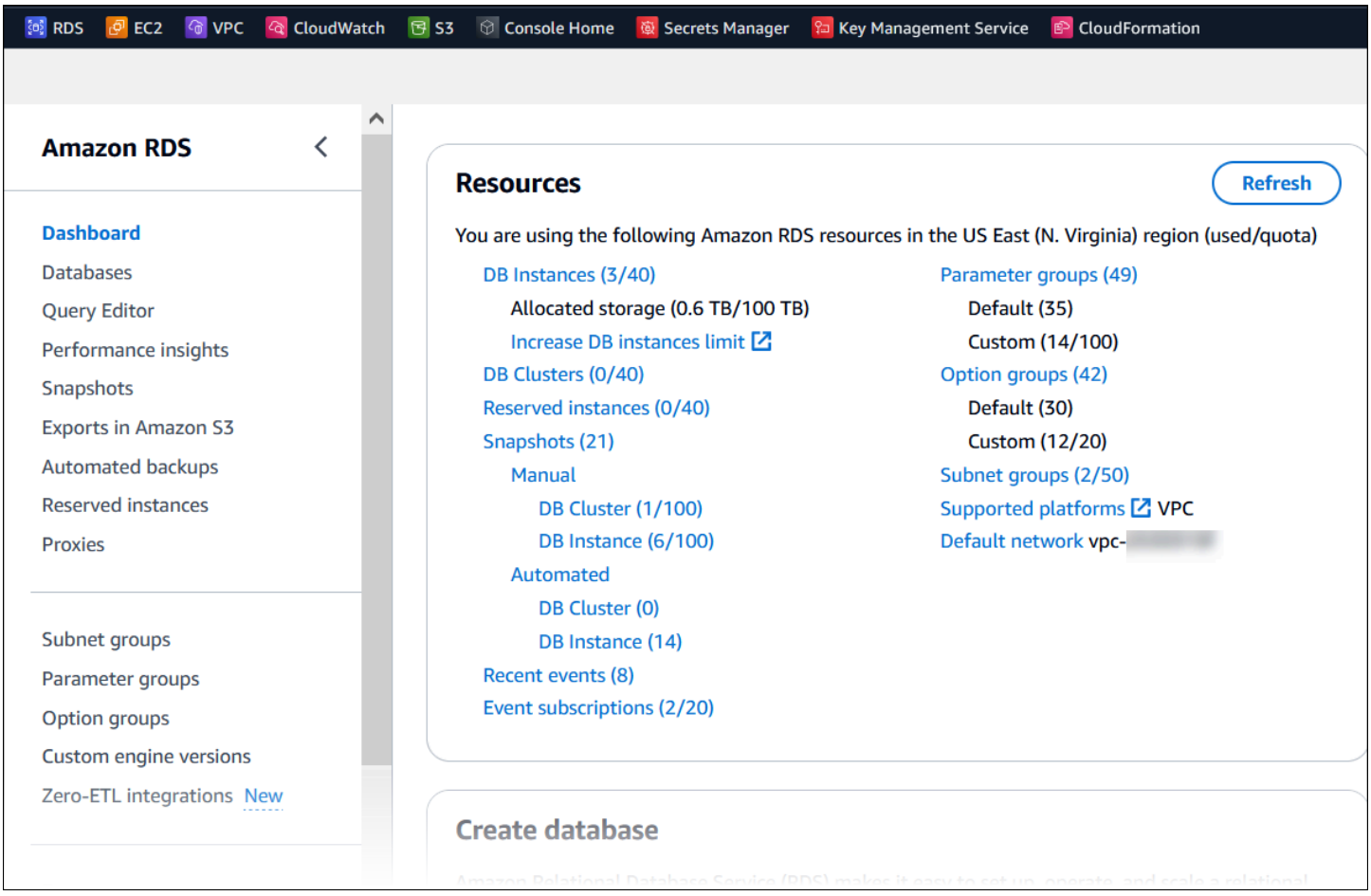
You can interact with Amazon RDS in multiple ways.

Topics

- AWS Management Console
- Command line interface
- Amazon RDS APIs

AWS Management Console

The AWS Management Console is a simple web-based user interface. You can manage your DB instances from the console with no programming required. To access the Amazon RDS console, sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.



Command line interface

You can use the AWS Command Line Interface (AWS CLI) to access the Amazon RDS API interactively. To install the AWS CLI, see Installing the AWS Command Line Interface. To begin using the AWS CLI for RDS, see AWS Command Line Interface reference for Amazon RDS.

Amazon RDS APIs

If you are a developer, you can access the Amazon RDS programmatically using APIs. For more information, see Amazon RDS API reference.

For application development, we recommend that you use one of the AWS Software Development Kits (SDKs). The AWS SDKs handle low-level details such as authentication, retry logic, and error handling, so that you can focus on your application logic. AWS SDKs are available for a wide variety of languages. For more information, see Tools for Amazon web services .

AWS also provides libraries, sample code, tutorials, and other resources to help you get started more easily. For more information, see Sample code & libraries.

How you are charged for Amazon RDS

When you use Amazon RDS, you can choose to use on-demand DB instances or reserved DB instances. For more information, see DB instance billing for Amazon RDS .

For Amazon RDS pricing information, see the Amazon RDS product page.

AWS Free Tier on Amazon RDS

You can use AWS Free Tier on Amazon RDS with the following engines and DB instance classes:

- **Engine types** – MariaDB, MySQL, PostgreSQL, or SQL Server Express Edition
- **DB instance classes**
  - **t3.micro** – For all engine types
  - **t4g.micro** – For all engine types except SQL Server Express Edition

You can't use the following features, resources, and actions with AWS Free Tier on Amazon RDS:

- Amazon RDS custom engine options
- Reserved DB instances
- Deployment options other than Single-AZ
- Migrating snapshots
- Query editor
- Creating Aurora read replicas

These limitations are subject to change as the AWS Free Tier program evolves. For more information about AWS Free Tier, see Explore AWS services with AWS Free Tier. For more information about Amazon RDS Free Tier, see Amazon RDS Free Tier.

If you were an AWS Free Tier customer before July 17, 2025, you can continue to use Amazon RDS Free Tier until the 12 months of your free usage expires. During this time period, you remain eligible for the following usage:

- 750 hours each month of Single-AZ db.t3.micro and db.t4g.micro instance classes running MySQL, MariaDB, or PostgreSQL on Amazon RDS
- 750 hours each month of a db.t3.micro instance class running SQL Server Express Edition on Amazon RDS

After July 17, 2025, any new AWS Free Tier resources that you create on AWS will use the new AWS Free Tier offering.

What's next?

The preceding section introduced you to the basic infrastructure components that RDS offers. What should you do next?

Getting started

Create a DB instance using instructions in Getting started with Amazon RDS.

Topics specific to database engines

You can review information specific to a particular DB engine in the following sections:

- Amazon RDS for Db2
- Amazon RDS for MariaDB
- Amazon RDS for Microsoft SQL Server
- Amazon RDS for MySQL
- Amazon RDS for Oracle
- Amazon RDS for PostgreSQL

Amazon RDS DB instances

A *DB instance* is an isolated database environment running in the cloud. It is the basic building block of Amazon RDS. A DB instance can contain multiple user-created databases, and can be accessed using the same client tools and applications you might use to access a standalone database instance. DB instances are simple to create and modify with the AWS command line tools, Amazon RDS API operations, or the AWS Management Console.

> **Note**
>
> Amazon RDS supports access to databases using any standard SQL client application. Amazon RDS does not allow direct host access except with RDS Custom.

You can have up to 40 Amazon RDS DB instances, with the following limitations:

- 10 for each SQL Server edition (Enterprise, Standard, Web, and Express) under the "license-included" model
- 10 for Oracle under the "license-included" model
- 40 for Db2 under the "bring-your-own-license" (BYOL) licensing model
- 40 for MySQL, MariaDB, or PostgreSQL
- 40 for Oracle under the "bring-your-own-license" (BYOL) licensing model

> **Note**
>
> If your application requires more DB instances, you can request additional DB instances by using this form.

Each DB instance has a DB instance identifier. This customer-supplied name uniquely identifies the DB instance when interacting with the Amazon RDS API and AWS CLI commands. The DB instance identifier must be unique for that customer in an AWS Region.

The DB instance identifier forms part of the DNS hostname allocated to your instance by RDS. For example, if you specify db1 as the DB instance identifier, then RDS will automatically allocate a DNS endpoint for your instance. An example endpoint is *db1*`.`*abcdefghijkl*`.`*us-east-1*`.rds.amazonaws.com`, where *db1* is your instance ID.

In the example endpoint *db1*.*abcdefghijkl*.*us-east-1*.rds.amazonaws.com, the string *abcdefghijkl* is a unique identifier for a specific combination of AWS Region and AWS account. The identifier *abcdefghijkl* in the example is internally generated by RDS and doesn't change for the specified combination of Region and account. Thus, all your DB instances in this Region share the same fixed identifier. Consider the following features of the fixed identifier:

- If you rename your DB instance, the endpoint is different but the fixed identifier is the same. For example, if you rename *db1* to *renamed-db1*, the new instance endpoint is *renamed-db1*.*abcdefghijkl*.*us-east-1*.rds.amazonaws.com.
- If you delete and re-create a DB instance with the same DB instance identifier, the endpoint is the same.
- If you use the same account to create a DB instance in a different Region, the internally generated identifier is different because the Region is different, as in *db2*.*mnopqrstuvwx*.*us-west-1*.rds.amazonaws.com.

Each DB instance supports a database engine. Amazon RDS currently supports Db2, MySQL, MariaDB, PostgreSQL, Oracle, Microsoft SQL Server, and Amazon Aurora database engines.

When creating a DB instance, some database engines require that a database name be specified. A DB instance can host multiple databases or a single Oracle database with multiple schemas. The database name value depends on the database engine:

- For the Db2 database engine, the database name is the name of the database hosted in your DB instance. This field is optional. You can create a database later by calling the `rdsadmin.create_database` stored procedure. For more information, see the section called "Creating a database".
- For the MySQL and MariaDB database engines, the database name is the name of a database hosted in your DB instance. Databases hosted by the same DB instance must have a unique name within that instance.
- For the Oracle database engine, database name is used to set the value of ORACLE_SID, which must be supplied when connecting to the Oracle RDS instance.
- For the Microsoft SQL Server database engine, database name is not a supported parameter.
- For the PostgreSQL database engine, the database name is the name of a database hosted in your DB instance. A database name is not required when creating a DB instance. Databases hosted by the same DB instance must have a unique name within that instance.

Amazon RDS creates a master user account for your DB instance as part of the creation process. This master user has permissions to create databases and to perform create, delete, select, update, and insert operations on tables the master user creates. You must set the master user password when you create a DB instance, but you can change it at any time using the AWS CLI, Amazon RDS API operations, or the AWS Management Console. You can also change the master user password and manage users using standard SQL commands.

Note

This guide covers non-Aurora Amazon RDS database engines. For information about using Amazon Aurora, see the *Amazon Aurora User Guide*.

DB instance classes

The DB instance class determines the computation and memory capacity of an Amazon RDS DB instance. The DB instance class that you need depends on your processing power and memory requirements.

A DB instance class consists of both the DB instance class type and the size. For example, db.r6g is a memory-optimized DB instance class type powered by AWS Graviton2 processors. Within the db.r6g instance class type, db.r6g.2xlarge is a DB instance class. The size of this class is 2xlarge.

For more information about instance class pricing, see Amazon RDS pricing.

For more information about DB instance class types, supported DB engines, supported AWS Regions, changing your DB instance class, configuring the processor for RDS for Oracle, or hardware specifications for DB instance classes, see the following sections.

Topics

- DB instance class types
- Supported DB engines for DB instance classes
- Determining DB instance class support in AWS Regions
- Changing your DB instance class
- Configuring the processor for a DB instance class in RDS for Oracle
- Hardware specifications for DB instance classes

DB instance class types

Amazon RDS supports DB instance classes for the following use cases:

- General-purpose
- Memory-optimized
- Compute-optimized
- Burstable-performance
- Optimized Reads

For more information about Amazon EC2 instance types, see Instance types in the Amazon EC2 documentation.

General-purpose instance class types

The following general-purpose DB instance classes are available:

- **db.m8g** – General-purpose DB instance classes powered by AWS Graviton4 processors. These instance classes deliver balanced compute, memory, and networking for a broad range of general-purpose workloads. Compared to seventh-generation AWS Graviton3-based M7g instances, these new classes offer larger instance sizes with up to 3x more vCPUs and memory. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.

  You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton4 processors. To do so, complete the same steps as with any other DB instance modification.
- **db.m7i** – General-purpose DB instance classes powered by 4th Generation Intel Xeon Scalable processors. The db.m7i instances are SAP certified and ideal for supporting enterprise applications that need larger instance sizes or high continuous CPU usage. These instance classes deliver balanced compute, memory, and networking for a broad range of general-purpose workloads. This instance class type delivers up to 40,000 Mbps EBS bandwidth and up to 50 Gbps network bandwidth. The db.m7i instances deliver up to 15% better price performance compared to db.m6i instances. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.m7g** – General-purpose DB instance classes powered by AWS Graviton3 processors. These instance classes deliver balanced compute, memory, and networking for a broad range of general-purpose workloads. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.

  You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton3 processors. To do so, complete the same steps as with any other DB instance modification.
- **db.m6g** – General-purpose DB instance classes powered by AWS Graviton2 processors. These instances deliver balanced compute, memory, and networking for a broad range of general-purpose workloads. The db.m6gd instance classes have local NVMe-based SSD block-level storage for applications that need high-speed, low latency local storage.

  You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton2 processors. To do so, complete the same steps as with any other DB instance modification.
- **db.m6i** – General-purpose DB instance classes powered by 3rd Generation Intel Xeon Scalable processors. These instances are SAP Certified and ideal for workloads such as backend servers

supporting enterprise applications, gaming servers, caching fleets, and application development environments. The db.m6id and db.m6idn instance classes offer up to 7.6 TB of local NVMe-based SSD storage, whereas db.m6in offers EBS-only storage. The db.m6in and db.m6idn classes offer up to 200 Gbps of network bandwidth. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.

- **db.m5** –General-purpose DB instance classes that provide a balance of compute, memory, and network resources, and are a good choice for many applications. The db.m5d instance class offers NVMe-based SSD storage that is physically connected to the host server. The db.m5 instance classes provide more computing capacity than the previous db.m4 instance classes. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.m4** – General-purpose DB instance classes that provide more computing capacity than the previous db.m3 instance classes.

  For the RDS for Oracle DB engines, Amazon RDS no longer supports db.m4 DB instance classes. If you had previously created RDS for Oracle db.m4 DB instances, Amazon RDS automatically upgrades those DB instances to equivalent db.m5 DB instance classes.

  For the RDS for MariaDB, RDS for MySQL, RDS for SQL Server, and RDS for PostgreSQL DB engines, Amazon RDS has started the end-of-support process for this DB instance class using the following schedule. For all RDS DB instances that use this instance class, we recommend that you upgrade to a newer generation DB instance class as soon as possible.

Action or recommendation	Date
Starting on this date, Amazon RDS began automatically upgrading instances using db.m4 to the newer generation db.m5 instance class. Creating DB instances using the db.m4 instance class is no longer supported.	June 1, 2024
Amazon RDS ends support for db.m4.	December 31, 2024

- **db.m3** – General-purpose DB instance classes that provide more computing capacity than the previous db.m1 instance classes.

  For the RDS for MariaDB, RDS for MySQL, and RDS for PostgreSQL DB engines, Amazon RDS has started the end-of-life process for db.m3 DB instance classes using the following schedule, which

includes upgrade recommendations. For all RDS DB instances that use db.m3 DB instance classes, we recommend that you upgrade to a higher generation DB instance class as soon as possible.

Action or recommendation	Dates
You can no longer create RDS DB instances that use db.m3 DB instance classes.	Now
Amazon RDS started automatic upgrades of RDS DB instances that use db.m3 DB instance classes to equivalent db.m5 DB instance classes.	February 1, 2023

Memory-optimized instance class types

The memory-optimized Z family supports the following instance classes:

- **db.z1d** – Instance classes optimized for memory-intensive applications. These instance classes offer both high compute capacity and a high memory footprint. High frequency z1d instances deliver a sustained all-core frequency of up to 4.0 GHz.

The memory-optimized X family supports the following instance classes:

- **db.x2g** – Instance classes optimized for memory-intensive applications and powered by AWS Graviton2 processors. These instance classes offer low cost per GiB of memory.

  You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton2 processors. To do so, complete the same steps as with any other DB instance modification.
- **db.x2i** – Instance classes optimized for memory-intensive applications. The **db.x2iedn** and **db.x2idn** instance class types are powered by third-generation Intel Xeon Scalable processors (Ice Lake). They include up to 3.8 TB of local NVMe SSD storage, up to 100 Gbps of networking bandwidth, and up to 4 TiB (db.x2iden) or 2 TiB (db.x2idn) of memory. The **db.x2iezn** type is powered by second-generation Intel Xeon Scalable processors (Cascade Lake) with an all-core turbo frequency of up to 4.5 GHz and up to 1.5 TiB of memory and by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.x1** – Instance classes optimized for memory-intensive applications. These instance classes offer one of the lowest price per GiB of RAM among the DB instance classes and up to 1,952

GiB of DRAM-based instance memory. The **db.x1e** instance class type offers up to 3,904 GiB of DRAM-based instance memory.

The memory-optimized R family supports the following instance class types:

- **db.r8g** – Instance classes powered by AWS Graviton4 processors. These instance classes are ideal for running memory-intensive workloads in open-source databases such as MySQL and PostgreSQL. These instances offer larger instance sizes with up to 3x more vCPUs and memory than the seventh-generation AWS Graviton3-based db.r7g instances. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton4 processors. To do so, complete the same steps as with any other DB instance modification.
- **db.r7g** – Instance classes powered by AWS Graviton3 processors. These instance classes are ideal for running memory-intensive workloads in open-source databases such as MySQL and PostgreSQL.

  You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton3 processors. To do so, complete the same steps as with any other DB instance modification. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.r7i** – Instance classes powered by 4th Generation Intel Xeon Scalable processors. These instance classes are SAP-Certified and are ideal for running memory-intensive workloads in open-source databases such as MySQL and PostgreSQL. You can modify a DB instance to use one of the DB instance classes powered by 4th Generation Intel Xeon Scalable processors. To do so, complete the same steps as with any other DB instance modification. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.r6g** – Instance classes powered by AWS Graviton2 processors. These instance classes are ideal for running memory-intensive workloads in open-source databases such as MySQL and PostgreSQL. The **db.r6gd** type offers local NVMe-based SSD block-level storage for applications that need high-speed, low latency local storage. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton2 processors. To do so, complete the same steps as with any other DB instance modification.
- **db.r6i** – Instance classes powered by 3rd Generation Intel Xeon Scalable processors. These instance classes are SAP-Certified and are an ideal fit for memory-intensive workloads in open-source databases such as MySQL and PostgreSQL. The **db.r6id**, **db.r6in**, and **db.r6idn** instance

classes have a memory-to-vCPU ratio of 8:1 and a maximum memory of 1 TiB. The db.r6id and db.r6idn classes offer up to 7.6 TB of direct-attached NVMe-based SSD storage, whereas db.r6in offers EBS-only storage. The db.r6idn and db.r6in classes offer up to 200 Gbps of network bandwidth. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.

- **db.r5b** – Instance classes that are memory-optimized for throughput-intensive applications. Powered by the AWS Nitro System, db.r5b instances deliver up to 60 Gbps bandwidth and 260,000 IOPS of EBS performance. This is the fastest block storage performance on EC2.
- **db.r5d** – Instance classes that are optimized for low latency, very high random I/O performance, and high sequential read throughput.
- **db.r5** – Instance classes optimized for memory-intensive applications. These instance classes offer improved networking performance. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.r4** – Instance classes that provide improved networking over previous db.r3 instance classes.

  For the RDS for Oracle DB engines, Amazon RDS has started the end-of-life process for db.r4 DB instance classes using the following schedule, which includes upgrade recommendations. For RDS for Oracle DB instances that use db.r4 instance classes, we recommend that you upgrade to a higher generation instance class as soon as possible.

Action or recommendation	Dates
You can no longer create RDS for Oracle DB instances that use db.r4 DB instance classes.	Now
Amazon RDS started automatic upgrades of RDS for Oracle DB instances that use db.r4 DB instance classes to equivalent db.r5 DB instance classes.	April 17, 2023

  For the RDS for MariaDB, RDS for MySQL, RDS for SQL Server, and RDS for PostgreSQL DB engines, Amazon RDS has started the end-of-support process for this DB instance class using the following schedule. For all RDS DB instances that use this instance class, we recommend that you upgrade to a newer generation DB instance class as soon as possible.

Action or recommendation	Dates
Starting on this date, Amazon RDS began automatically upgrading instances using db.r4 to the newer generation db.r5 instance class. Creating DB instances using the db.m4 instance class is no longer supported.	June 1, 2024
Amazon RDS ends support for db.r4.	December 31, 2024

- **db.r3** – Instance classes that provide memory optimization.

  For the RDS for MariaDB, RDS for MySQL, and RDS for PostgreSQL DB engines, Amazon RDS has started the end-of-life process for db.r3 DB instance classes using the following schedule, which includes upgrade recommendations. For all RDS DB instances that use db.r3 DB instance classes, we recommend that you upgrade to a higher generation DB instance class as soon as possible.

Action or recommendation	Dates
You can no longer create RDS DB instances that use db.r3 DB instance classes.	Now
Amazon RDS started automatic upgrades of RDS DB instances that use db.r3 DB instance classes to equivalent db.r5 DB instance classes.	February 1, 2023

Compute-optimized instance class type

The following compute-optimized instance class types are available:

- **db.c6gd** – Instance classes that are ideal for running advanced compute-intensive workloads. Powered by AWS Graviton2 processors, these instance classes offer local NVMe-based SSD block-level storage for applications that need high-speed, low latency local storage.

> **Note**
>
> The c6gd instance classes are supported only for Multi-AZ DB cluster deployments. They're the only instance class supported for Multi-AZ DB clusters that offer the `medium` instance size. For more information, see the section called "Multi-AZ DB cluster deployments".

Burstable-performance instance class types

The following burstable-performance DB instance class types are available:

- **db.t4g** – General-purpose instance classes powered by Arm-based AWS Graviton2 processors. These instance classes deliver better price performance than previous burstable-performance DB instance classes for a broad set of burstable general-purpose workloads. Amazon RDS db.t4g instances are configured for Unlimited mode. This means that they can burst beyond the baseline over a 24-hour window for an additional charge.

  You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton2 processors. To do so, complete the same steps as with any other DB instance modification.
- **db.t3** – Instance classes that provide a baseline performance level, with the ability to burst to full CPU usage. The db.t3 instances are configured for Unlimited mode. These instance classes provide more computing capacity than the previous db.t2 instance classes. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.t2** – Instance classes that provide a baseline performance level, with the ability to burst to full CPU usage. The db.t2 instances are configured for Unlimited mode. We recommend using these instance classes only for development and test servers, or other non-production servers.

  For the RDS for MariaDB, RDS for MySQL, RDS for SQL Server, and RDS for PostgreSQL DB engines, Amazon RDS has started the end-of-support process for this DB instance class using the following schedule. For all RDS DB instances that use this instance class, we recommend that you upgrade to a newer generation DB instance class as soon as possible.

Action or recommendation	Dates
Starting on this date, Amazon RDS began automatically upgrading instances using db.t2 to	June 1, 2024

Action or recommendation	Dates
the newer generation db.t3 instance class. Creating DB instances using the db.t2 instance class is no longer supported.	
Amazon RDS ends support for db.t2.	December 31, 2024

> **Note**
>
> The DB instance classes that use the AWS Nitro System (db.m5, db.r5, db.t3) are throttled on combined read plus write workload.

For DB instance class hardware specifications, see Hardware specifications for DB instance classes.

Optimized Reads instance class types

The following Optimized Reads instance class types are available:

- **db.m8gd** – Instance classes powered by AWS Graviton4 processor. These instance classes are ideal for general purpose workloads that need high-speed, low latency storage. They offer a maximum memory of 786 GiB and up to 11.4 TB of direct-attached NVMe-based SSD storage.
- **db.r8gd** – Instance classes powered by Graviton4 processors. These instance classes are ideal for running memory-intensive workloads and offer local NVMe-based SSD block-level storage for applications that need high-speed, low latency local storage. They offer a maximum memory of 1.5 TiB and up to 11.4 TB of direct-attached NVMe-based SSD storage.
- **db.r6gd** – Instance classes powered by AWS Graviton2 processors. These instance classes are ideal for running memory-intensive workloads and offer local NVMe-based SSD block-level storage for applications that need high-speed, low latency local storage.
- **db.r6id** – Instance classes powered by 3rd Generation Intel Xeon Scalable processors. These instance classes are SAP-Certified and are an ideal fit for memory-intensive workloads. They offer a maximum memory of 1 TiB and up to 7.6 TB of direct-attached NVMe-based SSD storage.

Supported DB engines for DB instance classes

The following are DB engine–specific considerations for DB instance classes:

Db2

DB instance class support varies according to the version and edition of Db2. For instance class support by version and edition, see Amazon RDS for Db2 instance classes.

Microsoft SQL Server

DB instance class support varies according to the version and edition of SQL Server. For instance class support by version and edition, see DB instance class support for Microsoft SQL Server.

Oracle

DB instance class support varies according to the Oracle Database version and edition. RDS for Oracle supports additional memory-optimized instance classes. These classes have names of the form db.r5.*instance_size*.tpc*threads_per_core*.mem*ratio*. For the vCPU count and memory allocation for each optimized class, see Supported RDS for Oracle DB instance classes.

RDS Custom

For information about the DB instance classes supported in RDS Custom, see DB instance class support for RDS Custom for Oracle and DB instance class support for RDS Custom for SQL Server.

In the following table, you can find details about supported Amazon RDS DB instance classes for each Amazon RDS DB engine. The cell for each engine contains one of the following values:

Yes

The instance class is supported for all versions of the DB engine.

No

The instance class isn't supported for the DB engine.

specific-versions

The instance class is supported only for the specified database versions of the DB engine.

Amazon RDS periodically deprecates major and minor DB engine versions. Not all AWS Regions might have support for earlier engine versions. For information about current supported versions,

see topics for the individual DB engines: Db2 versions, MariaDB versions, Microsoft SQL Server versions, MySQL versions, Oracle versions, and PostgreSQL versions.

Topics

- Supported DB engines for general-purpose instance classes
- Supported DB engines for memory-optimized instance classes
- Supported DB engines for compute-optimized instance classes
- Supported DB engines for burstable-performance instance classes
- Supported DB engines for Optimized Reads instance classes

Supported DB engines for general-purpose instance classes

The following tables show the supported databases and database versions for the general-purpose instance classes.

db.m8g – general-purpose instance classes powered by AWS Graviton4 processors

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.m8g.4 xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m8g.2 xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.m8g.1 xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m8g.1 xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m8g.8 large	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m8g.4 large	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.m8g.2 large	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m8g.xl arge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m8g.la rge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher

db.m7i – general-purpose instance classes powered by 4th generation Intel Xeon Scalable processors

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.m7i.48 xlarge	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, Enterpr e Edition only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m7i.24 xlarge	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, Enterpr e Edition only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m7i.16 xlarge	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, Enterpr e Edition only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m7i.12 xlarge	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, Enterpr e Edition only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.m7i.8x large	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, Enterp e Edition only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m7i.4x large	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, all edition	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m7i.2x large	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, all edition	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m7i.xl arge	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, all edition	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.m7i.la rge	Db2 11.5	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, all edition	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.m7i.m tal-48xl	No	No	No	No	BYOL only, Enterp e Edition only	No
db.m7i.m tal-24xl	No	No	No	No	BYOL only, Enterp e Edition only	No

db.m7g – general-purpose instance classes powered by AWS Graviton3 processors

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.m7g.1 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
						and higher 13 versions
db.m7g.1 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions
db.m7g.8 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions
db.m7g.4 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions
db.m7g.2 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.m7g.xl arge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions
db.m7g.la rge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions

db.m6g – general-purpose instance classes powered by AWS Graviton2 processors

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.m6g.1 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.m6g.1 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
						12.7 and higher 12 versions
db.m6g.8 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.m6g.4 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.m6g.2 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.m6g.x arge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.m6g.l rge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions

db.m6gd – general-purpose instance classes powered by AWS Graviton2 processors and SSD storage

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.m6gd.16xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, and 14 versions; 13.7 and higher 13 versions; and 13.4
db.m6gd.12xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, and 14 versions; 13.7 and higher 13 versions; and 13.4
db.m6gd.8xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, and 14 versions; 13.7 and higher 13 versions; and 13.4
db.m6gd.4xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, and 14 versions; 13.7 and higher 13 versions; and 13.4
db.m6gd.2xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, and 14 versions; 13.7 and higher 13 versions; and 13.4

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.m6gd.xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, and 14 versions; 13.7 and higher 13 versions; and 13.4
db.m6gd.large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, and 14 versions; 13.7 and higher 13 versions; and 13.4

db.m6id – general-purpose instance classes powered by 3rd generation Intel Xeon Scalable processors and SSD storage

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.m6id.32xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6id.24xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.m6id.16xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6id.12xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6id.8xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6id.4xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.m6id.2xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6id.xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6id.large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6id.metal	No	No	No	No	BYOL only, Enterprise Edition only	No

db.m6idn – general-purpose instance classes with 3rd Generation Intel Xeon Scalable processors, SSD storage, and network optimization

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.m6idn.32xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6idn.24xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6idn.16xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6idn.12xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6idn.8xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6idn.4xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions,

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
						and 13.7 and higher 13 versions
db.m6idn.2xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6idn.xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.m6idn.large	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions

db.m6in – general-purpose instance classes powered by 3rd generation Intel Xeon Scalable processors and network optimization

Instance class	Db	MariaDB	Micros SQL Serve	MySQL	Ora	PostgreSQL
db.m6in.32xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12

Instance class	Db	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
						versions, and 11.16 and higher 11 versions
db.m6in.24xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.m6in.16xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.m6in.12xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.m6in.8xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions

Instance class	Db	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
db.m6in.4xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.m6in.2xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.m6in.xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.m6in.large	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions

Instance class	Db	MariaDB	Micros SQL Serve	MySQL	Ora	PostgreSQL
db.m6in.m etal	No	No	No	No	BYO only Ente e Edit only	No

db.m6i – general-purpose instance classes powered by 3rd generation Intel Xeon Scalable processors

Instance class	Db	MariaDB	Micros SQL Serve	MySQL	Oracle	PostgreSQL
db.m6i.3 xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions
db.m6i.2 xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions
db.m6i.1 xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions
db.m6i.1 xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions
db.m6i.8 large	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.m6i.4x large	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions
db.m6i.2x large	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions
db.m6i.xl arge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions
db.m6i.la rge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Oracle Databa 19c	All available versions
db.m6i.m tal	No	No	No	No	BYOL only, Enterp e Edition only	No

db.m5d – general-purpose instance classes powered by Intel Xeon Platinum processors and SSD storage

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.m5d.2x xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
						and higher 13 versions, and 13.4
db.m5d.1(xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.m5d.1(xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.m5d.8(large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.m5d.4(large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.m5d.2(large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.m5d.xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.m5d.large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

db.m5 – general-purpose instance classes 2.5 GHz Intel Xeon Platinum processors

Instance class	Db	Maria	Microsoft SQL Server	MyS	Orac	PostgreSQL
db.m5.24xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.m5.16xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.m5.12xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.m5.8xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions

Instance class	Db2	Maria	Microsoft SQL Server	MyS	Orac	PostgreSQL
db.m5.4xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.m5.2xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.m5.xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.m5.large	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions

db.m4 – general-purpose instance classes with Intel Xeon processors

Instance class	Db	MariaDB	Microsoft SQL Server	MySQL	Oracl	PostgreSQL
db.m4.16 large	Nc	Deprecated	Deprecated	Deprecated	Depre d	Deprecated
db.m4.10 large	Nc	Deprecated	Deprecated	Deprecated	Depre d	Deprecated
db.m4.4x arge	Nc	Deprecated	Deprecated	Deprecated	Depre d	Deprecated
db.m4.2x arge	Nc	Deprecated	Deprecated	Deprecated	Depre d	Deprecated

Instance class	Db	MariaDB	Microsoft SQL Server	MySQL	Oracl	PostgreSQL
db.m4.xlarge	Nc	Deprecated	Deprecated	Deprecad	Depred	Deprecated
db.m4.large	Nc	Deprecated	Deprecated	Deprecad	Depred	Deprecated

db.m3 – general-purpose instance classes

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.m3.2xlarge	No	No	Deprecated	Yes	Deprecated	Deprecated
db.m3.xlarge	No	No	Deprecated	Yes	Deprecated	Deprecated
db.m3.large	No	No	Deprecated	Yes	Deprecated	Deprecated
db.m3.medium	No	No	Deprecated	Yes	Deprecated	Deprecated

Supported DB engines for memory-optimized instance classes

The following tables show the supported databases and database versions for the memory-optimized instance classes.

db.z1d – memory-optimized instance classes

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.z1d.1 xlarge	No	No	Yes	No	Yes	No
db.z1d.6 large	No	No	Yes	No	Yes	No
db.z1d.3 large	No	No	Yes	No	Yes	No
db.z1d.2 large	No	No	Yes	No	Yes	No
db.z1d.xl arge	No	No	Yes	No	Yes	No
db.z1d.la rge	No	No	Yes	No	Yes	No

db.x2g – memory-optimized instance classes powered by AWS Graviton2 processors

Instance class	Db	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
db.x2g.1 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.x2g.1 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions

Instance class	Db	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
db.x2g.8 large	Nc	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.x2g.4 large	Nc	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.x2g.2 large	Nc	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.x2g.x arge	Nc	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.x2g.l rge	Nc	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions

db.x2idn – memory-optimized instance classes powered by 3rd generation Intel Xeon Scalable processors

Instance class	DI	MariaDB	Micro SQL Serve	MySQL	Oracle	PostgreSQL
db.x2idn. 32xlarge	Nc	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	Enterp e Editio only	PostgreSQL 15 versions, 14.6, and 13.9

Instance class	DI	MariaDB	Micros SQL Serve	MySQL	Oracl	PostgreSQL
db.x2idn. 24xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	Enterp e Editio only	PostgreSQL 15 versions, 14.6, and 13.9
db.x2idn. 16xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	Enterp e Editio only	PostgreSQL 15 versions, 14.6, and 13.9
db.x2idn. metal	No	No	No	No	BYOL only, Enterp e Editio only	No

db.x2iedn – memory-optimized instance classes with local NVMe-based SSDs, powered by 3rd generation Intel Xeon Scalable processors

Instance class	DI	MariaDB	Microsoft SQL Server	MySQL	Oracl	PostgreSQL
db.x2iedn .32xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Enterpris e and Standard Editions only, SQL Server 2014 12.00 and higher	MySQL 8.4 and 8.0	Enterp e Editio only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

Instance class	DI	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.x2iedn .24xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Enterprise and Standard Editions only, SQL Server 2014 12.00 and higher	MySQL 8.4 and 8.0	Enterp e Editio only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.x2iedn .16xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Enterprise and Standard Editions only, SQL Server 2014 12.00 and higher	MySQL 8.4 and 8.0	Enterp e Editio only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.x2iedn .8xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Enterprise and Standard Editions only, SQL Server 2014 12.00 and higher	MySQL 8.4 and 8.0	Enterp e Editio only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

Instance class	DI	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.x2iedn .4xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Enterprise and Standard Editions only, SQL Server 2014 12.00 and higher	MySQL 8.4 and 8.0	Enterp e Editio and Stand Editio 2 (SE2)	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.x2iedn .2xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Enterprise and Standard Editions only, SQL Server 2014 12.00 and higher	MySQL 8.4 and 8.0	Enterp e Editio and Stand Editio 2 (SE2)	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.x2iedn .xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Enterprise and Standard Editions only, SQL Server 2014 12.00 and higher	MySQL 8.4 and 8.0	Enterp e Editio and Stand Editio 2 (SE2)	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.x2iedn .metal	No	No	No	No	BYOL only, Enterp e Editio only	No

db.x2iezn – memory-optimized instance classes powered by 2nd generation Intel Xeon Scalable processors

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.x2iezn.metal	No	No	No	No	BYOL only, Enterprise Edition only	No
db.x2iezn.8xlarge	No	No	No	No	Enterprise Edition only	No
db.x2iezn.6xlarge	No	No	No	No	Enterprise Edition only	No
db.x2iezn.4xlarge	No	No	No	No	Enterprise Edition and Standard Edition 2 (SE2)	No
db.x2iezn.2xlarge	No	No	No	No	Enterprise Edition and Standard Edition 2 (SE2)	No

db.x1e – memory-optimized instance classes

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.x1e.32xlarge	No	No	Deprecated	No	Deprecated	No
db.x1e.16xlarge	No	No	Deprecated	No	Deprecated	No
db.x1e.8xlarge	No	No	Deprecated	No	Deprecated	No
db.x1e.4xlarge	No	No	Deprecated	No	Deprecated	No

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.x1e.2xlarge	No	No	Deprecated	No	Deprecated	No
db.x1e.xlarge	No	No	Deprecated	No	Deprecated	No

db.x1 – memory-optimized instance classes

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.x1.32xlarge	No	No	Deprecated	No	Deprecated	No
db.x1.16xlarge	No	No	Deprecated	No	Deprecated	No

db.r8g – memory-optimized instance classes powered by AWS Graviton4 processors

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.r8g.48xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r8g.24xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher,	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
		10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher				higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r8g.16xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r8g.12xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r8g.8xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.r8g.4xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r8g.2xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r8g.xlarge	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r8g.large	No	MariaDB 11.8.3 and higher, 11.4.3 and higher, 10.11.7 and higher, 10.6.13 and higher, 10.5.20 and higher, and 10.4.29 and higher	No	MySQL 8.0.32 and higher	No	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher

db.r7i – memory-optimized instance classes preconfigured for high memory, storage, and I/O

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.r7i.8xlarge.tpc2.mem3x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.8xlarge.tpc2.mem2x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.6xlarge.tpc2.mem4x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.6xlarge.tpc2.mem2x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.r7i.4xlarge.tpc2.mem4x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.4xlarge.tpc2.mem3x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.4xlarge.tpc2.mem2x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.3xlarge.tpc2.mem4x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.r7i.2xlarge.tpc2.mem8x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.2xlarge.tpc2.mem4x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.xlarge.tpc2.mem4x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No
db.r7i.xlarge.tpc2.mem2x	No	No	No	No	BYOL only, Enterprise Edition and Standard Edition 2	No

db.r7i – memory-optimized instance classes powered by 4th generation Intel Xeon Scalable processors

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.r7i.48 xlarge	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r7i.24 xlarge	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r7i.16 xlarge	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r7i.12 xlarge	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
						higher, and 13.11 and higher
db.r7i.8x large	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r7i.4x large	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, all edition	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r7i.2x large	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, all edition	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r7i.xl arge	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, all edition	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Oracle	PostgreSQL
db.r7i.la rge	Db 11	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.0.32 and higher	BYOL only, all edition	PostgreSQL version 17.1 and higher, 16.1 and higher, 15.4 and higher, 14.9 and higher, and 13.11 and higher
db.r7i.me tal-48xl	No	No	No	No	BYOL only, Enterpı e Edition only	No
db.r7i.me tal-24xl	No	No	No	No	BYOL only, Enterpı e Edition only	No

db.r7g – memory-optimized instance classes powered by AWS Graviton3 processors

Instance class	Dl	MariaDB	Micros SQL Serve	MySQL	Ora	PostgreSQL
db.r7g.1 xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions,

Instance class	Dl	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
						and 13.4 and higher 13 versions
db.r7g.1 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions
db.r7g.8 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions
db.r7g.4 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions
db.r7g.2 large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions
db.r7g.x arge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r7g.la rge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.4 and higher 13 versions

db.r6g – memory-optimized instance classes powered by AWS Graviton2 processors

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6g.16 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r6g.12 xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r6g.8x large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r6g.4x large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6g.2x large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r6g.xl arge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r6g.la rge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions

db.r6gd – memory-optimized instance classes powered by AWS Graviton2 processors

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6gd. 6xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. 2xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

Instance class	DI	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6gd. xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. large	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. arge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

db.r6id – memory-optimized instance classes powered by 3rd generation Intel Xeon Scalable processors

Instance class	Db2	MariaDB	Microsc SQL Server	MySQL	Oracle	PostgreSQL
db.r6id.32xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.24xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.16xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.12xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.8xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.4xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14

Instance class	Db2	MariaDB	Microsc SQL Server	MySQL	Oracle	PostgreSQL
						versions, and 13.7 and higher 13 versions
db.r6id.2xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.metal	No	No	No	No	BYOL only, Enterpe Edition only	No

db.r6idn – memory-optimized instance classes powered by 3rd generation Intel Xeon Scalable processors

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6idn. 32xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6idn. 24xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6idn. 16xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6idn. 12xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6idn. 8xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6idn. 4xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions,

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
						and 13.7 and higher 13 versions
db.r6idn. 2xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6idn. xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions

db.r6in – memory-optimized instance classes powered by 3rd generation Intel Xeon Scalable processors

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6in.3 2xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYC only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.r6in.2 4xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYC only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
						higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.r6in.16xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.r6in.12xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.r6in.8xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYO only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6in.4xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYC only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.r6in.2xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYC only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.r6in.xlarge	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYC only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions
db.r6in.large	Yes	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYC only	All PostgreSQL 17, 16, and 15 versions, 14.3 and higher 14 versions, 13.7 and higher 13 versions, 12.11 and higher 12 versions, and 11.16 and higher 11 versions

Instance class	Db	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6in.metal	No	No	No	No	BYO only Ente e Edit only	No

db.r6i – memory-optimized instance classes preconfigured for high memory, storage, and I/O

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.r6i.8xlarge.tpc2.mem4x	No	No	No	No	Enterprise Edition only	No
db.r6i.8xlarge.tpc2.mem3x	No	No	No	No	Enterprise Edition only	No
db.r6i.6xlarge.tpc2.mem4x	No	No	No	No	Enterprise Edition only	No
db.r6i.4xlarge.tpc2.mem4x	No	No	No	No	EE and SE2 BYOL	No
db.r6i.4xlarge.tpc2.mem3x	No	No	No	No	EE and SE2 BYOL	No

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.r6i.4xlarge.tpc2.mem2x	No	No	No	No	EE and SE2 BYOL	No
db.r6i.2xlarge.tpc2.mem8x	No	No	No	No	EE and SE2 BYOL	No
db.r6i.2xlarge.tpc2.mem4x	No	No	No	No	EE and SE2 BYOL	No
db.r6i.2xlarge.tpc1.mem2x	No	No	No	No	EE and SE2 BYOL	No
db.r6i.xlarge.tpc2.mem4x	No	No	No	No	EE and SE2 BYOL	No
db.r6i.xlarge.tpc2.mem2x	No	No	No	No	EE and SE2 BYOL	No
db.r6i.large.tpc1.mem2x	No	No	No	No	EE and SE2 BYOL	No

db.r6i – memory-optimized instance classes

Instanc class	DI	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
db.r6i.3 xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions
db.r6i.2 xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions
db.r6i.1 xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions
db.r6i.1 xlarge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions

Instanc class	Dl	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
db.r6i.8 large	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions
db.r6i.4 large	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions
db.r6i.2 large	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions
db.r6i.x arge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions

Instanc class	Dl	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
db.r6i.la rge	Ye	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, and 14 versions, 13.4 and higher 13 versions, 12.8 and higher 12 versions, 11.13 and higher 11 versions, and 10.21 and higher 10 versions
db.r6i.n tal	N	No	No	No	BYC onl Ent e Edi onl	No

db.r5d – memory-optimized instance classes

Instanc class	Dl	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
db.r5d.2 xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r5d.1 xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r5d.1 xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher

Instanc class	DI	MariaDB	Micro SQL Serve	MySQL	Ora	PostgreSQL
				and 8.0		14 versions, 13.7 and higher 13 versions, and 13.4
db.r5d.8 large	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r5d.4 large	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r5d.2 large	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r5d.x arge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r5d.l rge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

db.r5b – memory-optimized instance classes preconfigured for high memory, storage, and I/O

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.r5b.8xlarge.tpc2.mem3x	No	No	No	No	Yes	No
db.r5b.6xlarge.tpc2.mem4x	No	No	No	No	Yes	No
db.r5b.4xlarge.tpc2.mem4x	No	No	No	No	Yes	No
db.r5b.4xlarge.tpc2.mem3x	No	No	No	No	Yes	No
db.r5b.4xlarge.tpc2.mem2x	No	No	No	No	Yes	No
db.r5b.2xlarge.tpc2.mem8x	No	No	No	No	Yes	No
db.r5b.2xlarge.tpc2.mem4x	No	No	No	No	Yes	No
db.r5b.2xlarge.tpc1.mem2x	No	No	No	No	Yes	No

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.r5b.xlarge.tpc2.mem4x	No	No	No	No	Yes	No
db.r5b.xlarge.tpc2.mem2x	No	No	No	No	Yes	No
db.r5b.large.tpc1.mem2x	No	No	No	No	Yes	No

db.r5b – memory-optimized instance classes

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.r5b.24xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r5b.16xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r5b.12xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions

Instance class	Db2	MariaDB	Micros SQL Server	MySQL	Orac	PostgreSQL
db.r5b.8xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	>Yes	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r5b.4xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r5b.2xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r5b.xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.r5b.large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	Yes	MySQL 8.4 and 8.0	Yes	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions

db.r5 – memory-optimized instance classes preconfigured for high memory, storage, and I/O

Instance class	Db2	MariaDB	Microsof SQL Server	MySQL	Oracle	PostgreSQL
db.r5.12xlarge.tpc2.mem2x	No	No	No	No	Yes	No

Instance class	Db2	MariaDB	Microsof SQL Server	MySQL	Oracle	PostgreSQL
db.r5.8xlarge.tpc2.mem3x	No	No	No	No	Yes	No
db.r5.6xlarge.tpc2.mem4x	No	No	No	No	Yes	No
db.r5.4xlarge.tpc2.mem4x	No	No	No	No	Yes	No
db.r5.4xlarge.tpc2.mem3x	No	No	No	No	Yes	No
db.r5.4xlarge.tpc2.mem2x	No	No	No	No	Yes	No
db.r5.2xlarge.tpc2.mem8x	No	No	No	No	Yes	No
db.r5.2xlarge.tpc2.mem4x	No	No	No	No	Yes	No
db.r5.2xlarge.tpc1.mem2x	No	No	No	No	Yes	No
db.r5.xlarge.tpc2.mem4x	No	No	No	No	Yes	No
db.r5.xlarge.tpc2.mem2x	No	No	No	No	Yes	No
db.r5.large.tpc1.mem2x	No	No	No	No	Yes	No

db.r5 – memory-optimized instance classes

Instance class	Db2	MariaD	Microsoft SQL Server	MySQ	Oracle	PostgreSQL
db.r5.24x large	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.r5.16x large	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.r5.12x large	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.r5.8xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.r5.4xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.r5.2xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.r5.xlarge	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions
db.r5.large	No	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, and 11 versions; 10.17 and higher 10 versions; and 9.6.22 and higher 9 versions

db.r4 – memory-optimized instance classes

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Oracle	PostgreSQL
db.r4.16xlarge	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
db.r4.8xlarge	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
db.r4.4xlarge	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
db.r4.2xlarge	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
db.r4.xlarge	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
db.r4.large	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated

db.r3 – memory-optimized instance classes

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQ	Oracle	PostgreSQL
db.r3.8xlarge**	No	All MariaDB 10.6, 10.5, 10.4, and 10.3 versions	Deprecated	Yes	Deprecated	Deprecated
db.r3.4xlarge	No	All MariaDB 10.6, 10.5, 10.4, and 10.3 versions	Deprecated	Yes	Deprecated	Deprecated
db.r3.2xlarge	No	All MariaDB 10.6, 10.5, 10.4, and 10.3 versions	Deprecated	Yes	Deprecated	Deprecated
db.r3.xlarge	No	All MariaDB 10.6, 10.5, 10.4, and 10.3 versions	Deprecated	Yes	Deprecated	Deprecated

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQ	Oracle	PostgreSQL
db.r3.large	No	All MariaDB 10.6, 10.5, 10.4, and 10.3 versions	Deprecated	Yes	Deprecated	Deprecated

Supported DB engines for compute-optimized instance classes

The following tables show the supported databases and database versions for the compute-optimized instance classes.

db.c6gd – compute-optimized instance classes (for Multi-AZ DB cluster deployments only)

Instance class	Db2	Maria	Microsoft SQL Server	MySQL	Orac	PostgreSQL
db.c6gd.16xlarge	No	No	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions; 14.5 and higher 14 versions; 13.4 and 13.7 and higher 13 versions
db.c6gd.12xlarge	No	No	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions; 14.5 and higher 14 versions; 13.4 and 13.7 and higher 13 versions
db.c6gd.8xlarge	No	No	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions; 14.5 and higher 14 versions; 13.4 and 13.7 and higher 13 versions
db.c6gd.4xlarge	No	No	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions; 14.5 and higher 14 versions; 13.4 and 13.7 and higher 13 versions

Instance class	Db2	Maria	Microsoft SQL Server	MySQL	Orac	PostgreSQL
db.c6gd.2xlarge	No	No	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions; 14.5 and higher 14 versions; 13.4 and 13.7 and higher 13 versions
db.c6gd.xlarge	No	No	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions; 14.5 and higher 14 versions; 13.4 and 13.7 and higher 13 versions
db.c6gd.large	No	No	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions; 14.5 and higher 14 versions; 13.4 and 13.7 and higher 13 versions
db.c6gd.medium	No	No	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, and 15 versions; 14.5 and higher 14 versions; 13.4 and 13.7 and higher 13 versions

Supported DB engines for burstable-performance instance classes

The following tables show the supported databases and database versions for the burstable-performance instance classes.

db.t4g – burstable-performance instance classes powered by AWS Graviton2 processors

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Orac	PostgreSQL
db.t4g.2xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4	No	All PostgreSQL 17, 16, 15, 14, and 13 versions;

Instance class	Db2	MariaDB	Microsoft SQL Server	MySQL	Orac	PostgreSQL
				and 8.0		and 12.7 and higher 12 versions
db.t4g.xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.t4g.large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.t4g.medium	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.t4g.small	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions
db.t4g.micro	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16, 15, 14, and 13 versions; and 12.7 and higher 12 versions

db.t3 – burstable-performance instance classes

Instance class	Db2	Maria	Microsoft SQL Server	MySQ	Oracle	PostgreSQL
db.t3.2xlarge	Yes	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, 11, and 10 versions; and 9.6.22 and higher 9 versions
db.t3.xlarge	Yes	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, 11, and 10 versions; and 9.6.22 and higher 9 versions
db.t3.large	Yes	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, 11, and 10 versions; and 9.6.22 and higher 9 versions
db.t3.medium	Yes	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, 11, and 10 versions; and 9.6.22 and higher 9 versions
db.t3.small	Yes	Yes	Yes	Yes	Yes	All PostgreSQL 17, 16, 15, 14, 13, 12, 11, and 10 versions; and 9.6.22 and higher 9 versions
db.t3.micro	No	Yes	Yes	Yes	Only on Oracle Database 12c Release 1 (12.1.0.2), which is deprecated	All PostgreSQL 17, 16, 15, 14, 13, 12, 11, and 10 versions; and 9.6.22 and higher 9 versions

db.t2 – burstable-performance instance classes

Instance class	Db2	MariaDB	Microsof SQL Server	MySQL	Oracle	PostgreSQL
db.t2.2xlarge	No	Deprecated	No	Deprecated	Deprecated	Deprecated
db.t2.xlarge	No	Deprecated	No	Deprecated	Deprecated	Deprecated
db.t2.large	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
db.t2.medium	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
db.t2.small	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
db.t2.micro	No	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated

Supported DB engines for Optimized Reads instance classes

The following tables show the supported databases and database versions for the Optimized Reads instance classes.

db.m8gd – memory-optimized instance classes that support Optimized Reads and are powered by AWS Graviton4 processors

Instance class	Db	MariaDB	Micros SQL Serve	MySQL	Ora	PostgreSQL
db.m8gc 8xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.4 and 8.0	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6

Instance class	DI	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
						and higher, 17.2 and higher, 18.1 and higher
db.m8gc 4xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.m8gc 6xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.m8gc 2xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.m8gc xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher

Instance class	DI	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.m8gc xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.m8gc xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.m8gc large	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.m8gc arge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher

db.r8gd – memory-optimized instance classes that support Optimized Reads and are powered by AWS Graviton4 processors

Instance class	DI	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r8gd. 8xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.4 and 8.0	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.r8gd. 4xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.r8gd. 6xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.r8gd. 2xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher

Instance class	DI	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r8gd. xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.r8gd. xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.r8gd. xlarge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher
db.r8gd. large	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher

Instance class	DI	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r8gd. arge	N	MariaDB 10.5.27 and higher, 10.6.19 and higher, 10.11.9 and higher, 11.4.3 and higher, 11.8.3 and higher	No	MySQL 8.0.40 and higher, 8.4.3 and higher	No	PostgreSQL 13.18 and higher, 14.17 and higher, 15.10 and higher, 16.6 and higher, 17.2 and higher, 18.1 and higher

db.r6gd – memory-optimized instance classes that support Optimized Reads and are powered by AWS Graviton2 processors

Instance class	DI	MariaDB	Micros SQL Server	MySQL	Ora	PostgreSQL
db.r6gd. 6xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. 2xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

Instance class	DI	MariaDB	Micros SQL Serve	MySQL	Ora	PostgreSQL
db.r6gd. xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. xlarge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. large	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4
db.r6gd. arge	N	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	No	All PostgreSQL 17, 16 and 15 versions, 14.5 and higher 14 versions, 13.7 and higher 13 versions, and 13.4

db.r6id – memory-optimized instance classes that support Optimized Reads and are powered by 3rd generation Intel Xeon Scalable processors

Instance class	Db2	MariaDB	Microsc SQL Server	MySQL	Oracle	PostgreSQL
db.r6id.32xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.24xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.16xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.12xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.8xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	EE and BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.4xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14

Instance class	Db2	MariaDB	Microsc SQL Server	MySQL	Oracle	PostgreSQL
						versions, and 13.7 and higher 13 versions
db.r6id.2xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.xlarge	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.large	No	MariaDB 11.8, 11.4, 10.11, 10.6, 10.5, and 10.4	No	MySQL 8.4 and 8.0	BYOL only	All PostgreSQL 17, 16, and 15 versions, 14.5 and higher 14 versions, and 13.7 and higher 13 versions
db.r6id.metal	No	No	No	No	BYOL only, Enterpe Edition only	No

Determining DB instance class support in AWS Regions

To determine the DB instance classes supported by each DB engine in a specific AWS Region, you can take one of several approaches. You can use the AWS Management Console, the Amazon RDS

Pricing page, or the describe-orderable-db-instance-options command for the AWS Command Line Interface (AWS CLI).

> **Note**
>
> When you perform operations with the AWS Management Console, it automatically shows the supported DB instance classes for a specific DB engine, DB engine version, and AWS Region. Examples of the operations that you can perform include creating and modifying a DB instance.

Contents

- Using the Amazon RDS pricing page to determine DB instance class support in AWS Regions
- Using the AWS CLI to determine DB instance class support in AWS Regions
  - Listing the DB instance classes that are supported by a specific DB engine version in an AWS Region
  - Listing the DB engine versions that support a specific DB instance class in an AWS Region
  - Listing AWS Regions that support a specific DB engine and instance class

Using the Amazon RDS pricing page to determine DB instance class support in AWS Regions

You can use the Amazon RDS Pricing page to determine the DB instance classes supported by each DB engine in a specific AWS Region.

To use the pricing page to determine the DB instance classes supported by each engine in a Region

1. Go to Amazon RDS Pricing.
2. In the **AWS Pricing Calculator for Amazon RDS** section, choose **Create your custom estimate now**.
3. In **Choose a Region**, choose an AWS Region.
4. In **Find a Service**, enter **`Amazon RDS`**.
5. Choose **Configure** for your configuration option and DB engine.
6. Use the section for compatible instances to view the supported DB instance classes.

7. (Optional) Choose other options in the calculator, and then choose **Save and view summary** or **Save and add service**.

Using the AWS CLI to determine DB instance class support in AWS Regions

You can use the AWS CLI to determine which DB instance classes are supported for specific DB engines and DB engine versions in an AWS Region. The following table shows the valid DB engine values.

Engine names	Engine values in CLI commands	More information about versions
Db2	`db2-ae` `db2-se`	Db2 on Amazon RDS versions
MariaDB	`mariadb`	MariaDB on Amazon RDS versions

Microsoft SQL Server	`sqlserver-ee`
	`sqlserver-se`
	`sqlserver-ex`
	`sqlserver-web`
		Microsoft SQL Server versions on Amazon RDS

MySQL	`mysql`	MySQL on Amazon RDS versions
Oracle	`oracle-ee` `oracle-se2`	*Amazon RDS for Oracle Release Notes*
PostgreSQL	`postgres`	Available PostgreSQL database versions

For information about AWS Region names, see AWS Regions.

The following examples demonstrate how to determine DB instance class support in an AWS Region using the describe-orderable-db-instance-options AWS CLI command.

> **Note**
>
> To limit the output, the following examples show results only for the General Purpose SSD (gp2) storage type. If necessary, you can change the storage type to General Purpose SSD (gp3), Provisioned IOPS (io1), Provisioned IOPS Block Express (io2), or magnetic (standard) in the commands.

Topics

- Listing the DB instance classes that are supported by a specific DB engine version in an AWS Region
- Listing the DB engine versions that support a specific DB instance class in an AWS Region
- Listing AWS Regions that support a specific DB engine and instance class

Listing the DB instance classes that are supported by a specific DB engine version in an AWS Region

To list the DB instance classes that are supported by a specific DB engine version in an AWS Region, run the following command.

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 \
    --query "*[].{DBInstanceClass:DBInstanceClass,StorageType:StorageType}|[?
StorageType=='gp2']|[].{DBInstanceClass:DBInstanceClass}" \
    --output text \
    --region region
```

For Windows:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 ^
    --query "*[].{DBInstanceClass:DBInstanceClass,StorageType:StorageType}|[?
StorageType=='gp2']|[].{DBInstanceClass:DBInstanceClass}" ^
    --output text ^
    --region region
```

For example, the following command lists the supported DB instance classes for version 13.6 of the RDS for PostgreSQL DB engine in US East (N. Virginia).

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options --engine postgres --engine-version 15.4
 \
    --query "*[].{DBInstanceClass:DBInstanceClass,StorageType:StorageType}|[?
StorageType=='gp2']|[].{DBInstanceClass:DBInstanceClass}" \
    --output text \
    --region us-east-1
```

For Windows:

```
aws rds describe-orderable-db-instance-options --engine postgres --engine-version 15.4
 ^
    --query "*[].{DBInstanceClass:DBInstanceClass,StorageType:StorageType}|[?
StorageType=='gp2']|[].{DBInstanceClass:DBInstanceClass}" ^
    --output text ^
    --region us-east-1
```

Listing the DB engine versions that support a specific DB instance class in an AWS Region

To list the DB engine versions that support a specific DB instance class in an AWS Region, run the following command.

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class \
    --query "*[].{EngineVersion:EngineVersion,StorageType:StorageType}|[?
StorageType=='gp2']|[].{EngineVersion:EngineVersion}" \
    --output text \
    --region region
```

For Windows:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class ^
    --query "*[].{EngineVersion:EngineVersion,StorageType:StorageType}|[?
StorageType=='gp2']|[].{EngineVersion:EngineVersion}" ^
    --output text ^
```

```
    --region region
```

For example, the following command lists the DB engine versions of the RDS for PostgreSQL DB engine that support the db.r5.large DB instance class in US East (N. Virginia).

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options --engine postgres --db-instance-class
 db.m7g.large \
    --query "*[].{EngineVersion:EngineVersion,StorageType:StorageType}|[?
StorageType=='gp2']|[].{EngineVersion:EngineVersion}" \
    --output text \
    --region us-east-1
```

For Windows:

```
aws rds describe-orderable-db-instance-options --engine postgres --db-instance-class
 db.m7g.large ^
    --query "*[].{EngineVersion:EngineVersion,StorageType:StorageType}|[?
StorageType=='gp2']|[].{EngineVersion:EngineVersion}" ^
    --output text ^
    --region us-east-1
```

Listing AWS Regions that support a specific DB engine and instance class

The following bash script lists all the AWS Regions that support the specified combination of DB engine and instance class.

```
#!/usr/bin/env bash
# Usage: check_region_support.sh <db-engine> <db-instance-class>

if [ $# -ne 2 ]; then
  echo "Usage: $0 <db-engine> <db-instance-class>"
  exit 1
fi
ENGINE="$1"
INSTANCE_CLASS="$2"
REGIONS=$(aws ec2 describe-regions --query "Regions[].RegionName" --output text)
for region in $REGIONS; do
  supported_count=$(aws rds describe-orderable-db-instance-options \
    --region "$region" \
    --engine "$ENGINE" \
```

```
    --db-instance-class "$INSTANCE_CLASS" \
    --query 'length(OrderableDBInstanceOptions)' \
    --output text 2>/dev/null || echo "0")
  if [ "$supported_count" -gt 0 ]; then
    echo "$region supports $INSTANCE_CLASS for $ENGINE."
  else
    echo "$region doesn't support $INSTANCE_CLASS for $ENGINE."
  fi
done
```

The following sample output checks Region support for RDS for MySQL using the db.r8g.large instance class.

```
./check_region_support.sh mysql db.r8g.large
ap-south-1 doesn't support db.r8g.large for mysql.
eu-north-1 doesn't support db.r8g.large for mysql.
eu-west-3 doesn't support db.r8g.large for mysql.
eu-west-2 doesn't support db.r8g.large for mysql.
eu-west-1 doesn't support db.r8g.large for mysql.
ap-northeast-3 doesn't support db.r8g.large for mysql.
ap-northeast-2 doesn't support db.r8g.large for mysql.
ap-northeast-1 doesn't support db.r8g.large for mysql.
ca-central-1 doesn't support db.r8g.large for mysql.
sa-east-1 doesn't support db.r8g.large for mysql.
ap-southeast-1 doesn't support db.r8g.large for mysql.
ap-southeast-2 doesn't support db.r8g.large for mysql.
eu-central-1 supports db.r8g.large for mysql.
us-east-1 supports db.r8g.large for mysql.
us-east-2 supports db.r8g.large for mysql.
us-west-1 doesn't support db.r8g.large for mysql.
us-west-2 supports db.r8g.large for mysql.
```

Changing your DB instance class

You can change the CPU and memory available to a DB instance by changing its DB instance class. To change the DB instance class, modify your DB instance by following the instructions in Modifying an Amazon RDS DB instance.

Configuring the processor for a DB instance class in RDS for Oracle

Amazon RDS DB instance classes support Intel Hyper-Threading Technology, which enables multiple threads to run concurrently on a single Intel Xeon CPU core. Each thread is represented as

a virtual CPU (vCPU) on the DB instance. A DB instance has a default number of CPU cores, which varies according to DB instance class. For example, a db.m4.xlarge DB instance class has two CPU cores and two threads per core by default—four vCPUs in total.

> **Note**
>
> Each vCPU is a hyperthread of an Intel Xeon CPU core.

Topics

- Overview of processor configuration for RDS for Oracle
- DB instance classes that support processor configuration
- Setting the CPU cores and threads per CPU core for a DB instance class

Overview of processor configuration for RDS for Oracle

When you use RDS for Oracle, you can usually find a DB instance class that has a combination of memory and number of vCPUs to suit your workloads. However, you can also specify the following processor features to optimize your RDS for Oracle DB instance for specific workloads or business needs:

- **Number of CPU cores** – You can customize the number of CPU cores for the DB instance. You might do this to potentially optimize the licensing costs of your software with a DB instance that has sufficient amounts of RAM for memory-intensive workloads but fewer CPU cores.
- **Threads per core** – You can disable Intel Hyper-Threading Technology by specifying a single thread per CPU core. You might do this for certain workloads, such as high-performance computing (HPC) workloads.

You can control the number of CPU cores and threads for each core separately. You can set one or both in a request. After a setting is associated with a DB instance, the setting persists until you change it.

The processor settings for a DB instance are associated with snapshots of the DB instance. When a snapshot is restored, its restored DB instance uses the processor feature settings used when the snapshot was taken.

If you modify the DB instance class for a DB instance with nondefault processor settings, either specify default processor settings or explicitly specify processor settings at modification. This requirement ensures that you are aware of the third-party licensing costs that might be incurred when you modify the DB instance.

There is no additional or reduced charge for specifying processor features on an RDS for Oracle DB instance. You're charged the same as for DB instances that are launched with default CPU configurations.

DB instance classes that support processor configuration

You can configure the number of CPU cores and threads per core only when the following conditions are met:

- You're configuring an RDS for Oracle DB instance. For information about the DB instance classes supported by different Oracle Database editions, see RDS for Oracle DB instance classes.
- Your DB instance is using the Bring Your Own License (BYOL) licensing option of RDS for Oracle. For more information about Oracle licensing options, see RDS for Oracle licensing options.
- Your DB instance doesn't belong to one of the db.r5 or db.r5b instance classes that have predefined processor configurations. These instance classes have names in the form db.r5.*instance_size*.tpc*threads_per_core*.mem*ratio* or db.r5b.*instance_size*.tpc*threads_per_core*.mem*ratio*. For example, db.r5b.xlarge.tpc2.mem4x is preconfigured with 2 threads per core (tpc2) and 4x as much memory as the standard db.r5b.xlarge instance class. You can't configure the processor features of these optimized instance classes. For more information, see Supported RDS for Oracle DB instance classes.

You can use the following AWS CLI command to show the default vCPUs, cores, threads per core, and valid numbers of cores for an instance class. Replace *r7i.48xlarge* in the sample command with the name of your instance class.

```
aws ec2 describe-instance-types \
    --instance-types r7i.48xlarge \
    --query '{
        DefaultVCPUs: InstanceTypes[0].VCpuInfo.DefaultVCpus,
        DefaultCores: InstanceTypes[0].VCpuInfo.DefaultCores,
        DefaultThreadsPerCore: InstanceTypes[0].VCpuInfo.DefaultThreadsPerCore,
        ValidCores: InstanceTypes[0].VCpuInfo.ValidCores
```

```
    }' \
    --output json
```

In the following table, you can find the DB instance classes that support setting a number of CPU cores and CPU threads per core. You can also find the default value and the valid values for the number of CPU cores and CPU threads per core for each DB instance class.

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.m6i – memory-optimized instance classes					
db.m6i.large	2	1	2	1	1, 2
db.m6i.xlarge	4	2	2	2	1, 2
db.m6i.2xlarge	8	4	2	2, 4	1, 2
db.m6i.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.m6i.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.m6i.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.m6i.12xlarge	48	24	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2
db.m6i.16xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.m6i.24xlarge	96	48	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28,	1, 2

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
				30, 32, 34, 36, 38, 40, 42, 44, 46, 48	
db.m6i.32xlarge	128	64	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64	1, 2
db.m5 – general-purpose instance classes					
db.m5.large	2	1	2	1	1, 2
db.m5.xlarge	4	2	2	2	1, 2
db.m5.2xlarge	8	4	2	2, 4	1, 2
db.m5.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.m5.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.m5.12xlarge	48	24	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2
db.m5.16xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.m5.24xlarge	96	48	2	4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48	1, 2
db.m5d – general-purpose instance classes					

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.m5d.large	2	1	2	1	1, 2
db.m5d.xlarge	4	2	2	2	1, 2
db.m5d.2xlarge	8	4	2	2, 4	1, 2
db.m5d.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.m5d.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.m5d.12xlarge	48	24	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2
db.m5d.16xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.m5d.24xlarge	96	48	2	4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48	1, 2
db.m4 – general-purpose instance classes					
db.m4.10xlarge	40	20	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20	1, 2
db.m4.16xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.r7i – memory-optimized instance classes					
db.r7i.large	2	1	2	1	1, 2
db.r7i.xlarge	4	2	2	1, 2	1, 2
db.r7i.2xlarge	8	4	2	1, 2, 3, 4	1, 2
db.r7i.4xlarge	16	8	2	1, 2, 3, 4, 5, 6, 7, 8	1, 2
db.r7i.8xlarge	32	16	2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16	1, 2
db.r7i.12 xlarge	48	24	2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24	1, 2
db.r7i.16 xlarge	64	32	2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32	1, 2
db.r7i.24 xlarge	96	48	2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48	1, 2

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.r7i.48 xlarge	192	96	2	4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96	1, 2
db.r6i – memory-optimized instance classes					
db.r6i.large	2	1	2	1	1, 2
db.r6i.xlarge	4	2	2	1, 2	1, 2
db.r6i.2xlarge	8	4	2	2, 4	1, 2
db.r6i.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.r6i.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.r6i.12 xlarge	48	24	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2
db.r6i.16 xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.r6i.24 xlarge	96	48	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48	1, 2

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.r6i.32 xlarge	128	64	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64	1, 2
db.r5 – memory-optimized instance classes					
db.r5.large	2	1	2	1	1, 2
db.r5.xlarge	4	2	2	2	1, 2
db.r5.2xlarge	8	4	2	2, 4	1, 2
db.r5.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.r5.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.r5.12xlarge	48	24	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2
db.r5.16xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.r5.24xlarge	96	48	2	4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48	1, 2
db.r5 – memory-optimized instance classes					
db.r5b.large	2	1	2	1	1, 2
db.r5b.xlarge	4	2	2	2	1, 2

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.r5b.2xlarge	8	4	2	2, 4	1, 2
db.r5b.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.r5b.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.r5b.12 xlarge	48	24	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2
db.r5b.16 xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.r5b.24 xlarge	96	48	2	4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48	1, 2
db.r5d – memory-optimized instance classes					
db.r5d.large	2	1	2	1	1, 2
db.r5d.xlarge	4	2	2	2	1, 2
db.r5d.2xlarge	8	4	2	2, 4	1, 2
db.r5d.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.r5d.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.r5d.12 xlarge	48	24	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.r5d.16 xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.r5d.24 xlarge	96	48	2	4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48	1, 2
db.r4 – memory-optimized instance classes					
db.r4.large	2	1	2	1	1, 2
db.r4.xlarge	4	2	2	1, 2	1, 2
db.r4.2xlarge	8	4	2	1, 2, 3, 4	1, 2
db.r4.4xlarge	16	8	2	1, 2, 3, 4, 5, 6, 7, 8	1, 2
db.r4.8xlarge	32	16	2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16	1, 2
db.r4.16xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.r3 – memory-optimized instance classes					
db.r3.large	2	1	2	1	1, 2
db.r3.xlarge	4	2	2	1, 2	1, 2
db.r3.2xlarge	8	4	2	1, 2, 3, 4	1, 2
db.r3.4xlarge	16	8	2	1, 2, 3, 4, 5, 6, 7, 8	1, 2

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.r3.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.x2idn – memory-optimized instance classes					
db.x2idn.16xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.x2idn.24xlarge	96	48	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48	1, 2
db.x2idn.32xlarge	128	64	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64	1, 2
db.x2iedn – memory-optimized instance classes					
db.x2iedn.xlarge	4	2	2	1, 2	1, 2
db.x2iedn.2xlarge	8	4	2	2, 4	1, 2
db.x2iedn.4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.x2iedn.8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.x2iedn .16xlarge	64	32	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32	1, 2
db.x2iedn .24xlarge	96	48	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48	1, 2
db.x2iedn .32xlarge	128	64	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64	1, 2
db.x2iezn – memory-optimized instance classes					
db.x2iezn .2xlarge	8	4	2	2, 4	1, 2
db.x2iezn .4xlarge	16	8	2	2, 4, 6, 8	1, 2
db.x2iezn .6xlarge	24	12	2	2, 4, 6, 8, 10, 12	1, 2
db.x2iezn .8xlarge	32	16	2	2, 4, 6, 8, 10, 12, 14, 16	1, 2
db.x2iezn .12xlarge	48	24	2	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2
db.z1d – memory-optimized instance classes					

DB instance class	Default vCPUs	Default CPU cores	Default threads per core	Valid number of CPU cores	Valid number of threads per core
db.z1d.large	2	1	2	1	1, 2
db.z1d.xlarge	4	2	2	2	1, 2
db.z1d.2xlarge	8	4	2	2, 4	1, 2
db.z1d.3xlarge	12	6	2	2, 4, 6	1, 2
db.z1d.6xlarge	24	12	2	2, 4, 6, 8, 10, 12	1, 2
db.z1d.12 xlarge	48	24	2	4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24	1, 2

Note

You can use AWS CloudTrail to monitor and audit changes to the process configuration of Amazon RDS for Oracle DB instances. For more information about using CloudTrail, see Monitoring Amazon RDS API calls in AWS CloudTrail.

Setting the CPU cores and threads per CPU core for a DB instance class

You can configure the number of CPU cores and threads per core for the DB instance class when you perform the following operations:

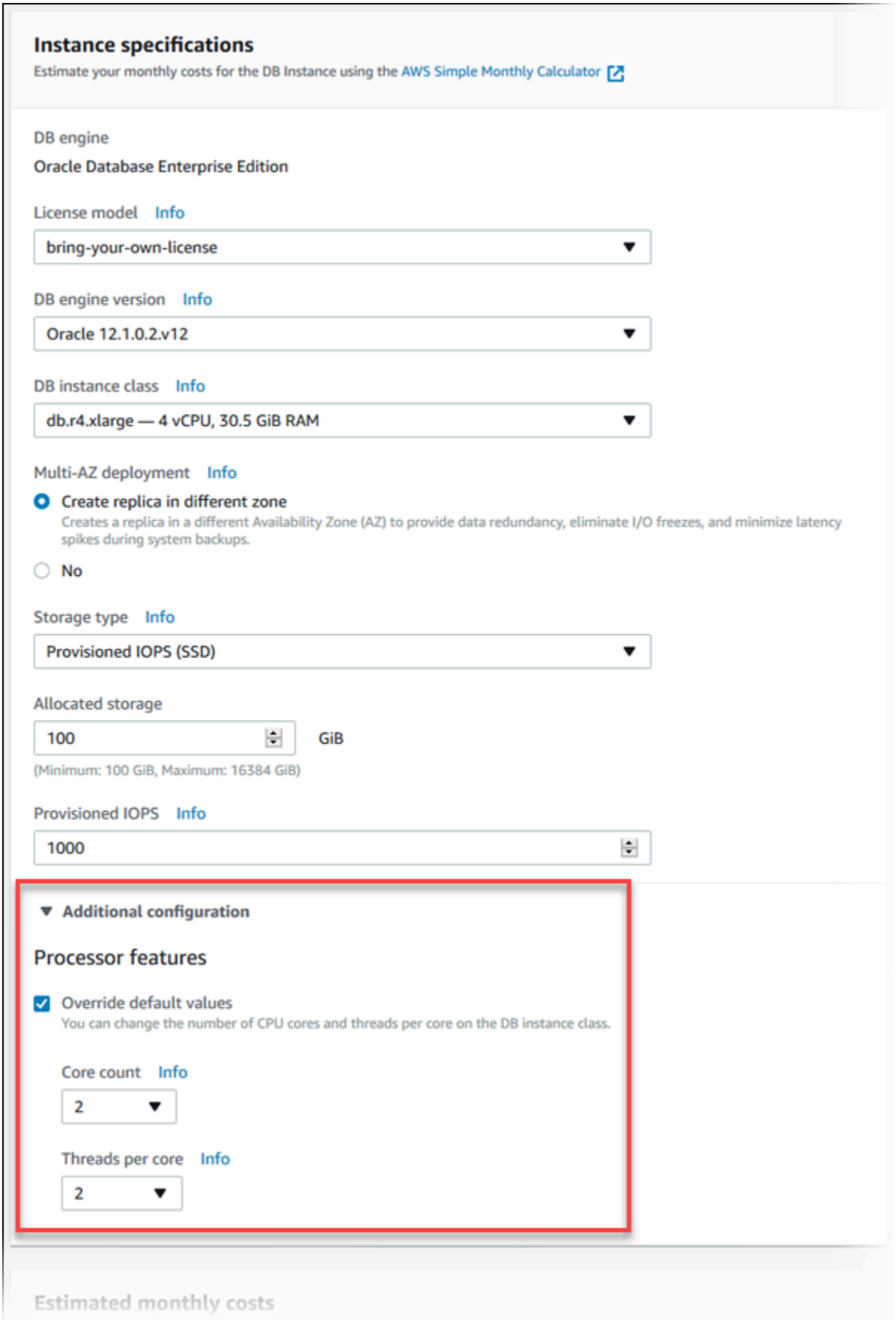
- Creating an Amazon RDS DB instance
- Modifying an Amazon RDS DB instance
- Restoring to a DB instance
- Restoring a DB instance to a specified time for Amazon RDS

> **Note**
>
> When you modify a DB instance to configure the number of CPU cores or threads per core, there is a brief DB instance outage.

You can set the CPU cores and the threads per CPU core for a DB instance class using the AWS Management Console, the AWS CLI, or the RDS API.

Console

When you are creating, modifying, or restoring a DB instance, you set the DB instance class in the AWS Management Console. The **Instance specifications** section shows options for the processor. The following image shows the processor features options.

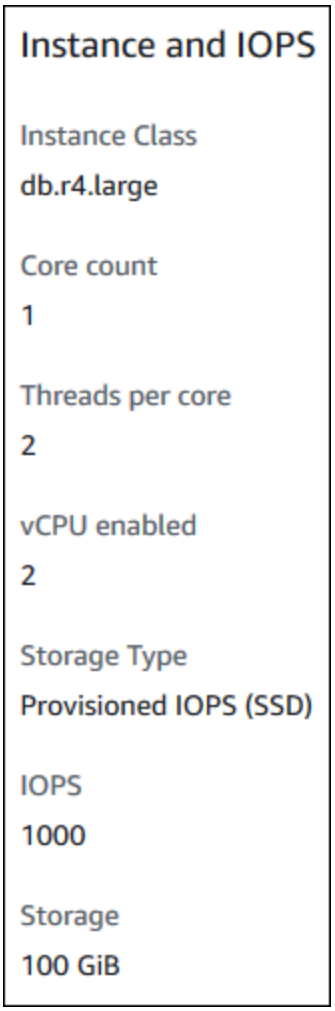

Instance specifications
Estimate your monthly costs for the DB Instance using the AWS Simple Monthly Calculator
DB engine
Oracle Database Enterprise Edition
License model Info
bring-your-own-license
DB engine version Info
Oracle 12.1.0.2.v12
DB instance class Info
db.r4.xlarge — 4 vCPU, 30.5 GiB RAM
Multi-AZ deployment Info
Create replica in different zone
Creates a replica in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.
No
Storage type Info
Provisioned IOPS (SSD)
Allocated storage
100
GiB
(Minimum: 100 GiB, Maximum: 16384 GiB)
Provisioned IOPS Info
1000
Additional configuration
Processor features
Override default values
You can change the number of CPU cores and threads per core on the DB instance class.
Core count Info
2
Threads per core Info
2
Estimated monthly costs

Set the following options to the appropriate values for your DB instance class under **Processor features**:

- **Core count –** Set the number of CPU cores using this option. The value must be equal to or less than the maximum number of CPU cores for the DB instance class.
- **Threads per core** – Specify **2** to enable multiple threads per core, or specify **1** to disable multiple threads per core.

When you modify or restore a DB instance, you can also set the CPU cores and the threads per CPU core to the defaults for the instance class.

When you view the details for a DB instance in the console, you can view the processor information for its DB instance class on the **Configuration** tab. The following image shows a DB instance class with one CPU core and multiple threads per core enabled.



For Oracle DB instances, the processor information only appears for Bring Your Own License (BYOL) DB instances.

AWS CLI

You can set the processor features for a DB instance when you run one of the following AWS CLI commands:

- create-db-instance
- modify-db-instance
- restore-db-instance-from-db-snapshot
- restore-db-instance-from-s3
- restore-db-instance-to-point-in-time

To configure the processor of a DB instance class for a DB instance by using the AWS CLI, include the `--processor-features` option in the command. Specify the number of CPU cores with the `coreCount` feature name, and specify whether multiple threads per core are enabled with the `threadsPerCore` feature name.

The option has the following syntax.

```
--processor-features "Name=coreCount,Value=<value>" "Name=threadsPerCore,Value=<value>"
```

The following are examples that configure the processor:

Examples

- Setting the number of CPU cores for a DB instance
- Setting the number of CPU cores and disabling multiple threads for a DB instance
- Viewing the valid processor values for a DB instance class
- Returning to default processor settings for a DB instance
- Returning to the default number of CPU cores for a DB instance
- Returning to the default number of threads per core for a DB instance

Setting the number of CPU cores for a DB instance

Example

The following example modifies `mydbinstance` by setting the number of CPU cores to 4. The changes are applied immediately by using `--apply-immediately`. If you want to apply the changes during the next scheduled maintenance window, omit the `--apply-immediately` option.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --processor-features "Name=coreCount,Value=4" \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --processor-features "Name=coreCount,Value=4" ^
    --apply-immediately
```

Setting the number of CPU cores and disabling multiple threads for a DB instance

Example

The following example modifies `mydbinstance` by setting the number of CPU cores to 4 and disabling multiple threads per core. The changes are applied immediately by using `--apply-immediately`. If you want to apply the changes during the next scheduled maintenance window, omit the `--apply-immediately` option.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --processor-features "Name=coreCount,Value=4" "Name=threadsPerCore,Value=1" \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --processor-features "Name=coreCount,Value=4" "Name=threadsPerCore,Value=1" ^
    --apply-immediately
```

Viewing the valid processor values for a DB instance class

Example

You can view the valid processor values for a particular DB instance class by running the describe-orderable-db-instance-options command and specifying the instance class for the `--db-instance-class` option. For example, the output for the following command shows the processor options for the db.r3.large instance class.

```
aws rds describe-orderable-db-instance-options --engine oracle-ee --db-instance-class
 db.r3.large
```

Following is sample output for the command in JSON format.

```
    {
                "SupportsIops": true,
                "MaxIopsPerGib": 50.0,
                "LicenseModel": "bring-your-own-license",
                "DBInstanceClass": "db.r3.large",
                "SupportsIAMDatabaseAuthentication": false,
                "MinStorageSize": 100,
                "AvailabilityZones": [
                    {
                        "Name": "us-west-2a"
                    },
                    {
                        "Name": "us-west-2b"
                    },
                    {
                        "Name": "us-west-2c"
                    }
                ],
                "EngineVersion": "12.1.0.2.v2",
                "MaxStorageSize": 32768,
                "MinIopsPerGib": 1.0,
                "MaxIopsPerDbInstance": 40000,
                "ReadReplicaCapable": false,
```

```
                "AvailableProcessorFeatures": [
                    {
                        "Name": "coreCount",
                        "DefaultValue": "1",
                        "AllowedValues": "1"
                    },
                    {
                        "Name": "threadsPerCore",
                        "DefaultValue": "2",
                        "AllowedValues": "1,2"
                    }
                ],
                "SupportsEnhancedMonitoring": true,
                "SupportsPerformanceInsights": false,
                "MinIopsPerDbInstance": 1000,
                "StorageType": "io1",
                "Vpc": false,
                "SupportsStorageEncryption": true,
                "Engine": "oracle-ee",
                "MultiAZCapable": true
    }
```

In addition, you can run the following commands for DB instance class processor information:

- describe-db-instances – Shows the processor information for the specified DB instance.
- describe-db-snapshots – Shows the processor information for the specified DB snapshot.
- describe-valid-db-instance-modifications – Shows the valid modifications to the processor for the specified DB instance.

In the output of the preceding commands, the values for the processor features are not null only if the following conditions are met:

- You are using an RDS for Oracle DB instance.
- Your RDS for Oracle DB instance supports changing processor values.
- The current CPU core and thread settings are set to nondefault values.

If the preceding conditions aren't met, you can get the instance type using describe-db-instances. You can get the processor information for this instance type by running the EC2 operation describe-instance-types.

Returning to default processor settings for a DB instance

Example

The following example modifies `mydbinstance` by returning its DB instance class to the default processor values for it. The changes are applied immediately by using `--apply-immediately`. If you want to apply the changes during the next scheduled maintenance window, omit the `--apply-immediately` option.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --use-default-processor-features \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --use-default-processor-features ^
    --apply-immediately
```

Returning to the default number of CPU cores for a DB instance

Example

The following example modifies `mydbinstance` by returning its DB instance class to the default number of CPU cores for it. The threads per core setting isn't changed. The changes are applied immediately by using `--apply-immediately`. If you want to apply the changes during the next scheduled maintenance window, omit the `--apply-immediately` option.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --processor-features "Name=coreCount,Value=DEFAULT" \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
```

```
    --db-instance-identifier mydbinstance ^
    --processor-features "Name=coreCount,Value=DEFAULT" ^
    --apply-immediately
```

Returning to the default number of threads per core for a DB instance

Example

The following example modifies `mydbinstance` by returning its DB instance class to the default number of threads per core for it. The number of CPU cores setting isn't changed. The changes are applied immediately by using `--apply-immediately`. If you want to apply the changes during the next scheduled maintenance window, omit the `--apply-immediately` option.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --processor-features "Name=threadsPerCore,Value=DEFAULT" \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --processor-features "Name=threadsPerCore,Value=DEFAULT" ^
    --apply-immediately
```

RDS API

You can set the processor features for a DB instance when you call one of the following Amazon RDS API operations:

- CreateDBInstance
- ModifyDBInstance
- RestoreDBInstanceFromDBSnapshot
- RestoreDBInstanceFromS3
- RestoreDBInstanceToPointInTime

To configure the processor features of a DB instance class for a DB instance by using the Amazon RDS API, include the `ProcessFeatures` parameter in the call.

The parameter has the following syntax.

```
ProcessFeatures "Name=coreCount,Value=<value>" "Name=threadsPerCore,Value=<value>"
```

Specify the number of CPU cores with the `coreCount` feature name, and specify whether multiple threads per core are enabled with the `threadsPerCore` feature name.

You can view the valid processor values for a particular DB instance class by running the DescribeOrderableDBInstanceOptions operation and specifying the instance class for the `DBInstanceClass` parameter. You can also use the following operations:

- DescribeDBInstances – Shows the processor information for the specified DB instance.
- DescribeDBSnapshots – Shows the processor information for the specified DB snapshot.
- DescribeValidDBInstanceModifications – Shows the valid modifications to the processor for the specified DB instance.

In the output of the preceding operations, the values for the processor features are not null only if the following conditions are met:

- You are using an RDS for Oracle DB instance.
- Your RDS for Oracle DB instance supports changing processor values.
- The current CPU core and thread settings are set to nondefault values.

If the preceding conditions aren't met, you can get the instance type using DescribeDBInstances. You can get the processor information for this instance type by running the EC2 operation DescribeInstanceTypes.

Hardware specifications for DB instance classes

In the tables in this section, you can find hardware details about the Amazon RDS DB instance classes.

For information about Amazon RDS DB engine support for each DB instance class, see Supported DB engines for DB instance classes.

Topics

- Hardware terminology for DB instance classes

- Hardware specifications for the general-purpose instance classes
- Hardware specifications for the memory-optimized instance classes
- Hardware specifications for the compute-optimized instance classes
- Hardware specifications for the burstable-performance instance classes

> **Note**
>
> RDS for SQL Server supports Optimize CPU starting with 7th generation instance classes (such as db.m7i and db.r7i). The vCPU counts documented below may differ for these instance classes. For accurate vCPU counts, refer to DB instance classes that support Optimize CPU.

Hardware terminology for DB instance classes

The following terminology is used to describe hardware specifications for DB instance classes:

vCPU

The number of virtual central processing units (CPUs). A *virtual CPU* is a unit of capacity that you can use to compare DB instance classes. Instead of purchasing or leasing a particular processor to use for several months or years, you are renting capacity by the hour. Our goal is to make a consistent and specific amount of CPU capacity available, within the limits of the actual underlying hardware.

ECU

The relative measure of the integer processing power of an Amazon EC2 instance. To make it easy for developers to compare CPU capacity between different instance classes, we have defined an Amazon EC2 Compute Unit. The amount of CPU that is allocated to a particular instance is expressed in terms of these EC2 Compute Units. One ECU currently provides CPU capacity equivalent to a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor.

Memory (GiB)

The RAM, in gibibytes, allocated to the DB instance. There is often a consistent ratio between memory and vCPU. As an example, take the db.r4 instance class, which has a memory to vCPU ratio similar to the db.r5 instance class. However, for most use cases the db.r5 instance class provides better, more consistent performance than the db.r4 instance class.

EBS-optimized

The DB instance uses an optimized configuration stack and provides additional, dedicated capacity for I/O. This optimization provides the best performance by minimizing contention between I/O and other traffic from your instance. For more information about Amazon EBS–optimized instances, see Amazon EBS–Optimized instances in the *Amazon EC2 User Guide.*

EBS-optimized instances have a baseline and maximum IOPS rate. The maximum IOPS rate is enforced at the DB instance level. A set of EBS volumes that combine to have an IOPS rate that is higher than the maximum can't exceed the instance-level threshold. For example, if the maximum IOPS for a particular DB instance class is 40,000, and you attach four 64,000 IOPS EBS volumes, the maximum IOPS is 40,000 rather than 256,000. For the IOPS maximum specific to each EC2 instance type, see Supported instance types in the *Amazon EC2 User Guide for Linux Instances*.

Max. EBS bandwidth (Mbps)

The maximum EBS bandwidth in megabits per second. Divide by 8 to get the expected throughput in megabytes per second.

> **Important**
>
> General Purpose SSD (gp2) volumes for Amazon RDS DB instances have a throughput limit of 250 MiB/s in most cases. However, the throughput limit can vary depending on volume size. For more information, see Amazon EBS volume types in the *Amazon EC2 User Guide.*

Network bandwidth

The network speed relative to other DB instance classes.

Hardware specifications for the general-purpose instance classes

The following tables show the compute, memory, storage, and bandwidth specifications for the general-purpose instance classes.

db.m8g – general-purpose instance classes powered by AWS Graviton4 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m8g.48xlarge	192	—	768	EBS-optimized only	40,000	50
db.m8g.24xlarge	96	—	512	EBS-optimized only	30,000	40
db.m8g.16xlarge	64	—	384	EBS-optimized only	20,000	30
db.m8g.12xlarge	48	—	256	EBS-optimized only	15,000	22.5
db.m8g.8xlarge	32	—	128	EBS-optimized only	10,000	15
db.m8g.4xlarge*	16	—	64	EBS-optimized only	Up to 10,000	Up to 15
db.m8g.2xlarge*	8	—	32	EBS-optimized only	Up to 10,000	Up to 15
db.m8g.xlarge*	4	—	16	EBS-optimized only	Up to 10,000	Up to 12.5
db.m8g.large*	2	—	8	EBS-optimized only	Up to 10,000	Up to 12.5

db.m7i – general-purpose instance classes powered by 4th generation Intel Xeon Scalable processors

Instance class	vCPU	Proc core	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m7i.metal-48xl	192	96	2	—	768	EBS-optimized only	40,000	50
db.m7i.metal-24xl	96	48	1	—	384	EBS-optimized only	30,000	37.5
db.m7i.48xlarge	192	—	—	—	768	EBS-optimized only	40,000	50
db.m7i.24xlarge	96	—	—	—	384	EBS-optimized only	30,000	37.5
db.m7i.16xlarge	64	—	—	—	256	EBS-optimized only	20,000	25
db.m7i.12xlarge	48	—	—	—	192	EBS-optimized only	15,000	18.75
db.m7i.8xlarge	32	—	—	—	128	EBS-optimized only	10,000	12.5
db.m7i.4xlarge	16	—	—	—	64	EBS-optimized only	Up to 10,000	Up to 12.5
db.m7i.2xlarge	8	—	—	—	32	EBS-optimized only	Up to 10,000	Up to 12.5
db.m7i.xlarge	4	—	—	—	16	EBS-optimized only	Up to 10,000	Up to 12.5
db.m7i.large	2	—	—	—	8	EBS-optimized only	Up to 10,000	Up to 12.5

db.m7g – general-purpose instance classes powered by AWS Graviton3 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m7g.16xlarge	64	—	256	EBS-optimized only	20,000	30
db.m7g.12xlarge	48	—	192	EBS-optimized only	15,000	22.5
db.m7g.8xlarge	32	—	128	EBS-optimized only	10,000	15
db.m7g.4xlarge	16	—	64	EBS-optimized only	Up to 10,000	Up to 15
db.m7g.2xlarge*	8	—	32	EBS-optimized only	Up to 10,000	Up to 15
db.m7g.xlarge*	4	—	16	EBS-optimized only	Up to 10,000	Up to 12.5
db.m7g.large*	2	—	8	EBS-optimized only	Up to 10,000	Up to 12.5

db.m6g – general-purpose instance classes powered by AWS Graviton2 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6g.16xlarge	64	—	256	EBS-optimized only	19,000	25
db.m6g.12xlarge	48	—	192	EBS-optimized only	13,500	20

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6g.8xlarge	32	—	128	EBS-optimized only	9,000	12
db.m6g.4xlarge	16	—	64	EBS-optimized only	4,750	Up to 10
db.m6g.2xlarge*	8	—	32	EBS-optimized only	Up to 4,750	Up to 10
db.m6g.xlarge*	4	—	16	EBS-optimized only	Up to 4,750	Up to 10
db.m6g.large*	2	—	8	EBS-optimized only	Up to 4,750	Up to 10

db.m6gd – general-purpose instance classes powered by AWS Graviton2 processors and SSD storage

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6gd.16xlarge	64	—	256	2 x 1900 NVMe SSD	19,000	25
db.m6gd.12xlarge	48	—	192	2 x 1425 NVMe SSD	13,500	20
db.m6gd.8xlarge	32	—	128	1 x 1900 NVMe SSD	9,000	12
db.m6gd.4xlarge*	16	—	64	1 x 950 NVMe SSD	4,750	Up to 10

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6gd.2xlarge*	8	—	32	1 x 474 NVMe SSD	Up to 4,750	Up to 10
db.m6gd.xlarge*	4	—	16	1 x 237 NVMe SSD	Up to 4,750	Up to 10
db.m6gd.large*	2	—	8	1 x 118 NVMe SSD	Up to 4,750	Up to 10

db.m6id – general-purpose instance classes powered by 3rd generation Intel Xeon Scalable processors and SSD storage

Instance class	vCP	Physic cores	Socke	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6id.metal	128	64	2	—	512	4 x 1900 NVMe SSD	40,000	50
db.m6id.32xlarge	128	—	—	—	512	4 x 1900 NVMe SSD	40,000	50
db.m6id.24xlarge	96	—	—	—	384	4 x 1425 NVMe SSD	30,000	37.5
db.m6id.16xlarge	64	—	—	—	256	2 x 1900 NVMe SSD	20,000	25
db.m6id.12xlarge	48	—	—	—	192	2 x 1425 NVMe SSD	15,000	18.75
db.m6id.8xlarge	32	—	—	—	128	1 x 1900 NVMe SSD	10,000	12.5

Instance class	vCP	Physic cores	Socke	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6id.4xlarge*	16	—	—	—	64	1 x 950 NVMe SSD	Up to 10,000	Up to 12.5
db.m6id.2xlarge*	8	—	—	—	32	1 x 474 NVMe SSD	Up to 10,000	Up to 12.5
db.m6id.xlarge*	4	—	—	—	16	1 x 237 NVMe SSD	Up to 10,000	Up to 12.5
db.m6id.large*	2	—	—	—	8	1 x 118 NVMe SSD	Up to 10,000	Up to 12.5

db.m6idn – general-purpose instance classes with 3rd Generation Intel Xeon Scalable processors, SSD storage, and network optimization

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6idn.32xlarge	128	—	512	4 x 1900 NVMe SSD	80,000	200
db.m6idn.24xlarge	96	—	384	4 x 1425 NVMe SSD	60,000	150
db.m6idn.16xlarge	64	—	256	2 x 1900 NVMe SSD	40,000	100
db.m6idn.12xlarge	48	—	192	2 x 1425 NVMe SSD	30,000	75
db.m6idn.8xlarge	32	—	128	1 x 1900 NVMe SSD	20,000	50

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6idn.4xlarge*	16	—	64	1 x 950 NVMe SSD	Up to 20,000	Up to 50
db.m6idn.2xlarge*	8	—	32	1 x 474 NVMe SSD	Up to 20,000	Up to 40
db.m6idn.xlarge*	4	—	16	1 x 237 NVMe SSD	Up to 20,000	Up to 30
db.m6idn.large*	2	—	8	1 x 118 NVMe SSD	Up to 20,000	Up to 25

db.m6in – general-purpose instance classes powered by 3rd generation Intel Xeon Scalable processors and network optimization

Instance class	vCPU	Proc core	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6in.metal	128	64	2	—	512	EBS-optimized only	100,000	200
db.m6in.32xlarge	128			—	512	EBS-optimized only	80,000	200
db.m6in.24xlarge	96			—	384	EBS-optimized only	60,000	150
db.m6in.16xlarge	64			—	256	EBS-optimized only	40,000	100
db.m6in.12xlarge	48			—	192	EBS-optimized only	30,000	75

Instance class	vCPU	Proc core	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6in.8xlarge	32			—	128	EBS-optimized only	20,000	50
db.m6in.4xlarge*	16			—	64	EBS-optimized only	Up to 20,000	Up to 50
db.m6in.2xlarge*	8			—	32	EBS-optimized only	Up to 20,000	Up to 40
db.m6in.xlarge*	4			—	16	EBS-optimized only	Up to 20,000	Up to 30
db.m6in.large*	2			—	8	EBS-optimized only	Up to 20,000	Up to 25

db.m6i – general-purpose instance classes powered by 3rd generation Intel Xeon Scalable processors

Instance class	vCPU	Proc core	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6i.metal	128	64	2	—	512	EBS-optimized only	40,000	50
db.m6i.32xlarge	128			—	512	EBS-optimized only	40,000	50
db.m6i.24xlarge	96			—	384	EBS-optimized only	30,000	37.5
db.m6i.16xlarge	64			—	256	EBS-optimized only	20,000	25

Instance class	vCPU	Proc core	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m6i.12xlarge	48			—	192	EBS-optimized only	15,000	18.75
db.m6i.8xlarge	32			—	128	EBS-optimized only	10,000	12.5
db.m6i.4xlarge*	16			—	64	EBS-optimized only	Up to 10,000	Up to 12.5
db.m6i.2xlarge*	8			—	32	EBS-optimized only	Up to 10,000	Up to 12.5
db.m6i.xlarge*	4			—	16	EBS-optimized only	Up to 10,000	Up to 12.5
db.m6i.large*	2			—	8	EBS-optimized only	Up to 10,000	Up to 12.5

db.m5d – general-purpose instance classes powered by Intel Xeon Platinum processors and SSD storage

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m5d.24xlarge	96	345	384	4 x 900 NVMe SSD	19,000	25
db.m5d.16xlarge	64	262	256	4 x 600 NVMe SSD	13,600	20
db.m5d.12xlarge	48	173	192	2 x 900 NVMe SSD	9,500	10

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m5d.8xlarge	32	131	128	2 x 600 NVMe SSD	6,800	10
db.m5d.4xlarge	16	61	64	2 x 300 NVMe SSD	4,750	Up to 10
db.m5d.2xlarge*	8	31	32	1 x 300 NVMe SSD	Up to 4,750	Up to 10
db.m5d.xlarge*	4	15	16	1 x 150 NVMe SSD	Up to 4,750	Up to 10
db.m5d.large*	2	10	8	1 x 75 NVMe SSD	Up to 4,750	Up to 10

db.m5 – general-purpose instance classes with Intel Xeon Platinum processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m5.24xlarge	96	345	384	EBS-optimized only	19,000	25
db.m5.16xlarge	64	262	256	EBS-optimized only	13,600	20
db.m5.12xlarge	48	173	192	EBS-optimized only	9,500	10
db.m5.8xlarge	32	131	128	EBS-optimized only	6,800	10

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m5.4xlarge	16	61	64	EBS-optimized only	4,750	Up to 10
db.m5.2xlarge*	8	31	32	EBS-optimized only	Up to 4,750	Up to 10
db.m5.xlarge*	4	15	16	EBS-optimized only	Up to 4,750	Up to 10
db.m5.large*	2	10	8	EBS-optimized only	Up to 4,750	Up to 10

db.m4 – general-purpose instance classes with Intel Xeon Scalable processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m4.16xlarge	64	188	256	EBS-optimized only	10,000	25
db.m4.10xlarge	40	124.5	160	EBS-optimized only	4,000	10
db.m4.4xlarge	16	53.5	64	EBS-optimized only	2,000	High
db.m4.2xlarge	8	25.5	32	EBS-optimized only	1,000	High
db.m4.xlarge	4	13	16	EBS-optimized only	750	High

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m4.large	2	6.5	8	EBS-optimized only	450	Moderate

db.m3 – general-purpose instance classes

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m3.2xlarge	8	26	30	EBS-optimized only	1,000	High
db.m3.xlarge	4	13	15	EBS-optimized only	500	High
db.m3.large	2	6.5	7.5	EBS only	—	Moderate
db.m3.medium	1	3	3.75	EBS only	—	Moderate

* These DB instance classes can support maximum performance for 30 minutes at least once every 24 hours. For more information on baseline performance of the underlying EC2 instance types, see Amazon EBS-optimized instances in the *Amazon EC2 User Guide.*

Hardware specifications for the memory-optimized instance classes

The following tables show the compute, memory, storage, and bandwidth specifications for the memory-optimized instance classes.

db.z1d – memory-optimized instance classes

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.z1d.12xlarge	48	271	384	2 x 900 NVMe SSD	14,000	25
db.z1d.6xlarge	24	134	192	1 x 900 NVMe SSD	7,000	10
db.z1d.3xlarge	12	75	96	1 x 450 NVMe SSD	3,500	Up to 10
db.z1d.2xlarge	8	53	64	1 x 300 NVMe SSD	2,333	Up to 10
db.z1d.xlarge*	4	28	32	1 x 150 NVMe SSD	Up to 2,333	Up to 10
db.z1d.large*	2	15	16	1 x 75 NVMe SSD	Up to 2,333	Up to 10

db.x2g – memory-optimized instance classes with AWS Graviton2 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x2g.16xlarge	64	—	1024	EBS-optimized only	19,000	25
db.x2g.12xlarge	48	—	768	EBS-optimized only	14,250	20
db.x2g.8xlarge	32	—	512	EBS-optimized only	9,500	12

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x2g.4xlarge	16	—	256	EBS-optimized only	4,750	Up to 10
db.x2g.2xlarge	8	—	128	EBS-optimized only	Up to 4,750	Up to 10
db.x2g.xlarge	4	—	64	EBS-optimized only	Up to 4,750	Up to 10
db.x2g.large	2	—	32	EBS-optimized only	Up to 4,750	Up to 10

db.x2idn – memory-optimized instance classes with 3rd generation Intel Xeon Scalable processors

Instance class	vCP	Processor cores	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x2idn.metal	128	64	2	—	2,048	2 x 1900 NVMe SSD	80,000	100
db.x2idn.32xlarge	128	—	—	—	2,048	2 x 1900 NVMe SSD	80,000	100
db.x2idn.24xlarge	96	—	—	—	1,536	2 x 1425 NVMe SSD	60,000	75
db.x2idn.16xlarge	64	—	—	—	1,024	1 x 1900 NVMe SSD	40,000	50

db.x2iedn – memory-optimized instance classes with local NVMe-based SSDs, with 3rd generation Intel Xeon Scalable processors

Instance class	vCPU	Proces cores	Socket	ECU	Memo (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x2iedn .metal	128	64	2	—	4,096	2 x 1900 NVMe SSD	80,000	100
db.x2iedn .32xlarge	128	—	—	—	4,096	2 x 1900 NVMe SSD	80,000	100
db.x2iedn .24xlarge	96	—	—	—	3,072	2 x 1425 NVMe SSD	60,000	75
db.x2iedn .16xlarge	64	—	—	—	2,048	1 x 1900 NVMe SSD	40,000	50
db.x2iedn .8xlarge	32	—	—	—	1,024	1 x 950 NVMe SSD	20,000	25
db.x2iedn .4xlarge	16	—	—	—	512	1 x 475 NVMe SSD	Up to 20,000	Up to 25
db.x2iedn .2xlarge	8	—	—	—	256	1 x 237 NVMe SSD	Up to 20,000	Up to 25
db.x2iedn .xlarge	4	—	—	—	128	1 x 118 NVMe SSD	Up to 20,000	Up to 25

db.x2iezn – memory-optimized instance classes with 2nd generation Intel Xeon Scalable processors

Instance class	vCF	Processo cores	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x2iezn .metal	48	24	2	—	1,536	EBS-optim ized only	19,000	100

Instance class	vCF	Processo cores	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x2iezn .12xlarge	>48	—	—	—	1,536	EBS-optim ized only	19,000	100
db.x2iezn .8xlarge	32	—	—	—	1,024	EBS-optim ized only	12,000	75
db.x2iezn .6xlarge	24	—	—	—	768	EBS-optim ized only	Up to 9,500	50
db.x2iezn .4xlarge	16	—	—	—	512	EBS-optim ized only	Up to 4,750	Up to 25
db.x2iezn .2xlarge	8	—	—	—	256	EBS-optim ized only	Up to 3,170	Up to 25

db.x1e – memory-optimized instance classes

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x1e.32xlarge	128	340	3,904	EBS-optim ized only	14,000	25
db.x1e.16xlarge	64	179	1,952	EBS-optim ized only	7,000	10
db.x1e.8xlarge	32	91	976	EBS-optim ized only	3,500	Up to 10
db.x1e.4xlarge	16	47	488	EBS-optim ized only	1,750	Up to 10

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x1e.2xlarge	8	23	244	EBS-optimized only	1,000	Up to 10
db.x1e.xlarge	4	12	122	EBS-optimized only	500	Up to 10

db.x1 – memory-optimized instance classes

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.x1.32xlarge	128	349	1,952	EBS-optimized only	14,000	25
db.x1.16xlarge	64	174.5	976	EBS-optimized only	7,000	10

db.m8gd – memory-optimized instance classes powered by AWS Graviton4 processors and SSD storage

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m8gd.48xlarge	192	—	768	6 x 1900 NVMe SSD	40,000	50
db.m8gd.24xlarge	96	—	512	3 x 1900 NVMe SSD	30,000	40

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.m8gd.16xlarge	64	—	384	2 x 1900 NVMe SSD	20,000	30
db.m8gd.12xlarge	48	—	256	3 x 950 NVMe SSD	15,000	22.5
db.m8gd.8xlarge	32	—	128	1 x 1900 NVMe SSD	10,000	15
db.m8gd.4xlarge	16	—	64	1 x 950 NVMe SSD	Up to 10,000	Up to 15
db.m8gd.2xlarge	8	—	32	1 x 474 NVMe SSD	Up to 10,000	Up to 15
db.m8gd.xlarge	4	—	16	1 x 237 NVMe SSD	Up to 10,000	Up to 12.5
db.m8gd.large	2	—	8	1 x 118 NVMe SSD	Up to 10,000	Up to 12.5

db.r8gd – memory-optimized instance classes with AWS Graviton4 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r8gd.48xlarge	192	—	1536	6 x 1900 NVMe SSD	40,000	50
db.r8gd.24xlarge	96	—	768	3 x 1900 NVMe SSD	30,000	40

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r8gd.16xlarge	64	—	512	2 x 1900 NVMe SSD	20,000	30
db.r8gd.12xlarge	48	—	384	3 x 950 NVMe SSD	15,000	22.5
db.r8gd.8xlarge	32	—	256	1 x 1900 NVMe SSD	10,000	15
db.r8gd.4xlarge	16	—	128	1 x 950 NVMe SSD	Up to 10,000	Up to 15
db.r8gd.2xlarge	8	—	64	1 x 474 NVMe SSD	Up to 10,000	Up to 15
db.r8gd.xlarge	4	—	32	1 x 237 NVMe SSD	Up to 10,000	Up to 12.5
db.r8gd.large	2	—	16	1 x 118 NVMe SSD	Up to 10,000	Up to 12.5

db.r8g – memory-optimized instance classes with AWS Graviton4 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r8g.48xlarge	192	—	1536	EBS-optimized only	40,000	50
db.r8g.24xlarge	96	—	768	EBS-optimized only	30,000	40

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r8g.16xlarge	64	—	512	EBS-optimized only	20,000	30
db.r8g.12xlarge	48	—	384	EBS-optimized only	15,000	22.5
db.r8g.8xlarge	32	—	256	EBS-optimized only	10,000	15
db.r8g.4xlarge*	16	—	128	EBS-optimized only	Up to 10,000	Up to 15
db.r8g.2xlarge*	8	—	64	EBS-optimized only	Up to 10,000	Up to 15
db.r8g.xlarge*	4	—	32	EBS-optimized only	Up to 10,000	Up to 12.5
db.r8g.large*	2	—	16	EBS-optimized only	Up to 10,000	Up to 12.5

db.r7i – memory-optimized instance classes powered by 4th generation Intel Xeon Scalable processors

Instance class	vCP	Pro core	Soc	ECU	Memory (GiB)	Normalized units	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r7i.metal-48xl	192	96	2	—	1536	192	EBS-optimized only	40,000	50
db.r7i.metal-24xl	96	48	1	—	768	96	EBS-optimized only	30,000	37.5

Instance class	vCP	Pro core	Soc	ECU	Memory (GiB)	Normalized units	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r7i.48xlarge	192	—	—	—	1536	192	EBS-optimized only	40,000	50
db.r7i.24xlarge	96	—	—	—	768	96	EBS-optimized only	30,000	37.5
db.r7i.16xlarge	64	—	—	—	512	64	EBS-optimized only	20,000	25
db.r7i.12xlarge	48	—	—	—	384	48	EBS-optimized only	15,000	18.75
db.r7i.8xlarge.tpc2.mem3x	32	—	—	—	768	96	EBS-optimized only	30,000	12.5
db.r7i.8xlarge.tpc2.mem2x	32	—	—	—	512	64	EBS-optimized only	20,000	12.5
db.r7i.8xlarge	32	—	—	—	256	32	EBS-optimized only	10,000	12.5
db.r7i.6xlarge.tpc2.mem4x	24	—	—	—	768	96	EBS-optimized only	30,000	Up to 12.5
db.r7i.6xlarge.tpc2.mem2x	24	—	—	—	384	48	EBS-optimized only	15,000	Up to 12.5
db.r7i.4xlarge.tpc2.mem4x	16	—	—	—	512	64	EBS-optimized only	20,000	Up to 12.5

Instance class	vCP	Pro core	Soc	ECU	Memory (GiB)	Normalized units	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r7i.4xlarge.tpc2.mem3x	16	—	—	—	384	48	EBS-optimized only	15,000	Up to 12.5
db.r7i.4xlarge.tpc2.mem2x	16	—	—	—	256	32	EBS-optimized only	10,000	Up to 12.5
db.r7i.4xlarge	16	—	—	—	128	16	EBS-optimized only	Up to 10,000	Up to 12.5
db.r7i.3xlarge.tpc2.mem4x	12	—	—	—	384	48	EBS-optimized only	15,000	Up to 12.5
db.r7i.2xlarge.tpc2.mem8x	8	—	—	—	512	64	EBS-optimized only	20,000	Up to 12.5
db.r7i.2xlarge.tpc2.mem4x	8	—	—	—	256	32	EBS-optimized only	10,000	Up to 12.5
db.r7i.2xlarge	8	—	—	—	64	8	EBS-optimized only	Up to 10,000	Up to 12.5
db.r7i.xlarge.tpc2.mem4x	4	—	—	—	128	16	EBS-optimized only	Up to 10,000	Up to 12.5
db.r7i.xlarge.tpc2.mem2x	4	—	—	—	64	8	EBS-optimized only	Up to 10,000	Up to 12.5

Instance class	vCP	Pro core	Soc	ECU	Memory (GiB)	Norma d units	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r7i.xlarge	4	—	—	—	32	4	EBS-optim ized only	Up to 10,000	Up to 12.5
db.r7i.large	2	—	—	—	16	2	EBS-optim ized only	Up to 10,000	Up to 12.5

db.r7g – memory-optimized instance classes with AWS Graviton3 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r7g.16xlarge	64	—	512	EBS-optim ized only	20,000	30
db.r7g.12xlarge	48	—	384	EBS-optim ized only	15,000	22.5
db.r7g.8xlarge	32	—	256	EBS-optim ized only	10,000	15
db.r7g.4xlarge	16	—	128	EBS-optim ized only	Up to 10,000	Up to 15
db.r7g.2xlarge*	8	—	64	EBS-optim ized only	Up to 10,000	Up to 15
db.r7g.xlarge*	4	—	32	EBS-optim ized only	Up to 10,000	Up to 12.5
db.r7g.large*	2	—	16	EBS-optim ized only	Up to 10,000	Up to 12.5

db.r6g – memory-optimized instance classes with AWS Graviton2 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6g.16xlarge	64	—	512	EBS-optimized only	19,000	25
db.r6g.12xlarge	48	—	384	EBS-optimized only	13,500	20
db.r6g.8xlarge	32	—	256	EBS-optimized only	9,000	12
db.r6g.4xlarge	16	—	128	EBS-optimized only	4,750	Up to 10
db.r6g.2xlarge*	8	—	64	EBS-optimized only	Up to 4,750	Up to 10
db.r6g.xlarge*	4	—	32	EBS-optimized only	Up to 4,750	Up to 10
db.r6g.large*	2	—	16	EBS-optimized only	Up to 4,750	Up to 10

db.r6gd – memory-optimized instance classes with AWS Graviton2 processors and SSD storage

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6gd.16xlarge	64	—	512	2 x 1900 NVMe SSD	19,000	25
db.r6gd.12xlarge	48	—	384	2 x 1425 NVMe SSD	13,500	20

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6gd.8xlarge	32	—	256	1 x 1900 NVMe SSD	9,000	12
db.r6gd.4xlarge	16	—	128	1 x 950 NVMe SSD	4,750	Up to 10
db.r6gd.2xlarge	8	—	64	1 x 474 NVMe SSD	Up to 4,750	Up to 10
db.r6gd.xlarge	4	—	32	1 x 237 NVMe SSD	Up to 4,750	Up to 10
db.r6gd.large	2	—	16	1 x 118 NVMe SSD	Up to 4,750	Up to 10

db.r6id – memory-optimized instance classes with 3rd generation Intel Xeon Scalable processors and SSD storage

Instance class	vCPU	Processor cores	Socket	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6id.metal	128	64	2	—	1,024	4 x 1900 NVMe SSD	40,000	50
db.r6id.32xlarge	128	—	—	—	1,024	4x1900 NVMe SSD	40,000	50
db.r6id.24xlarge	96	—	—	—	768	4x1425 NVMe SSD	30,000	37.5
db.r6id.16xlarge	64	—	—	—	512	2x1900 NVMe SSD	20,000	25

Instance class	vCP	Processo cores	Soc	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6id.12xlarge	48	—	—	—	384	2x1425 NVMe SSD	15,000	18.75
db.r6id.8xlarge	32	—	—	—	256	1x1900 NVMe SSD	10,000	12.5
db.r6id.4xlarge*	16	—	—	—	128	1x950 NVMe SSD	Up to 10,000	Up to 12.5
db.r6id.2xlarge*	8	—	—	—	64	1x474 NVMe SSD	Up to 10,000	Up to 12.5
db.r6id.xlarge*	4	—	—	—	32	1x237 NVMe SSD	Up to 10,000	Up to 12.5
db.r6id.large*	2	—	—	—	16	1x118 NVMe SSD	Up to 10,000	Up to 12.5

db.r6idn – memory-optimized instance classes with 3rd generation Intel Xeon Scalable processors, SSD storage, and network optimization

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6idn.32xlarge	128	—	1,024	4x1900 NVMe SSD	80,000	200
db.r6idn.24xlarge	96	—	768	4x1425 NVMe SSD	60,000	150
db.r6idn.16xlarge	64	—	512	2x1900 NVMe SSD	40,000	100

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6idn.12xlarge	48	—	384	2x1425 NVMe SSD	30,000	75
db.r6idn.8xlarge	32	—	256	1x1900 NVMe SSD	20,000	50
db.r6idn.4xlarge*	16	—	128	1x950 NVMe SSD	Up to 20,000	Up to 50
db.r6idn.2xlarge*	8	—	64	1x474 NVMe SSD	Up to 20,000	Up to 40
db.r6idn.xlarge*	4	—	32	1x237 NVMe SSD	Up to 20,000	Up to 30
db.r6idn.large*	2	—	16	1x118 NVMe SSD	Up to 20,000	Up to 25

db.r6in – memory-optimized instance classes with 3rd generation Intel Xeon Scalable processors and network optimization

Instance class	vCPU	Proc core	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6in.metal	128	64	2	—	1,024	EBS-optimized only	100,000	200
db.r6in.32xlarge	128	—	—	—	1,024	EBS-optimized only	80,000	200
db.r6in.24xlarge	96	—	—	—	768	EBS-optimized only	60,000	150

Instance class	vCPU	Proc core	Sock	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6in.16xlarge	64	—	—	—	512	EBS-optimized only	40,000	100
db.r6in.12xlarge	48	—	—	—	384	EBS-optimized only	30,000	75
db.r6in.8xlarge	32	—	—	—	256	EBS-optimized only	20,000	50
db.r6in.4xlarge*	16	—	—	—	128	EBS-optimized only	Up to 20,000	Up to 50
db.r6in.2xlarge*	8	—	—	—	64	EBS-optimized only	Up to 20,000	Up to 40
db.r6in.xlarge*	4	—	—	—	32	EBS-optimized only	Up to 20,000	Up to 30
db.r6in.large*	2	—	—	—	16	EBS-optimized only	Up to 20,000	Up to 25

db.r6i – Oracle memory-optimized instance classes preconfigured for high memory, storage, and I/O

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6i.8xlarge.tpc2.mem4x	32	—	1024	EBS-optimized only	40,000	50
db.r6i.8xlarge.tpc2.mem3x	32	—	768	EBS-optimized only	30,000	37.5

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6i.6xlarge.tpc2.mem4x	24	—	768	EBS-optimized only	30,000	37.5
db.r6i.4xlarge.tpc2.mem4x	16	—	512	EBS-optimized only	20,000	25
db.r6i.4xlarge.tpc2.mem3x	16	—	384	EBS-optimized only	15,000	18.75
db.r6i.4xlarge.tpc2.mem2x	16	—	256	EBS-optimized only	10,000	12.5
db.r6i.2xlarge.tpc2.mem8x	8	—	512	EBS-optimized only	20,000	12.5
db.r6i.2xlarge.tpc2.mem4x	8	—	256	EBS-optimized only	10,000	12.5
db.r6i.2xlarge.tpc1.mem2x	8	—	128	EBS-optimized only	Up to 10,000	12.5
db.r6i.xlarge.tpc2.mem4x	4	—	128	EBS-optimized only	Up to 10,000	12.5
db.r6i.xlarge.tpc2.mem2x	4	—	64	EBS-optimized only	Up to 10,000	12.5
db.r6i.large.tpc1.mem2x	2	—	32	EBS-optimized only	Up to 10,000	12.5

db.r6i – memory-optimized instance classes with 3rd Generation Intel Xeon Scalable processors

Instance class	vCP	Pro core	Soc	ECU	Memory (GiB)	Norma d units	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r6i.metal	128	64	2	—	1,024	256	EBS-optim ized only	40,000	50
db.r6i.32xlarge	128	—	—	—	1,024	256	EBS-optim ized only	40,000	50
db.r6i.24xlarge	96	—	—	—	768	192	EBS-optim ized only	30,000	37.5
db.r6i.16xlarge	64	—	—	—	512	128	EBS-optim ized only	20,000	25
db.r6i.12xlarge	48	—	—	—	384	96	EBS-optim ized only	15,000	18.75
db.r6i.8xlarge	32	—	—	—	256	64	EBS-optim ized only	10,000	12.5
db.r6i.4xlarge*	16	—	—	—	128	32	EBS-optim ized only	Up to 10,000	Up to 12.5
db.r6i.2xlarge*	8	—	—	—	64	16	EBS-optim ized only	Up to 10,000	Up to 12.5
db.r6i.xlarge*	4	—	—	—	32	8	EBS-optim ized only	Up to 10,000	Up to 12.5
db.r6i.large*	2	—	—	—	16	4	EBS-optim ized only	Up to 10,000	Up to 12.5

db.r5d – memory-optimized instance classes with Intel Xeon Platinum processors and SSD storage

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r5d.24xlarge	96	347	768	4 x 900 NVMe SSD	19,000	25
db.r5d.16xlarge	64	264	512	4 x 600 NVMe SSD	13,600	20
db.r5d.12xlarge	48	173	384	2 x 900 NVMe SSD	9,500	10
db.r5d.8xlarge	32	132	256	2 x 600 NVMe SSD	6,800	10
db.r5d.4xlarge	16	71	128	2 x 300 NVMe SSD	4,750	Up to 10
db.r5d.2xlarge*	8	38	64	1 x 300 NVMe SSD	Up to 4,750	Up to 10
db.r5d.xlarge*	4	19	32	1 x 150 NVMe SSD	Up to 4,750	Up to 10
db.r5d.large*	2	10	16	1 x 75 NVMe SSD	Up to 4,750	Up to 10

db.r5b – Oracle memory-optimized instance classes preconfigured for high memory, storage, and I/O

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r5b.8xlarge.tpc2.mem3x	32	—	768	EBS-optimized only	60,000	25

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r5b.6xlarge.tpc2.mem4x	24	—	768	EBS-optimized only	60,000	25
db.r5b.4xlarge.tpc2.mem4x	16	—	512	EBS-optimized only	40,000	20
db.r5b.4xlarge.tpc2.mem3x	16	—	384	EBS-optimized only	30,000	10
db.r5b.4xlarge.tpc2.mem2x	16	—	256	EBS-optimized only	20,000	10
db.r5b.2xlarge.tpc2.mem8x	8	—	512	EBS-optimized only	40,000	20
db.r5b.2xlarge.tpc2.mem4x	8	—	256	EBS-optimized only	20,000	10
db.r5b.2xlarge.tpc1.mem2x	8	—	128	EBS-optimized only	10,000	Up to 10
db.r5b.xlarge.tpc2.mem4x	4	—	128	EBS-optimized only	10,000	Up to 10
db.r5b.xlarge.tpc2.mem2x	4	—	64	EBS-optimized only	Up to 10,000	Up to 10
db.r5b.large.tpc1.mem2x	2	—	32	EBS-optimized only	Up to 10,000	Up to 10

db.r5b – memory-optimized instance classes with Intel Xeon Platinum processors and EBS optimization

Instance class	vCPU	ECU	Memory (GiB)	Normalized units	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r5b.24xlarge	96	347	768	192	EBS-optimized only	60,000	25
db.r5b.16xlarge	64	264	512	128	EBS-optimized only	40,000	20
db.r5b.12xlarge	48	173	384	96	EBS-optimized only	30,000	10
db.r5b.8xlarge	32	132	256	64	EBS-optimized only	20,000	10
db.r5b.4xlarge	16	71	128	32	EBS-optimized only	10,000	Up to 10
db.r5b.2xlarge*	8	38	64	16	EBS-optimized only	Up to 10,000	Up to 10
db.r5b.xlarge*	4	19	32	8	EBS-optimized only	Up to 10,000	Up to 10
db.r5b.large*	2	10	16	4	EBS-optimized only	Up to 10,000	Up to 10

db.r5 – Oracle memory-optimized instance classes preconfigured for high memory, storage, and I/O

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r5.12xlarge.tpc2.mem2x	48	—	768	EBS-optimized only	19,000	25

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r5.8xlarge.tpc2.mem3x	32	—	768	EBS-optimized only	19,000	25
db.r5.6xlarge.tpc2.mem4x	24	—	768	EBS-optimized only	19,000	25
db.r5.4xlarge.tpc2.mem4x	16	—	512	EBS-optimized only	13,600	20
db.r5.4xlarge.tpc2.mem3x	16	—	384	EBS-optimized only	9,500	10
db.r5.4xlarge.tpc2.mem2x	16	—	256	EBS-optimized only	6,800	10
db.r5.2xlarge.tpc2.mem8x	8	—	512	EBS-optimized only	13,600	20
db.r5.2xlarge.tpc2.mem4x	8	—	256	EBS-optimized only	6,800	10
db.r5.2xlarge.tpc1.mem2x	8	—	128	EBS-optimized only	4,750	Up to 10
db.r5.xlarge.tpc2.mem4x	4	—	128	EBS-optimized only	4,750	Up to 10
db.r5.xlarge.tpc2.mem2x	4	—	64	EBS-optimized only	Up to 4,750	Up to 10
db.r5.large.tpc1.mem2x	2	—	32	EBS-optimized only	Up to 4,750	Up to 10

db.r5 – memory-optimized instance classes

Instance class	vCPU	ECU	Memory (GiB)	Normalized units	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r5.24xlarge	96	347	768	192	EBS-optimized only	19,000	25
db.r5.16xlarge	64	264	512	128	EBS-optimized only	13,600	20
db.r5.12xlarge	48	173	384	96	EBS-optimized only	9,500	12
db.r5.8xlarge	32	132	256	64	EBS-optimized only	6,800	10
db.r5.4xlarge	16	71	128	32	EBS-optimized only	4,750	Up to 10
db.r5.2xlarge*	8	38	64	16	EBS-optimized only	Up to 4,750	Up to 10
db.r5.xlarge*	4	19	32	8	EBS-optimized only	Up to 4,750	Up to 10
db.r5.large*	2	10	16	4	EBS-optimized only	Up to 4,750	Up to 10

db.r4 – memory-optimized instance classes with Intel Xeon Scalable processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r4.16xlarge	64	195	488	EBS-optimized only	14,000	25

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r4.8xlarge	32	99	244	EBS-optimized only	7,000	10
db.r4.4xlarge	16	53	122	EBS-optimized only	3,500	Up to 10
db.r4.2xlarge	8	27	61	EBS-optimized only	1,700	Up to 10
db.r4.xlarge	4	13.5	30.5	EBS-optimized only	850	Up to 10
db.r4.large	2	7	15.25	EBS-optimized only	425	Up to 10

db.r3 – memory-optimized instance classes

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.r3.8xlarge**	32	104	244	EBS only	—	10
db.r3.4xlarge	16	52	122	EBS-optimized only	2,000	High
db.r3.2xlarge	8	26	61	EBS-optimized only	1,000	High
db.r3.xlarge	4	13	30.5	EBS-optimized only	500	Moderate
db.r3.large	2	6.5	15.25	EBS-optimized only	—	Moderate

* These DB instance classes can support maximum performance for 30 minutes at least once every 24 hours. For more information on baseline performance of the underlying EC2 instance types, see Amazon EBS-optimized instances in the *Amazon EC2 User Guide.*

** The r3.8xlarge DB instance class doesn't have dedicated EBS bandwidth and therefore doesn't offer EBS optimization. For this instance class, network traffic and Amazon EBS traffic share the same 10-gigabit network interface.

Hardware specifications for the compute-optimized instance classes

The following tables show the compute, memory, storage, and bandwidth specifications for the compute-optimized instance classes.

db.c6gd – compute-optimized instance classes (for Multi-AZ DB cluster deployments only)

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.c6gd.16xlarge	64	—	128	2 x 1900 NVMe SSD	19,000	25
db.c6gd.12xlarge	48	—	96	2 x 1425 NVMe SSD	13,500	20
db.c6gd.8xlarge	32	—	64	1 x 1900 NVMe SSD	9,000	12
db.c6gd.4xlarge	16	—	32	1 x 950 NVMe SSD	4,750	Up to 10
db.c6gd.2xlarge	8	—	16	1 x 474 NVMe SSD	Up to 4,750	Up to 10
db.c6gd.xlarge	4	—	8	1 x 237 NVMe SSD	Up to 4,750	Up to 10
db.c6gd.large	2	—	4	1 x 118 NVMe SSD	Up to 4,750	Up to 10

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.c6gd.medium	1	—	2	1 x 59 NVMe SSD	Up to 4,750	Up to 10

Hardware specifications for the burstable-performance instance classes

The following tables show the compute, memory, storage, and bandwidth specifications for the burstable-performance instance classes.

db.t4g – burstable-performance instance classes powered by AWS Graviton2 processors

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.t4g.2xlarge*	8	—	32	EBS-optimized only	Up to 2,780	Up to 5
db.t4g.xlarge*	4	—	16	EBS-optimized only	Up to 2,780	Up to 5
db.t4g.large*	2	—	8	EBS-optimized only	Up to 2,780	Up to 5
db.t4g.medium*	2	—	4	EBS-optimized only	Up to 2,085	Up to 5
db.t4g.small*	2	—	2	EBS-optimized only	Up to 2,085	Up to 5
db.t4g.micro*	2	—	1	EBS-optimized only	Up to 2,085	Up to 5

db.t3 – burstable-performance instance classes

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.t3.2xlarge*	8	Variab	32	EBS-optimized only	Up to 2,048	Up to 5
db.t3.xlarge*	4	Variab	16	EBS-optimized only	Up to 2,048	Up to 5
db.t3.large*	2	Variab	8	EBS-optimized only	Up to 2,048	Up to 5
db.t3.medium*	2	Variab	4	EBS-optimized only	Up to 1,536	Up to 5
db.t3.small*	2	Variab	2	EBS-optimized only	Up to 1,536	Up to 5
db.t3.micro*	2	Variab	1	EBS-optimized only	Up to 1,536	Up to 5

db.t2 – burstable-performance instance classes

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.t2.2xlarge	8	Variab	32	EBS only	—	Moderate
db.t2.xlarge	4	Variab	16	EBS only	—	Moderate
db.t2.large	2	Variab	8	EBS only	—	Moderate
db.t2.medium	2	Variab	4	EBS only	—	Moderate
db.t2.small	1	Variab	2	EBS only	—	Low

Instance class	vCPU	ECU	Memory (GiB)	Instance storage (GiB)	Max. EBS bandwidth (Mbps)	Network bandwidth (Gbps)
db.t2.micro	1	Variab	1	EBS only	—	Low

* These DB instance classes can support maximum performance for 30 minutes at least once every 24 hours. For more information on baseline performance of the underlying EC2 instance types, see Amazon EBS-optimized instances in the *Amazon EC2 User Guide.*

Amazon RDS DB instance storage

DB instances for Amazon RDS for Db2, MariaDB, MySQL, PostgreSQL, Oracle, and Microsoft SQL Server use Amazon Elastic Block Store (Amazon EBS) volumes for database and log storage.

In some cases, your database workload might not be able to achieve 100 percent of the IOPS that you have provisioned. For more information, see Factors that affect database performance.

For more information about instance storage pricing, see Amazon RDS pricing.

Topics

- Amazon RDS storage types
- Provisioned IOPS SSD storage
- General Purpose SSD storage
- Performance characteristics of solid-state drive (SSD) storage types
- Magnetic storage (legacy, not recommended)
- Additional storage volumes
- Dedicated log volume (DLV)
- Monitoring database performance
- Factors that affect database performance

Amazon RDS storage types

Amazon RDS provides three storage types: Provisioned IOPS SSD (also known as io1 and io2 Block Express), General Purpose SSD (also known as gp2 and gp3), and magnetic (also known as standard). They differ in performance characteristics and price, which means that you can tailor your storage performance and cost to the needs of your database workload. You can create Db2, MySQL, MariaDB, and PostgreSQL RDS DB instances with up to 64 tebibytes (TiB) of storage. You can create Oracle and SQL Server instances with up to 256 TiB of storage with additional storage volumes. For more information, see Additional storage volumes. RDS for Db2 doesn't support the gp2 and magnetic storage types.

The following list briefly describes the three storage types:

- **Provisioned IOPS SSD** – Provisioned IOPS storage is designed to meet the needs of I/O-intensive workloads, particularly database workloads, that require low I/O latency and consistent I/O throughput. Provisioned IOPS storage is best suited for production environments.

For more information about Provisioned IOPS storage, including the storage size ranges, see Provisioned IOPS SSD storage.

- **General Purpose SSD** – General Purpose SSD volumes offer cost-effective storage that is ideal for a broad range of workloads running on medium-sized DB instances. General Purpose storage is best suited for development and testing environments.

  For more information about General Purpose SSD storage, including the storage size ranges, see General Purpose SSD storage.

- **Magnetic** – Amazon RDS also supports magnetic storage for backward compatibility. We recommend that you use General Purpose SSD or Provisioned IOPS SSD for any new storage needs. The maximum amount of storage allowed for DB instances on magnetic storage is 3 TiB. For more information, see Magnetic storage (legacy, not recommended).

Provisioned IOPS SSD storage

For a production application that requires fast and consistent I/O performance, we recommend Provisioned IOPS storage. Provisioned IOPS storage is a storage type that delivers predictable performance, and consistently low latency. Provisioned IOPS storage is optimized for online transaction processing (OLTP) workloads that require consistent performance. Provisioned IOPS helps performance tuning of these workloads.

Amazon RDS offers two types of Provisioned IOPS SSD storage: io2 and io1. When you create a DB instance, you specify the IOPS rate and the size of the volume. Amazon RDS provides that IOPS rate for the DB instance until you change it.

Topics

- io2 Block Express storage (recommended)
- io1 storage (previous generation)
- Combining Provisioned IOPS storage with Multi-AZ deployments or read replicas
- Provisioned IOPS storage costs
- Getting the best performance from Amazon RDS Provisioned IOPS storage

io2 Block Express storage (recommended)

For I/O-intensive and latency-sensitive workloads, we recommend that you use Provisioned IOPS SSD io2 Block Express storage to achieve up to 256,000 I/O operations per second (IOPS). The throughput of io2 Block Express volumes varies based on the amount of IOPS provisioned per volume and on the size of the I/O operations being run.

All RDS io2 volumes based on the AWS Nitro System are io2 Block Express volumes and provide sub-millisecond average latency. DB instances not based on the AWS Nitro System are io2 volumes.

The following table shows the range of Provisioned IOPS and maximum throughput per storage volume for each database engine and storage size range. In Amazon RDS for Oracle and SQL Server, you can attach up to three additional storage volumes in addition to the primary storage volumes to provision up to 256 TiB storage size in total. With additional storage volumes, you can have higher provisioned IOPS and maximum throughput for your DB instance. However, your DB instance might not be able to fully utilize the provisioned IOPS and maximum throughput if your instance class has lower limits than the values you've provisioned for your storage volumes. For more information, see Factors that affect database performance.

Database engine	Range of storage size	Range of Provisioned IOPS	Maximum throughput
Db2, MariaDB, MySQL, and PostgreSQL	100–65,536 GiB	1,000–256,000 IOPS	16,000 MiB/s
Oracle	100–199 GiB	1,000–199,000 IOPS	4,000 MiB/s
Oracle	200–65,536 GiB	1,000–256,000 IOPS	16,000 MiB/s
SQL Server	20–65,536 GiB	1,000–256,000 IOPS	4,000 MiB/s

The IOPS and storage size ranges have the following constraints:

- The ratio of IOPS to allocated storage (in GiB) must be from 0.5–1,000. For DB instances not based on the AWS Nitro System, the ratio must be from 0.5–500.
- Maximum IOPS can be provisioned with volumes 256 GiB and larger (1,000 IOPS × 256 GiB = 256,000 IOPS). For DB instances not based on the AWS Nitro System, maximum IOPS are achieved at 512 GiB (500 IOPS x 512 GiB = 256,000 IOPS).

- Throughput scales proportionally up to 0.256 MiB/s per provisioned IOPS. Maximum throughput of 4,000 MiB/s can be achieved at 256,000 IOPS with a 16-KiB I/O size and 16,000 IOPS or higher with a 256-KiB I/O size. For DB instances not based on the AWS Nitro System, maximum throughput of 2,000 MiB/s can be achieved at 128,000 IOPS with a 16-KiB I/O size.
- If you're using storage autoscaling, the same ratios between IOPS and maximum storage threshold (in GiB) also apply. For more information on storage autoscaling, see Managing capacity automatically with Amazon RDS storage autoscaling.

Amazon RDS io2 Block Express volumes are available in all commercial AWS Regions and AWS GovCloud (US) Regions. These volumes aren't available in the China Regions.

io1 storage (previous generation)

For I/O-intensive workloads, you can use Provisioned IOPS SSD io1 storage and achieve up to 256,000 I/O operations per second (IOPS). The throughput of io1 volumes varies based on the amount of IOPS provisioned per volume and on the size of the I/O operations being run. We recommend using io2 Block Express storage where it's available.

The following table shows the range of Provisioned IOPS and maximum throughput for each database engine and storage size range.

Database engine	Range of storage size	Range of Provisioned IOPS	Maximum throughput
Db2, MariaDB, MySQL, and PostgreSQL	100–399 GiB	1,000–19,950 IOPS	500 MiB/s
Db2, MariaDB, MySQL, and PostgreSQL	400–65,536 GiB	1,000–256,000 IOPS	4,000 MiB/s
Oracle	100–199 GiB	1,000–9,950 IOPS	500 MiB/s
Oracle	200–65,536 GiB	1,000–256,000 IOPS[1]	4,000 MiB/s
SQL Server	20–16,384 GiB	1,000–64,000 IOPS[2]	1,000 MiB/s

Note

[1] For Oracle, you can provision the maximum 256,000 IOPS only on the r5b instance type.
[2] For SQL Server, the maximum 64,000 IOPS is guaranteed only on Nitro-based instances that are on the m5*, m6i, r5*, r6i, and z1d instance types. Other instance types guarantee performance up to 32,000 IOPS.

The IOPS and storage size ranges have the following constraints:

- The ratio of IOPS to allocated storage (in GiB) must be from 1–50 on RDS for SQL Server, and 0.5–50 on other RDS DB engines.
- If you're using storage autoscaling, the same ratios between IOPS and maximum storage threshold (in GiB) also apply.

  For more information on storage autoscaling, see Managing capacity automatically with Amazon RDS storage autoscaling.

Combining Provisioned IOPS storage with Multi-AZ deployments or read replicas

For production OLTP use cases, we recommend that you use Multi-AZ deployments for enhanced fault tolerance with Provisioned IOPS storage for fast and predictable performance.

You can also use Provisioned IOPS storage with read replicas for MySQL, MariaDB or PostgreSQL. The type of storage for a read replica is independent of that on the primary DB instance. For example, you might use General Purpose SSD for read replicas with a primary DB instance that uses Provisioned IOPS SSD storage to reduce costs. However, your read replica's performance in this case might differ from that of a configuration where both the primary DB instance and the read replicas use Provisioned IOPS storage.

Provisioned IOPS storage costs

With Provisioned IOPS storage, you are charged for the provisioned resources whether or not you use them in a given month.

For more information about pricing, see Amazon RDS pricing.

Getting the best performance from Amazon RDS Provisioned IOPS storage

If your workload is I/O constrained, using Provisioned IOPS storage can increase the number of I/O requests that the system can process concurrently. Increased concurrency allows for decreased latency because I/O requests spend less time in a queue. Decreased latency allows for faster database commits, which improves response time and allows for higher database throughput.

Provisioned IOPS storage provides a way to reserve I/O capacity by specifying IOPS. However, as with any other system capacity attribute, its maximum throughput under load is constrained by the resource that is consumed first. That resource might be network bandwidth, CPU, memory, or database internal resources.

General Purpose SSD storage

General Purpose storage offers cost-effective storage that is acceptable for most database workloads that aren't latency or performance sensitive.

> **Note**
>
> DB instances that use General Purpose storage can experience much longer latency than instances that use Provisioned IOPS storage. If you need a DB instance with minimum latency after these operations, we recommend using Provisioned IOPS SSD storage.

Amazon RDS offers two types of General Purpose storage: gp3 storage (recommended) and gp2 storage (previous generation).

gp3 storage (recommended)

By using General Purpose gp3 storage volumes, you can customize storage performance independently of storage capacity. *Storage performance* is the combination of I/O operations per second (IOPS) and how fast the storage volume can perform reads and writes (storage throughput). On gp3 storage volumes, Amazon RDS provides a baseline storage performance of 3000 IOPS and 125 MiB/s.

For every RDS DB engine except RDS for SQL Server, when the storage size for gp3 volumes reaches a certain threshold, the baseline storage performance increases. This is because of *volume striping*, where the storage uses four volumes instead of one. RDS for SQL Server doesn't support

volume striping, and therefore doesn't have a threshold value. For striped volumes, Amazon RDS provides a baseline storage performance of 12,000 IOPS and 500 MiB/s.

Storage performance for gp3 volumes on Amazon RDS DB engines, including the threshold per storage volume, is shown in the following table. In RDS for Oracle and SQL Server, you can attach up to three additional storage volumes in addition to the primary storage volume. You can provision up to 256 TiB storage size in total with three gp3 additional storage volumes in RDS for Oracle. You can provision up to 64 TiB storage size in total with three gp3 additional storage volumes in RDS for SQL Server because each gp3 storage volume can be sized up to 16 TiB. With additional storage volumes, you can have higher provisioned IOPS and maximum throughput for your DB instance. However, your DB instance might not be able to fully utilize the provisioned IOPS and maximum throughput if your instance class has lower limits than the values you've provisioned for your storage volumes. For more information, see Factors that affect database performance.

DB engine	Storage size	Baseline storage performance	Range of Provisioned IOPS	Range of provisioned storage throughput
Db2, MariaDB, MySQL, and PostgreSQL	20–399 GiB	3,000 IOPS/125 MiB/s	N/A	N/A
Db2, MariaDB, MySQL, and PostgreSQL	400–65,536 GiB	12,000 IOPS/500 MiB/s	12,000–64,000 IOPS	500–4,000 MiB/s
Oracle	20–199 GiB	3,000 IOPS/125 MiB/s	N/A	N/A
Oracle	200–65,536 GiB	12,000 IOPS/500 MiB/s	12,000–64,000 IOPS	500–4,000 MiB/s
SQL Server	20–16,384 GiB	3,000 IOPS/125 MiB/s	3,000–16,000 IOPS	125–1,000 MiB/s

For every DB engine except RDS for SQL Server, you can provision additional IOPS and storage throughput when storage size is at or above the threshold value. For RDS for SQL Server, you

can provision additional IOPS and storage throughput for any available storage size. For all DB engines, you pay for only the additional provisioned storage performance. For more information, see Amazon RDS pricing.

Although the added Provisioned IOPS and storage throughput aren't dependent on the storage size, they are related to each other. When you raise the IOPS above 32,000 for MariaDB and MySQL, the storage throughput value automatically increases from 500 MiBps. For example, when you set the IOPS to 40,000 on RDS for MySQL, the storage throughput must be at least 625 MiBps. The automatic increase doesn't happen for Db2, Oracle, PostgreSQL, and SQL Server DB instances.

For Multi-AZ DB clusters, Amazon RDS automatically sets the throughput value based on the IOPS that you provision. You can't modify the throughput value.

Storage performance values for gp3 volumes on RDS have the following constraints:

- The maximum ratio of storage throughput to IOPS is 0.25 for all supported DB engines.
- The minimum ratio of IOPS to allocated storage (in GiB) is 0.5 on RDS for SQL Server. There is no minimum ratio for the other supported DB engines.
- The maximum ratio of IOPS to allocated storage is 500 for all supported DB engines.
- If you're using storage autoscaling, the same ratios between IOPS and maximum storage threshold (in GiB) also apply.

  For more information on storage autoscaling, see Managing capacity automatically with Amazon RDS storage autoscaling.

gp2 storage (previous generation)

When your applications don't need high storage performance, you can use General Purpose SSD gp2 storage. Baseline I/O performance for gp2 storage is 3 IOPS for each GiB, with a minimum of 100 IOPS. This relationship means that larger volumes have better performance. For example, baseline performance for one 100 GiB volume is 300 IOPS. Baseline performance for one 1,000 GiB volume is 3,000 IOPS.

Individual gp2 volumes below 1,000 GiB in size also have the ability to burst to 3,000 IOPS for extended periods of time. Volume I/O credit balance determines burst performance. For a more detailed description of how baseline performance and I/O credit balance affect performance, see the post Understanding burst vs. baseline performance with Amazon RDS and gp2 on the AWS Database Blog.

Many workloads never deplete the burst balance. However, some workloads can exhaust the 3,000 IOPS burst storage credit balance, so plan your storage capacity to meet the needs of your workloads.

For gp2 volumes larger than 4,000 GiB, the baseline performance is greater than the burst performance. For such volumes, burst is irrelevant because the baseline performance is better than the 3,000 IOPS burst performance. However, for DB instances of certain engines and sizes, storage is *striped* across four volumes providing four times the baseline throughput, and four times the burst IOPS of a single volume.

Storage performance for gp2 volumes of various storage sizes on Amazon RDS DB engines is shown in the following table.

DB engine	RDS storage size	Range of baseline IOPS	Range of baseline throughput	Burst IOPS
MariaDB, MySQL, and PostgreSQL	5–399 GiB	100-1197 IOPS	128-250 MiB/s	3,000
MariaDB, MySQL, and PostgreSQL	400–1,335 GiB	1,200-4,005 IOPS	512-1,000 MiB/s	12,000
MariaDB, MySQL, and PostgreSQL	1,336–3,999 GiB	4008-11,997 IOPS	1,000 MiB/s	12,000
MariaDB, MySQL, and PostgreSQL	4,000–65,536 GiB	12,000-64,000 IOPS	1,000 MiB/s	N/A[1]
Oracle	20–199 GiB	100-597 IOPS	128-250 MiB/s	3,000
Oracle	200–1,335 GiB	600-4,005 IOPS	512-1,000 MiB/s	12,000
Oracle	1,336–3,999 GiB	4008-11,997 IOPS	1,000 MiB/s	12,000

DB engine	RDS storage size	Range of baseline IOPS	Range of baseline throughput	Burst IOPS
Oracle	4,000–65,536 GiB	12,000-64,000 IOPS	1,000 MiB/s	N/A[1]
SQL Server	20–333 GiB	100-999 IOPS	128-250 MiB/s	3,000
SQL Server	334–999 GiB	1,002-2,997 IOPS	250 MiB/s	3,000
SQL Server	1,000–16,384 GiB	3,000-16,000 IOPS	250 MiB/s	N/A[1]

[1] The baseline performance of the volume exceeds the maximum burst performance.

Performance characteristics of solid-state drive (SSD) storage types

The following table describes use cases and per-volume performance characteristics for the SSD storage volumes used by Amazon RDS.

Characteristic	Provisioned IOPS (io2 Block Express)	Provisioned IOPS (io1)	General Purpose (gp3)	General Purpose (gp2)
Description	Highest performance within the RDS storage portfolio (IOPS, throughput, latency) Designed for latency-sensitive, transactional workloads	Consistent storage performance (IOPS, throughput, latency) Designed for latency-sensitive,	Flexibility in provisioning storage, IOPS, and throughput independently Balances price performance for a wide variety of transactional workloads	Provides burstable IOPS Balances price performance for a wide variety of transactional workloads

Characteristic	Provisioned IOPS (io2 Block Express)	Provisioned IOPS (io1)	General Purpose (gp3)	General Purpose (gp2)
		transactional workloads		
Use cases	Business-critical transactional workloads that require sub-milli second latency and sustained IOPS performan ce up to 256,000 IOPS	Transactional workloads that require sustained IOPS performance up to 256,000 IOPS	Broad range of workloads running on medium-si zed relationa l databases in development/ test environme nts	Broad range of workloads running on medium-si zed relationa l databases in development/ test environme nts
Latency	Sub-milli second, provided consistently 99.9% of the time	Single-di git milliseco nd, provided consistently 99.9% of the time	Single-di git milliseco nd, provided consistently 99% of the time	Single-di git milliseco nd, provided consistently 99% of the time
Volume size	100–65,536 GiB	100–65,536 GiB (20–16,384 GiB on RDS for SQL Server)	20–65,536 GiB (16,384 GiB on RDS for SQL Server)	20–65,536 GiB (16,384 GiB on RDS for SQL Server)

Characteristic	Provisioned IOPS (io2 Block Express)	Provisioned IOPS (io1)	General Purpose (gp3)	General Purpose (gp2)
Maximum IOPS	256,000	256,000 (64,000 on RDS for SQL Server)	64,000 (16,000 on RDS for SQL Server)	64,000 (16,000 on RDS for SQL Server) **Note** You can't provision IOPS directly on gp2 storage. IOPS varies with the allocated storage size.

Characteristic	Provisioned IOPS (io2 Block Express)	Provisioned IOPS (io1)	General Purpose (gp3)	General Purpose (gp2)
Maximum throughput	Scales based on Provisioned IOPS up to 4,000 MB/s Throughput scales proportionally up to 0.256 MiB/s per provisioned IOPS. Maximum throughput of 4,000 MiB/s can be achieved at 256,000 IOPS with a 16-KiB I/O size and 16,000 IOPS or higher with a 256-KiB I/O size. For instances not based on the AWS Nitro System, maximum throughput of 2,000 MiB/s can be achieved at 128,000 IOPS with a 16-KiB I/O size.	Scales based on Provisioned IOPS up to 4,000 MB/s	Provision additional throughput up to 4,000 MB/s (1000 MB/s on RDS for SQL Server)	1000 MB/s (250 MB/s on RDS for SQL Server)
AWS CLI and RDS API name	io2	io1	gp3	gp2

Automatic striping across SSD volumes

When you select General Purpose SSD or Provisioned IOPS SSD, depending on the engine selected and the amount of storage requested, Amazon RDS automatically stripes across multiple volumes to enhance performance, as shown in the following table.

Database engine	Amazon RDS storage size	Number of volumes provisioned
Db2	Less than 400 GiB	1
Db2	400–65,536 GiB	4
MariaDB, MySQL, and PostgreSQL	Less than 400 GiB	1
MariaDB, MySQL, and PostgreSQL	400–65,536 GiB	4
Oracle	Less than 200 GiB	1
Oracle	200–65,536 GiB	4
SQL Server	Any	1

Performance impact when you modify an SSD volume

When you modify a General Purpose SSD or Provisioned IOPS SSD volume, it goes through a sequence of states. While the volume is in the `optimizing` state, your volume performance is between the source and target configuration specifications. Transitional volume performance will be no less than the lower of the two specifications.

When you modify an instance's storage so that it goes from one volume to four volumes, or when you modify an instance using magnetic storage, Amazon RDS doesn't use the Elastic Volumes feature. Instead, Amazon RDS provisions new volumes and transparently moves the data from the old volume to the new volumes. This operation consumes a significant amount of IOPS and throughput of both the old and new volumes. Depending on the size of the volume and the amount of database workload present during the modification, this operation can consume a high

amount of IOPS, significantly increase I/O latency, and take several hours to complete, while the RDS instance remains in the `Modifying` state.

Baseline and maximum IOPS rates for EBS-optimized instances

EBS-optimized instances have a baseline and maximum IOPS rate. The maximum IOPS rate is enforced at the DB instance level. A set of EBS volumes that combine to have an IOPS rate that is higher than the maximum can't exceed the instance-level threshold. For example, if the maximum IOPS for a particular DB instance class is 40,000, and you attach four 64,000 IOPS EBS volumes, the maximum IOPS is 40,000 rather than 256,000. For the IOPS maximum specific to each EC2 instance type, see Supported instance types in the *Amazon EC2 User Guide for Linux Instances*.

Magnetic storage (legacy, not recommended)

Amazon RDS also supports magnetic storage for backward compatibility. We recommend that you use General Purpose SSD or Provisioned IOPS SSD for any new storage needs. The following are some limitations for magnetic storage:

- Doesn't allow you to scale storage when using the SQL Server database engine.
- Doesn't allow you to convert to a different storage type when using the SQL Server database engine.
- Doesn't support storage autoscaling.
- Doesn't support zero-ETL integrations with Amazon Redshift.
- Doesn't support elastic volumes.
- Limited to a maximum size of 3 TiB.
- Limited to a maximum of 1,000 IOPS.

Additional storage volumes

With RDS for Oracle and RDS for SQL Server, you can attach up to three additional storage volumes to your DB instance. Depending on your workload requirements, choose between gp3 and io2 storage for each volume.

Additional storage volumes provide the following benefits:

- **Flexible storage configuration and performance optimization** – Mix different storage types (gp3 and io2) to optimize for both cost and performance based on your data access patterns.

Separate frequently accessed data on high-performance io2 storage from archival data on cost-effective gp3 storage.

- **Enhanced capacity** – Scale your total storage up to 256 TiB per DB instance by combining primary and additional storage volumes.
- **Expand and reduce storage capacity as needed** – Create a volume when you need additional storage, as during data migration, and then later delete the volume. In this way, you can expand and reduce the total DB instance storage.
- **Online data movement** – Use the built-in capabilities of Oracle Database to move data between volumes without downtime.

> **Note**
>
> You can remove additional storage volumes but you can't remove the primary volume.

The additional volumes must use the volume names shown in the following table.

RDS for Oracle volume name	RDS for SQL Server volume name
`rdsdbdata2`	`H:`
`rdsdbdata3`	`I:`
`rdsdbdata4`	`J:`

For more information about working with additional storage volumes, see the following sections:

- Working with storage for Amazon RDS DB instances
- Working with storage in RDS for Oracle
- Working with storage in RDS for SQL Server

Dedicated log volume (DLV)

You can use a dedicated log volume (DLV) for a DB instance that uses Provisioned IOPS (PIOPS) storage by using the Amazon RDS console, AWS CLI, or Amazon RDS API. A DLV moves PostgreSQL

database transaction logs and MySQL/MariaDB redo logs and binary logs to a storage volume that's separate from the volume containing the database tables. A DLV makes transaction write logging more efficient and consistent. DLVs are ideal for databases with large allocated storage, high I/O per second (IOPS) requirements, or latency-sensitive workloads.

DLVs are supported for PIOPS storage (io1 and io2 Block Express), and are created with a fixed size of 1,024 GiB and 3,000 Provisioned IOPS.

> **Note**
>
> DLVs aren't supported for General Purpose storage (gp2 and gp3).

Amazon RDS supports DLVs in all AWS Regions for the following versions:

- MariaDB 10.6.7 and higher 10 versions
- MySQL 8.0.28 and higher 8.0 versions, MySQL 8.4.3 and higher 8.4 versions
- PostgreSQL 13.10 and higher 13 versions, 14.7 and higher 14 versions, 15.2 and higher 15 versions, and 16.1 and higher 16 versions

RDS supports DLVs with Multi-AZ deployments. When you modify or create a Multi-AZ instance, A DLV is created for both the primary and the secondary.

RDS supports DLVs with read replicas. If the primary DB instance has a DLV enabled, all read replicas created after enabling DLV will also have a DLV. Any read replicas created before the switch to DLV will not have it enabled unless explicitly modified to do so. We recommend all read replicas attached to a primary instance before DLV was enabled also be manually modified to have A DLV.

After you modify the DLV setting for a DB instance, the DB instance must be rebooted.

For information on enabling a DLV, see Using a dedicated log volume (DLV).

Monitoring database performance

Amazon RDS provides several metrics that you can use to determine how your DB instance is performing. You can view the metrics on the summary page for your instance in Amazon RDS Management Console. You can also use Amazon CloudWatch to monitor these metrics. For more information, see Viewing metrics in the Amazon RDS console. Enhanced Monitoring provides more detailed I/O metrics; for more information, see Monitoring OS metrics with Enhanced Monitoring.

The following metrics are useful for monitoring performance for your DB instance:

- `DiskQueueDepth` – The number of I/O requests in the queue waiting to be serviced. These are I/O requests that have been submitted by the application but have not been sent to the device because the device is busy servicing other I/O requests. Time spent waiting in the queue is a component of latency and service time (not available as a metric). This metric is reported as the average queue depth for a given time interval. Amazon RDS reports queue depth at 1-minute intervals. Typical values for queue depth range from zero to several hundred.
- `EBSByteBalance%` – The percentage of throughput credits remaining in the burst bucket of your RDS database. This metric is available for basic monitoring only. The metric value is based on the throughput of all volumes, including the root volume, rather than on only those volumes containing database files.

  When this metric approaches zero, it means that your DB instance is running out of computing capacity. If this happens regularly, consider upgrading to a larger instance class size, for example from db.r6g.large to db.r6g.xlarge. For more information, see DB instance class.
- `ReadIOPS` and `WriteIOPS` – The number of I/O operations completed each second. This metric is reported as the average IOPS for a given time interval. Amazon RDS reports read and write IOPS separately at 1-minute intervals. `TotalIOPS` is the sum of the read and write IOPS. Typical values for IOPS range from zero to tens of thousands per second.

  If your `TotalIOPS` values regularly approach the Provisioned IOPS value that you have set for your DB instance, then consider increasing the Provisioned IOPS (io1, io2 Block Express, and gp3 storage types).

  Measured IOPS values are independent of the size of the individual I/O operation. This means that when you measure I/O performance, make sure to look at the throughput of the instance, not simply the number of I/O operations.
- `ReadLatency` and `WriteLatency` – The elapsed time between the submission of an I/O request and its completion. This metric is reported as the average latency for a given time interval. Amazon RDS reports read and write latency separately at 1-minute intervals. Typical values for latency are in milliseconds (ms).
- `ReadThroughput` and `WriteThroughput` – The number of bytes each second that are transferred to or from disk. This metric is reported as the average throughput for a given time interval. Amazon RDS reports read and write throughput separately at 1-minute intervals using units of bytes per second (B/s). Typical values for throughput range from zero to the I/O channel's maximum bandwidth.

If your throughput values regularly approach the maximum throughput for your DB instance, then consider provisioning more storage throughput if you're using the gp3 storage type.

Factors that affect database performance

System activities, database workload, and DB instance class can affect database performance.

Topics

- System activities
- Database workload
- DB instance class

System activities

The following system-related activities consume I/O capacity and might reduce DB instance performance while in progress:

- Multi-AZ standby creation
- Read replica creation
- Changing storage types

Database workload

In some cases, your database or application design results in concurrency issues, locking, or other forms of database contention. In these cases, you might not be able to use all the provisioned bandwidth directly. In addition, you might encounter the following workload-related situations:

- The throughput limit of the underlying instance type is reached.
- Queue depth is consistently less than 1 because your application isn't driving enough I/O operations.
- You experience query contention in the database even though some I/O capacity is unused.

In some cases, there isn't a system resource that is at or near a limit, and adding threads doesn't increase the database transaction rate. In such cases, the bottleneck is most likely contention in

the database. The most common forms are row lock and index page lock contention, but there are many other possibilities. If this is your situation, seek the advice of a database performance tuning expert.

DB instance class

To get the most performance out of your Amazon RDS DB instance, choose a current generation instance type with enough bandwidth and IOPS to support your storage type. For example, you can choose Amazon EBS–optimized instances and instances with 10-gigabit network connectivity.

> **Important**
>
> Depending on the instance class you're using, you might see lower bandwidth, throughput and IOPS performance than the maximum that you can provision with RDS. For specific information on bandwidth, throughput and IOPS performance for DB instance classes, see Amazon EBS–optimized instances in the *Amazon EC2 User Guide*. We recommend that you determine the maximum bandwidth, throughput and IOPS for the instance class before setting a Provisioned IOPS and throughput value for your storage volumes in your DB instance.

We encourage you to use the latest generation of instances to get the best performance. Previous generation DB instances can also have lower maximum storage.

Some older 32-bit file systems might have lower storage capacities. To determine the storage capacity of your DB instance, you can use the describe-valid-db-instance-modifications AWS CLI command.

The following list shows the maximum storage that most DB instance classes can scale to for each database engine:

- Db2 – 64 TiB
- MariaDB – 64 TiB
- Microsoft SQL Server – 64 TiB
- MySQL – 64 TiB
- Oracle – 64 TiB
- PostgreSQL – 64 TiB

The following table shows some exceptions for maximum storage (in TiB). All RDS for Microsoft SQL Server DB instances apart from io2 Block Express storage have a maximum storage of 16 TiB, so there are no entries for SQL Server.

Instance class	Db2	MariaDB	MySQL	Oracle	PostgreSQL
db.m3 – standard instance classes					
db.t4g – burstable-performance instance classes					
db.t4g.medium	N/A	16	16	N/A	32
db.t4g.small	N/A	16	16	N/A	16
db.t4g.micro	N/A	6	6	N/A	6
db.t3 – burstable-performance instance classes					
db.t3.medium	32	16	16	32	32
db.t3.small	32	16	16	32	16
db.t3.micro	N/A	6	6	32	6
db.t2 – burstable-performance instance classes					

For more details about all instance classes supported, see Previous generation DB instances.

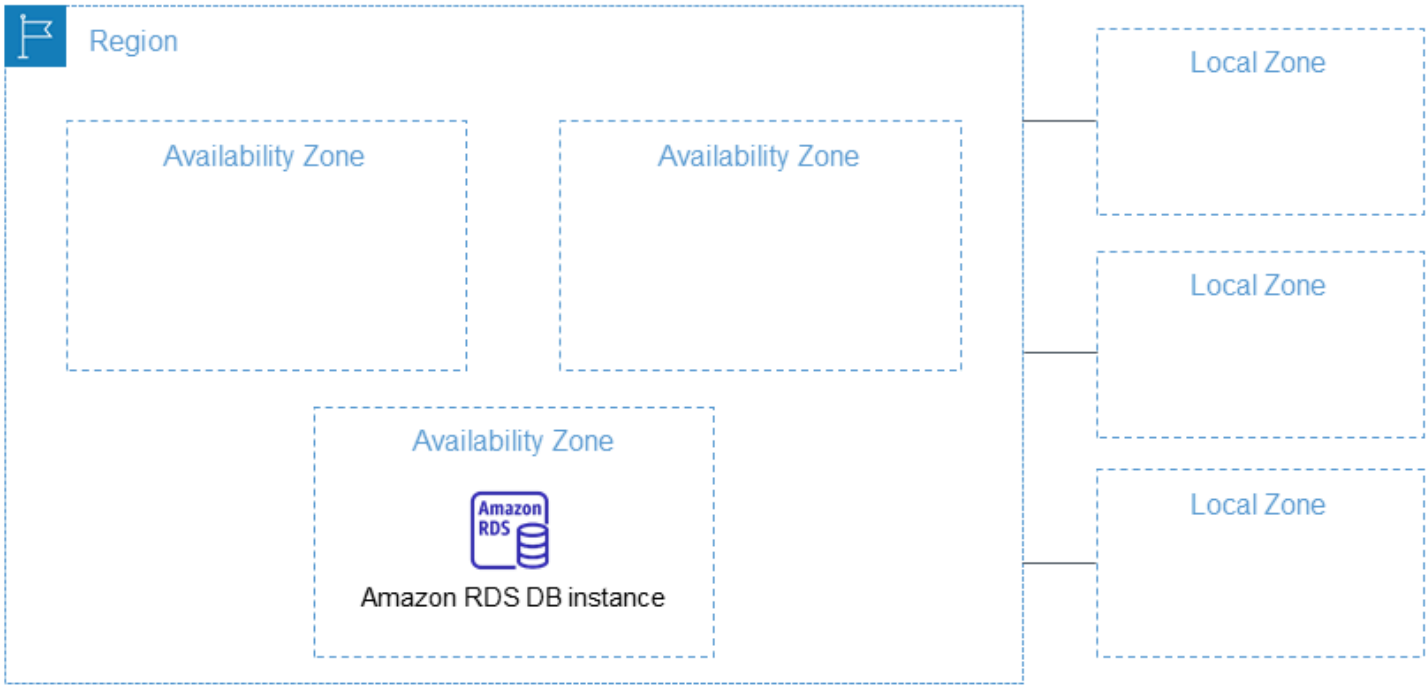
Regions, Availability Zones, and Local Zones

Amazon cloud computing resources are hosted in multiple locations world-wide. These locations are composed of AWS Regions, Availability Zones, and Local Zones. Each *AWS Region* is a separate geographic area. Each AWS Region has multiple, isolated locations known as *Availability Zones*.

> **Note**
>
> For information about finding the Availability Zones for an AWS Region, see Describe your Availability Zones in the Amazon EC2 documentation.

By using Local Zones, you can place resources, such as compute and storage, in multiple locations closer to your users. Amazon RDS enables you to place resources, such as DB instances, and data in multiple locations. Resources aren't replicated across AWS Regions unless you do so specifically.

Amazon operates state-of-the-art, highly-available data centers. Although rare, failures can occur that affect the availability of DB instances that are in the same location. If you host all your DB instances in one location that is affected by such a failure, none of your DB instances will be available.



It is important to remember that each AWS Region is completely independent. Any Amazon RDS activity you initiate (for example, creating database instances or listing available database

instances) runs only in your current default AWS Region. The default AWS Region can be changed in the console, or by setting the `AWS_DEFAULT_REGION` environment variable. Or it can be overridden by using the `--region` parameter with the AWS Command Line Interface (AWS CLI). For more information, see Configuring the AWS Command Line Interface, specifically the sections about environment variables and command line options.

Amazon RDS supports special AWS Regions called AWS GovCloud (US). These are designed to allow US government agencies and customers to move more sensitive workloads into the cloud. The AWS GovCloud (US) Regions address the US government's specific regulatory and compliance requirements. For more information, see What is AWS GovCloud (US)?

To create or work with an Amazon RDS DB instance in a specific AWS Region, use the corresponding regional service endpoint.

AWS Regions

Each AWS Region is designed to be isolated from the other AWS Regions. This design achieves the greatest possible fault tolerance and stability.

When you view your resources, you see only the resources that are tied to the AWS Region that you specified. This is because AWS Regions are isolated from each other, and we don't automatically replicate resources across AWS Regions.

Region availability

The following table shows the AWS Regions where Amazon RDS is currently available and the endpoint for each Region.

Region Name	Region	Endpoint	Protocol	
US East (Ohio)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
		rds-fips.us-east-2.api.aws	HTTPS	
		rds.us-east-2.api.aws	HTTPS	
		rds-fips.us-east-2.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol
US East (N. Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS
		rds-fips.us-east-1.api.aws	HTTPS
		rds-fips.us-east-1.amazonaws.com	HTTPS
		rds.us-east-1.api.aws	HTTPS
US West (N. California)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
		rds.us-west-1.api.aws	HTTPS
		rds-fips.us-west-1.amazonaws.com	HTTPS
		rds-fips.us-west-1.api.aws	HTTPS
US West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
		rds-fips.us-west-2.amazonaws.com	HTTPS
		rds.us-west-2.api.aws	HTTPS
		rds-fips.us-west-2.api.aws	HTTPS
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
		rds.af-south-1.api.aws	HTTPS
Asia Pacific (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
		rds.ap-east-1.api.aws	HTTPS
Asia Pacific (Hyderabad)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
		rds.ap-south-2.api.aws	HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com rds.ap-southeast-3.api.aws	HTTPS HTTPS
Asia Pacific (Malaysia)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS
Asia Pacific (Melbourne)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com rds.ap-southeast-4.api.aws	HTTPS HTTPS
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com rds.ap-south-1.api.aws	HTTPS HTTPS
Asia Pacific (New Zealand)	ap-southeast-6	rds.ap-southeast-6.amazonaws.com	HTTPS
Asia Pacific (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com rds.ap-northeast-3.api.aws	HTTPS HTTPS
Asia Pacific (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com rds.ap-northeast-2.api.aws	HTTPS HTTPS
Asia Pacific (Singapore)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com rds.ap-southeast-1.api.aws	HTTPS HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
		rds.ap-southeast-2.api.aws	HTTPS
Asia Pacific (Taipei)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS
Asia Pacific (Thailand)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
		rds.ap-northeast-1.api.aws	HTTPS
Canada (Central)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
		rds.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.amazonaws.com	HTTPS
Canada West (Calgary)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
		rds-fips.ca-west-1.amazonaws.com	HTTPS
Europe (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
		rds.eu-central-1.api.aws	HTTPS
Europe (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS
		rds.eu-west-1.api.aws	HTTPS

Region Name	Region	Endpoint	Protocol
Europe (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
		rds.eu-west-2.api.aws	HTTPS
Europe (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
		rds.eu-south-1.api.aws	HTTPS
Europe (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
		rds.eu-west-3.api.aws	HTTPS
Europe (Spain)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
		rds.eu-south-2.api.aws	HTTPS
Europe (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS
		rds.eu-north-1.api.aws	HTTPS
Europe (Zurich)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS
		rds.eu-central-2.api.aws	HTTPS
Israel (Tel Aviv)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS
		rds.il-central-1.api.aws	HTTPS
Mexico (Central)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS
Middle East (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS
		rds.me-south-1.api.aws	HTTPS
Middle East (UAE)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS
		rds.me-central-1.api.aws	HTTPS

Region Name	Region	Endpoint	Protocol
South America (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com rds.sa-east-1.api.aws	HTTPS HTTPS
AWS GovCloud (US-East)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com rds.us-gov-east-1.api.aws	HTTPS HTTPS
AWS GovCloud (US-West)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com rds.us-gov-west-1.api.aws	HTTPS HTTPS

If you do not explicitly specify an endpoint, the US West (Oregon) endpoint is the default.

When you work with a DB instance using the AWS CLI or API operations, make sure that you specify its regional endpoint.

Availability Zones

When you create a DB instance, you can choose an Availability Zone or have Amazon RDS choose one for you randomly. An Availability Zone is represented by an AWS Region code followed by a letter identifier (for example, `us-east-1a`).

Use the describe-availability-zones Amazon EC2 command as follows to describe the Availability Zones within the specified Region that are enabled for your account.

```
aws ec2 describe-availability-zones --region region-name
```

For example, to describe the Availability Zones within the US East (N. Virginia) Region (us-east-1) that are enabled for your account, run the following command:

```
aws ec2 describe-availability-zones --region us-east-1
```

You can't choose the Availability Zones for the primary and secondary DB instances in a Multi-AZ DB deployment. Amazon RDS chooses them for you randomly. For more information about Multi-AZ deployments, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

> **Note**
>
> Random selection of Availability Zones by RDS doesn't guarantee an even distribution of DB instances among Availability Zones within a single account or DB subnet group. You can request a specific AZ when you create or modify a Single-AZ instance, and you can use more-specific DB subnet groups for Multi-AZ instances. For more information, see Creating an Amazon RDS DB instance and Modifying an Amazon RDS DB instance.

Local Zones

A *Local Zone* is an extension of an AWS Region that is geographically close to your users. You can extend any VPC from the parent AWS Region into Local Zones. To do so, create a new subnet and assign it to the AWS Local Zone. When you create a subnet in a Local Zone, your VPC is extended to that Local Zone. The subnet in the Local Zone operates the same as other subnets in your VPC.

When you create a DB instance, you can choose a subnet in a Local Zone. Local Zones have their own connections to the internet and support Direct Connect. Thus, resources created in a Local Zone can serve local users with very low-latency communications. For more information, see AWS Local Zones.

A Local Zone is represented by an AWS Region code followed by an identifier that indicates the location, for example `us-west-2-lax-1a`.

> **Note**
>
> A Local Zone can't be included in a Multi-AZ deployment.

To use a Local Zone

1. Enable the Local Zone in the Amazon EC2 console.

   For more information, see Enabling Local Zones in the *Amazon EC2 User Guide.*
2. Create a subnet in the Local Zone.

For more information, see Creating a subnet in your VPC in the *Amazon VPC User Guide.*

3. Create a DB subnet group in the Local Zone.

   When you create a DB subnet group, choose the Availability Zone group for the Local Zone.

   For more information, see Creating a DB instance in a VPC.

4. Create a DB instance that uses the DB subnet group in the Local Zone.

   For more information, see Creating an Amazon RDS DB instance.

> **Important**
>
> Currently, the only AWS Local Zone where Amazon RDS is available is Los Angeles in the US West (Oregon) Region.

Supported features in Amazon RDS by AWS Region and DB engine

Support for Amazon RDS features and options varies across AWS Regions and specific versions of each DB engine. To identify RDS DB engine version support and availability in a given AWS Region, you can use the following sections.

Amazon RDS features are different from engine-native features and options. For more information on engine-native features and options, see Engine-native features.

Supported Regions and DB engines

- Table conventions
- Feature quick reference
- Supported Regions and DB engines for Amazon RDS Blue/Green Deployments
- Supported Regions and DB engines for cross-Region automated backups in Amazon RDS
- Supported Regions and DB engines for cross-Region read replicas in Amazon RDS
- Supported Regions and DB engines for database activity streams in Amazon RDS
- Supported Regions and DB engines for dual-stack mode in Amazon RDS
- Supported Regions and DB engines for exporting snapshots to S3 in Amazon RDS
- Supported Regions and DB engines for IAM database authentication in Amazon RDS
- Supported Regions and DB engines for Kerberos authentication in Amazon RDS
- Supported Regions and DB engines for Multi-AZ DB clusters in Amazon RDS
- Supported Regions and DB engines for Performance Insights in Amazon RDS
- Supported Regions and DB engines for RDS Custom
- Supported Regions and DB engines for Amazon RDS Proxy
- Supported Regions and DB engines for the Secrets Manager integration with Amazon RDS
- Supported Regions and DB engines for Amazon RDS zero-ETL integrations
- Engine-native features in Amazon RDS

Table conventions

The tables in the feature sections use these patterns to specify version numbers and level of availability:

- **Version x.y** – The specific version alone is available.
- **Version x.y and higher** – The specified version and all higher minor versions of its major version are supported. For example, "version 10.11 and higher" means that versions 10.11, 10.11.1, and 10.12 are available.

Feature quick reference

The following quick reference table lists each feature and available RDS DB engine. Region and specific version availability appears in the later feature sections.

Featu	RDS for Db2	RDS for MariaDB	RDS for MySQL	RDS for Oracle	RDS for PostgreSQL	RDS for SQL Server
Blue/ Gree n Deplo ts	Not available	Available	Available	Not available	Available	Not available
Cross- Reg ion autom backu	Available	Available	Available	Available	Available	Available
Cross- Reg ion read replic	Not available	Available	Available	Available	Available	Available
Datab activit stream	Not available	Not available	Not available	Available	Not available	Available
Dual- stac	Not available	Available	Available	Available	Available	Available

Featu	RDS for Db2	RDS for MariaDB	RDS for MySQL	RDS for Oracle	RDS for PostgreSQL	RDS for SQL Server
k mode						
Export Snaps to Amaz S3	Not available	Available	Available	Not available	Available	Not available
AWS Identit and Access Manag t (IAM) databa auther ation	Not available	Available	Available	Not available	Available	Not available
Kerber auther ation	Available	Not available	Available	Available	Available	Available
Multi-AZ DB cluste	Not available	Not available	Available	Not available	Available	Not available
Perfor ce Insigh	Not available	Available	Available	Available	Available	Available
RDS Custor	Not available	Not available	Not available	Available	Not available	Available

Featu	RDS for Db2	RDS for MariaDB	RDS for MySQL	RDS for Oracle	RDS for PostgreSQL	RDS for SQL Server
RDS Proxy	Not available	Available	Available	Not available	Available	Available
Secret Manag integra on	Available	Available	Available	Available	Available	Available

Supported Regions and DB engines for Amazon RDS Blue/Green Deployments

A blue/green deployment copies a production database environment in a separate, synchronized staging environment. By using Amazon RDS Blue/Green Deployments, you can make changes to the database in the staging environment without affecting the production environment. For example, you can upgrade the major or minor DB engine version, change database parameters, or make schema changes in the staging environment. When you are ready, you can promote the staging environment to be the new production database environment. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

Blue/green deployments are supported in all AWS Regions.

Blue/green deployments are *not* supported with the following engines:

- RDS for Db2
- RDS for SQL Server
- RDS for Oracle

Topics

- Blue/green deployments with RDS for MariaDB
- Blue/green deployments with RDS for MySQL
- Blue/green deployments with RDS for PostgreSQL

Blue/green deployments with RDS for MariaDB

For RDS for MariaDB, blue/green deployments are supported for the following versions:

- RDS for MariaDB 11.8 (All available versions)
- RDS for MariaDB 11.4 (All available versions)
- RDS for MariaDB 10.2 and higher 10 versions

Blue/green deployments with RDS for MySQL

For RDS for MySQL, blue/green deployments are supported for the following versions:

- RDS for MySQL 8.4 (All available versions)
- RDS for MySQL 8.0 (All available versions)
- RDS for MySQL 5.7 (All available versions)

Blue/green deployments with RDS for PostgreSQL

For RDS for PostgreSQL, blue/green deployments are supported for version 11.1 and all higher major and minor versions.

> **Note**
>
> Under certain conditions, RDS for PostgreSQL uses logical replication instead of physical replication to keep the green environment in sync with the blue environment. For more information, see the section called “PostgreSQL replication methods”.

Supported Regions and DB engines for cross-Region automated backups in Amazon RDS

By using backup replication in Amazon RDS, you can configure your RDS DB instance to replicate snapshots and transaction logs to a destination Region. When backup replication is configured for a DB instance, RDS starts a cross-Region copy of all snapshots and transaction logs when they're ready. For more information, see Replicating automated backups to another AWS Region.

For information about AWS Regions where backup replication is available, see Replicating automated backups to another AWS Region.

Topics

- Backup replication with RDS for Db2
- Backup replication with RDS for MariaDB
- Backup replication with RDS for MySQL
- Backup replication with RDS for Oracle
- Backup replication with RDS for PostgreSQL
- Backup replication with RDS for SQL Server

Backup replication with RDS for Db2

Amazon RDS supports backup replication for all currently available versions of RDS for Db2.

Backup replication with RDS for MariaDB

Amazon RDS supports backup replication for all currently available versions of RDS for MariaDB.

Backup replication with RDS for MySQL

Amazon RDS supports backup replication for all currently available versions of RDS for MySQL.

Backup replication with RDS for Oracle

Amazon RDS supports backup replication for all currently available versions of RDS for Oracle.

Backup replication with RDS for PostgreSQL

Amazon RDS supports backup replication for all currently available versions of RDS for PostgreSQL.

Backup replication with RDS for SQL Server

Amazon RDS supports backup replication for all currently available versions of RDS for SQL Server.

Supported Regions and DB engines for cross-Region read replicas in Amazon RDS

By using cross-Region read replicas in Amazon RDS, you can create a MariaDB, MySQL, Oracle, PostgreSQL, or SQL Server read replica in a different Region from the source DB instance. For

more information about cross-Region read replicas, including source and destination Region considerations, see Creating a read replica in a different AWS Region.

Cross-Region read replicas are not available for the following engines:

- RDS for Db2

Topics

- Cross-Region read replicas with RDS for MariaDB
- Cross-Region read replicas with RDS for MySQL
- Cross-Region read replicas with RDS for Oracle
- Cross-Region read replicas with RDS for PostgreSQL
- Cross-Region read replicas with RDS for SQL Server

Cross-Region read replicas with RDS for MariaDB

Cross-Region read replicas with RDS for MariaDB are available in all Regions for the following versions:

- RDS for MariaDB 11.8 (All available versions)
- RDS for MariaDB 11.4 (All available versions)
- RDS for MariaDB 10.11 (All available versions)
- RDS for MariaDB 10.6 (All available versions)
- RDS for MariaDB 10.5 (All available versions)
- RDS for MariaDB 10.4 (All available versions)

Cross-Region read replicas with RDS for MySQL

Cross-Region read replicas with RDS for MySQL are available in all Regions for the following versions:

- RDS for MySQL 8.4 (All available versions)
- RDS for MySQL 8.0 (All available versions)
- RDS for MySQL 5.7 (All available versions)

Cross-Region read replicas with RDS for Oracle

Cross-Region read replicas for RDS for Oracle are available in all AWS Regions for all supported database versions using Enterprise Edition. Replicas are supported only in non-CDBs and in the single-tenant configuration of the CDB architecture. Cross-Region read replicas aren't supported in the multi-tenant configuration of the CDB architecture.

For more information on additional requirements for cross-Region read replicas with RDS for Oracle, see Requirements and considerations for RDS for Oracle replicas.

Cross-Region read replicas with RDS for PostgreSQL

Cross-Region read replicas with RDS for PostgreSQL are available in all Regions for the following versions:

- RDS for PostgreSQL 18 (All available versions)
- RDS for PostgreSQL 17 (All available versions)
- RDS for PostgreSQL 16 (All available versions)
- RDS for PostgreSQL 15 (All available versions)
- RDS for PostgreSQL 14 (All available versions)
- RDS for PostgreSQL 13 (All available versions)
- RDS for PostgreSQL 12 (All available versions)
- RDS for PostgreSQL 11 (All available versions)
- RDS for PostgreSQL 10 (All available versions)

Cross-Region read replicas with RDS for SQL Server

Cross-Region read replicas with RDS for SQL Server are available in all Regions for the following versions using Microsoft SQL Server Enterprise Edition:

- RDS for SQL Server 2022
- RDS for SQL Server 2019 (Version 15.00.4073.23 and higher)
- RDS for SQL Server 2017 (Version 14.00.3281.6 and higher)
- RDS for SQL Server 2016 (Version 13.00.6300.2 and higher)

Supported Regions and DB engines for database activity streams in Amazon RDS

By using database activity streams in Amazon RDS, you can monitor and set alarms for auditing activity in your Oracle database and SQL Server database. For more information, see Overview of Database Activity Streams.

Database activity streams aren't available with the following engines:

- RDS for Db2
- RDS for MariaDB
- RDS for MySQL
- RDS for PostgreSQL

Topics

- Database activity streams with RDS for Oracle
- Database activity streams with RDS for SQL Server

Database activity streams with RDS for Oracle

The following Regions and engine versions are available for database activity streams with RDS for Oracle.

For more information on additional requirements for database activity streams with RDS for Oracle, see Overview of Database Activity Streams.

Region	RDS for Oracle 21c	RDS for Oracle 19c
US East (N. Virginia)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
US East (Ohio)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using

Region	RDS for Oracle 21c	RDS for Oracle 19c
		either Enterprise Edition (EE) or Standard Edition 2 (SE2)
US West (N. California)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
US West (Oregon)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Africa (Cape Town)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (Hong Kong)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (Hyderabad)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (Jakarta)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (Malaysia)	Not available	Not available
Asia Pacific (Melbourne)	Not available	Not available

Region	RDS for Oracle 21c	RDS for Oracle 19c
Asia Pacific (Mumbai)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (New Zealand)	Not available	Not available
Asia Pacific (Osaka)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (Seoul)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (Singapore)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (Sydney)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Asia Pacific (Taipei)	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available
Asia Pacific (Tokyo)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)

Region	RDS for Oracle 21c	RDS for Oracle 19c
Canada (Central)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Canada West (Calgary)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
China (Beijing)	Not available	Not available
China (Ningxia)	Not available	Not available
Europe (Frankfurt)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Europe (Ireland)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Europe (London)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Europe (Milan)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)

Region	RDS for Oracle 21c	RDS for Oracle 19c
Europe (Paris)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Europe (Spain)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Europe (Stockholm)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Europe (Zurich)	Not available	Not available
Asia Pacific (Melbourne)	Not available	Not available
Mexico (Central)	Not available	Not available
Middle East (Bahrain)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
Middle East (UAE)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
South America (São Paulo)	Not available	Oracle Database 19.0.0.0.ru-2019-07.rur-2019-07.r1 and higher, using either Enterprise Edition (EE) or Standard Edition 2 (SE2)
AWS GovCloud (US-East)	Not available	Not available

Region	RDS for Oracle 21c	RDS for Oracle 19c
AWS GovCloud (US-West)	Not available	Not available

Database activity streams with RDS for SQL Server

The following Regions and engine versions are available for database activity streams with RDS for SQL Server.

For more information on additional requirements for database activity streams with RDS for SQL Server, see Overview of Database Activity Streams.

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
US East (N. Virginia)	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	Not available	Not available	Not available
Asia Pacific (Melbourne)	Not available	Not available	Not available

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions
China (Beijing)	Not available	Not available	Not available
China (Ningxia)	Not available	Not available	Not available
Europe (Frankfurt)	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
Europe (Spain)	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions
Europe (Zurich)	Not available	Not available	Not available
Israel (Tel Aviv)	Not available	Not available	Not available
Mexico (Central)	Not available	Not available	Not available
Middle East (Bahrain)	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	Not available	Not available	Not available
AWS GovCloud (US-West)	Not available	Not available	Not available

Supported Regions and DB engines for dual-stack mode in Amazon RDS

By using dual-stack mode in RDS, resources can communicate with a DB instance over Internet Protocol version 4 (IPv4), Internet Protocol version 6 (IPv6), or both. For more information, see Dual-stack mode.

Topics

- Dual-stack mode with RDS for Db2
- Dual-stack mode with RDS for MariaDB
- Dual-stack mode with RDS for MySQL
- Dual-stack mode with RDS for Oracle
- Dual-stack mode with RDS for PostgreSQL

- Dual-stack mode with RDS for SQL Server

Dual-stack mode with RDS for Db2

The following Regions and engine versions are available for dual-stack mode with RDS for Db2.

Region	RDS for Db2 11.5				
US East (N. Virginia)	All available versions				
US East (Ohio)	All available versions				
US West (N. California)	All available versions				
US West (Oregon)	All available versions				
Africa (Cape Town)	All available versions				
Asia Pacific (Hong Kong)	All available versions				
Asia Pacific (Hyderabad)	Not available				
Asia Pacific (Jakarta)	All available versions				
Asia Pacific (Malaysia)	All available versions				
Asia Pacific (Melbourne)	Not available				

Region	RDS for Db2 11.5				
Asia Pacific (Mumbai)	All available versions				
Asia Pacific (New Zealand)	Not available				
Asia Pacific (Osaka)	All available versions				
Asia Pacific (Seoul)	All available versions				
Asia Pacific (Singapore)	All available versions				
Asia Pacific (Sydney)	All available versions				
Asia Pacific (Taipei)	All available versions				
Asia Pacific (Thailand)	All available versions				
Asia Pacific (Tokyo)	All available versions				
Canada (Central)	All available versions				
Canada West (Calgary)	All available versions				
China (Beijing)	Not available				

Region	RDS for Db2 11.5				
China (Ningxia)	Not available				
Europe (Frankfurt)	All available versions				
Europe (Ireland)	All available versions				
Europe (London)	All available versions				
Europe (Milan)	All available versions				
Europe (Paris)	All available versions				
Europe (Spain)	Not available				
Europe (Stockholm)	All available versions				
Europe (Zurich)	Not available				
Israel (Tel Aviv)	All available versions				
Mexico (Central)	All available versions				
Middle East (Bahrain)	All available versions				

Region	RDS for Db2 11.5
Middle East (UAE)	All available versions
South America (São Paulo)	All available versions
AWS GovCloud (US-East)	Not available
AWS GovCloud (US-West)	Not available

Dual-stack mode with RDS for MariaDB

The following Regions and engine versions are available for dual-stack mode with RDS for MariaDB.

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
US East (N. Virginia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
US West (Oregon)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available	Not available	Not available

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Asia Pacific (Osaka)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Thailand)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
China (Beijing)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Europe (Zurich)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Mexico (Central)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Dual-stack mode with RDS for MySQL

The following Regions and engine versions are available for dual-stack mode with RDS for MySQL.

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
US East (N. Virginia)	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	Not available	Not available	Not available
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions	All available versions
Asia Pacific (Melbourne)	Not available	Not available	Not available
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
Asia Pacific (Singapore)	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	All available versions	All available versions	All available versions
Asia Pacific (Thailand)	All available versions	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions
Europe (Spain)	Not available	Not available	Not available
Europe (Stockholm)	All available versions	All available versions	All available versions
Europe (Zurich)	Not available	Not available	Not available
Israel (Tel Aviv)	All available versions	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
Israel (Tel Aviv)	All available versions	All available versions	All available versions
Mexico (Central)	All available versions	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions	All available versions
Middle East (UAE)	Not available	Not available	Not available
South America (São Paulo)	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions

Dual-stack mode with RDS for Oracle

The following Regions and engine versions are available for dual-stack mode with RDS for Oracle.

Region	RDS for Oracle 21c	RDS for Oracle 19c
US East (N. Virginia)	All available versions	All available versions
US East (Ohio)	All available versions	All available versions
US West (N. California)	All available versions	All available versions
US West (Oregon)	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions

Region	RDS for Oracle 21c	RDS for Oracle 19c
Asia Pacific (Jakarta)	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions
Asia Pacific (Melbourne)	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions
Asia Pacific (Taipei)	All available versions	All available versions
Asia Pacific (Thailand)	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions
Canada (Central)	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions
China (Beijing)	All available versions	All available versions
China (Ningxia)	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions
Europe (London)	All available versions	All available versions
Europe (Milan)	All available versions	All available versions

Region	RDS for Oracle 21c	RDS for Oracle 19c
Europe (Paris)	All available versions	All available versions
Europe (Spain)	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions
Mexico (Central)	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions

Dual-stack mode with RDS for PostgreSQL

The following Regions and engine versions are available for dual-stack mode with RDS for PostgreSQL.

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
US East (N. Virginia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
US East (Ohio)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Asia Pacific (Malaysia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Mumbai	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Asia Pacific (Sydney)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Thailand)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Europe (Frankfurt)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Israel (Tel Aviv)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Mexico (Central)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Dual-stack mode with RDS for SQL Server

The following Regions and engine versions are available for dual-stack mode with RDS for SQL Server.

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
US East (N. Virginia)	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions	All available versions
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
Asia Pacific (Singapore)	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	All available versions	All available versions	All available versions
Asia Pacific (Thailand)	All available versions	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions
Mexico (Central)	All available versions	All available versions	All available versions

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
Middle East (Bahrain)	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions

Supported Regions and DB engines for exporting snapshots to S3 in Amazon RDS

You can export RDS DB snapshot data to an Amazon S3 bucket. You can export all types of DB snapshots—including manual snapshots, automated system snapshots, and snapshots created by AWS Backup. After the data is exported, you can analyze the exported data directly through tools like Amazon Athena or Amazon Redshift Spectrum. For more information, see Exporting DB snapshot data to Amazon S3 for Amazon RDS.

Exporting snapshots to S3 is not available for the following engines:

- RDS for Db2
- RDS for Oracle
- RDS for SQL Server

Topics

- Export snapshots to S3 with RDS for MariaDB
- Export snapshots to S3 with RDS for MySQL
- Export snapshots to S3 with RDS for PostgreSQL

Export snapshots to S3 with RDS for MariaDB

The following Regions and engine versions are available for exporting snapshots to S3 with RDS for MariaDB.

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
US East (N. Virginia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Asia Pacific (Taipei)	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Europe (London)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Mexico (Central)	Not available	Not available	Not available	Not available	Not available	Not available
Middle East (Bahrain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Middle East (UAE)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Export snapshots to S3 with RDS for MySQL

The following Regions and engine versions are available for exporting snapshots to S3 with RDS for MySQL.

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
US East (Ohio)	All available versions	All available versions	All available versions
US East (N. Virginia)	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
Africa (Cape Town)	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	Not available	Not available	Not available
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
Canada West (Calgary)	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions
Mexico (Central)	Not available	Not available	Not available
Middle East (Bahrain)	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
AWS GovCloud (US-West)	All available versions	All available versions	All available versions

Export snapshots to S3 with RDS for PostgreSQL

The following Regions and engine versions are available for exporting snapshots to S3 with RDS for PostgreSQL.

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
US East (Ohio)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US East (N. Virginia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Africa (Cape Town)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Mumbai	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Europe (Milan)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Mexico (Central)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Middle East (Bahrain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
South America (São Paulo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Supported Regions and DB engines for IAM database authentication in Amazon RDS

By using IAM database authentication in Amazon RDS, you can authenticate without a password when you connect to a DB instance. Instead, you use an authentication token. For more information, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.

IAM database authentication isn't available with the following engines:

- RDS for Db2
- RDS for Oracle
- RDS for SQL Server

Topics

- IAM database authentication with RDS for MariaDB
- IAM database authentication with RDS for MySQL

- IAM database authentication with RDS for PostgreSQL

IAM database authentication with RDS for MariaDB

The following Regions and engine versions are available for IAM database authentication with RDS for MariaDB.

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
US East (N. Virginia)	All available versions	All available versions	All available versions	Not available	Not available	Not available
US East (Ohio)	All available versions	All available versions	All available versions	Not available	Not available	Not available
US West (N. California)	All available versions	All available versions	All available versions	Not available	Not available	Not available
US West (Oregon)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Africa (Cape Town)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Asia Pacific (Hyderabad)	Not available	Not available	Not available	Not available	Not available	Not available

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Asia Pacific (Malaysia)	All available versions	All available versions	Not available	Not available	Not available	Not available
Asia Pacific (Melbourne)	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Asia Pacific (New Zealand)	All available versions	All available versions	Not available	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Asia Pacific (Seoul)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Asia Pacific (Singapore)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Asia Pacific (Sydney)	All available versions	All available versions	All available versions	Not available	Not available	Not available

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Asia Pacific (Taipei)	All available versions	All available versions	Not available	Not available	Not available	Not available
Asia Pacific (Thailand)	All available versions	All available versions	Not available	Not available	Not available	Not available
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Canada (Central)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Canada West (Calgary)	All available versions	All available versions	All available versions	Not available	Not available	Not available
China (Beijing)	All available versions	All available versions	All available versions	Not available	Not available	Not available
China (Ningxia)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Europe (Frankfurt)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Europe (Ireland)	All available versions	All available versions	All available versions	Not available	Not available	Not available

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
Europe (London)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Europe (Milan)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Europe (Paris)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Europe (Spain)	Not available	Not available	Not available	Not available	Not available	Not available
Europe (Stockholm)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Europe (Zurich)	Not available	Not available	Not available	Not available	Not available	Not available
Israel (Tel Aviv)	Not available	Not available	Not available	Not available	Not available	Not available
Mexico (Central)	All available versions	All available versions	Not available	Not available	Not available	Not available
Middle East (Bahrain)	All available versions	All available versions	All available versions	Not available	Not available	Not available
Middle East (UAE)	Not available	Not available	Not available	Not available	Not available	Not available

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5	RDS for MariaDB 10.4
South America (São Paulo)	All available versions	All available versions	All available versions	Not available	Not available	Not available
AWS GovCloud (US-East)	All available versions	All available versions	All available versions	Not available	Not available	Not available
AWS GovCloud (US-West)	All available versions	All available versions	All available versions	Not available	Not available	Not available

IAM database authentication with RDS for MySQL

IAM database authentication with RDS for MySQL is available in all Regions for the following versions:

- RDS for MySQL 8.4 – All available versions
- RDS for MySQL 8.0 – All available versions
- RDS for MySQL 5.7 – All available versions

IAM database authentication with RDS for PostgreSQL

IAM database authentication with RDS for PostgreSQL is available in all Regions for the following versions:

- RDS for PostgreSQL 18 – All available versions
- RDS for PostgreSQL 17 – All available versions
- RDS for PostgreSQL 16 – All available versions
- RDS for PostgreSQL 15 – All available versions
- RDS for PostgreSQL 14 – All available versions
- RDS for PostgreSQL 13 – All available versions

- RDS for PostgreSQL 12 – All available versions
- RDS for PostgreSQL 11 – All available versions
- RDS for PostgreSQL 10 – All available versions

Supported Regions and DB engines for Kerberos authentication in Amazon RDS

By using Kerberos authentication in Amazon RDS, you can support external authentication of database users using Kerberos and Microsoft Active Directory. Using Kerberos and Active Directory provides the benefits of single sign-on and centralized authentication of database users.

Kerberos authentication isn't available with the following engines:

- RDS for MariaDB

Although most AWS Regions are active by default for your AWS account, certain Regions are activated only when you manually select them. These Regions are referred to as *opt-in Regions*. In contrast, Regions that are active by default, as soon as your AWS account is created, are referred to as *commercial Regions*, or simply, *Regions*. For opt-in Regions, you must use a regionalized service principal of the form `directoryservice.rds.region_name.amazonaws.com`. For example, for Africa (Cape Town), you must add service principal `directoryservice.rds.af-south-1.amazonaws.com` to your trust policy. For more information, see Kerberos authentication.

Topics

- Kerberos authentication with RDS for Db2
- Kerberos authentication with RDS for MySQL
- Kerberos authentication with RDS for Oracle
- Kerberos authentication with RDS for PostgreSQL
- Kerberos authentication with RDS for SQL Server

Kerberos authentication with RDS for Db2

The following Regions and engine versions are available for Kerberos authentication with RDS for Db2.

Region	RDS for Db2 11.5
US East (N. Virginia)	All versions
US East (Ohio)	All versions
US West (N. California)	All versions
US West (Oregon)	All versions
Africa (Cape Town)	All versions
Asia Pacific (Hong Kong)	Not available
Asia Pacific (Hyderabad)	All versions
Asia Pacific (Jakarta)	All versions
Asia Pacific (Malaysia)	Not available
Asia Pacific (Melbourne)	All versions
Asia Pacific (Mumbai)	All versions
Asia Pacific (New Zealand)	Not available
Asia Pacific (Osaka)	Not available
Asia Pacific (Seoul)	All versions
Asia Pacific (Singapore)	All versions
Asia Pacific (Sydney)	All versions
Asia Pacific (Taipei)	Not available
Asia Pacific (Thailand)	Not available
Asia Pacific (Tokyo)	All versions
Canada (Central)	All versions

Region	RDS for Db2 11.5
Canada West (Calgary)	Not available
China (Beijing)	Not available
China (Ningxia)	Not available
Europe (Frankfurt)	All versions
Europe (Ireland)	All versions
Europe (London)	All versions
Europe (Milan)	All versions
Europe (Paris)	Not available
Europe (Spain)	All versions
Europe (Stockholm)	All versions
Europe (Zurich)	All versions
Israel (Tel Aviv)	All versions
Mexico (Central)	Not available
Middle East (Bahrain)	All versions
Middle East (UAE)	All versions
South America (São Paulo)	All versions
AWS GovCloud (US-East)	Not available
AWS GovCloud (US-West)	Not available

Kerberos authentication with RDS for MySQL

The following Regions and engine versions are available for Kerberos authentication with RDS for MySQL.

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
US East (N. Virginia)	All versions	All versions	All versions
US East (Ohio)	All versions	All versions	All versions
US West (N. California)	All versions	All versions	All versions
US West (Oregon)	All versions	All versions	All versions
Africa (Cape Town)	All versions	All versions	All versions
Asia Pacific (Hong Kong)	All versions	All versions	All versions
Asia Pacific (Hyderabad)	All versions	All versions	All versions
Asia Pacific (Jakarta)	All versions	All versions	All versions
Asia Pacific (Malaysia)	Not available	Not available	Not available
Asia Pacific (Melbourne)	All versions	All versions	All versions
Asia Pacific (Mumbai)	All versions	All versions	All versions
Asia Pacific (New Zealand)	Not available	Not available	Not available
Asia Pacific (Osaka)	All versions	All versions	All versions
Asia Pacific (Seoul)	All versions	All versions	All versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
Asia Pacific (Singapore)	All versions	All versions	All versions
Asia Pacific (Sydney)	All versions	All versions	All versions
Asia Pacific (Taipei)	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available
Asia Pacific (Tokyo)	All versions	All versions	All versions
Canada (Central)	All versions	All versions	All versions
Canada West (Calgary)	Not available	Not available	Not available
China (Beijing)	All versions	All versions	All versions
China (Ningxia)	All versions	All versions	All versions
Europe (Frankfurt)	All versions	All versions	All versions
Europe (Ireland)	All versions	All versions	All versions
Europe (London)	All versions	All versions	All versions
Europe (Milan)	All versions	All versions	All versions
Europe (Paris)	All versions	All versions	All versions
Europe (Spain)	All versions	All versions	All versions
Europe (Stockholm)	All versions	All versions	All versions
Europe (Zurich)	All versions	All versions	All versions
Israel (Tel Aviv)	All versions	All versions	All versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7 (under RDS Extended Support)
Mexico (Central)	Not available	Not available	Not available
Middle East (Bahrain)	All versions	All versions	All versions
Middle East (UAE)	All versions	All versions	All versions
South America (São Paulo)	All versions	All versions	All versions
AWS GovCloud (US-East)	All versions	All versions	All versions
AWS GovCloud (US-West)	All versions	All versions	All versions

Kerberos authentication with RDS for Oracle

The following Regions and engine versions are available for Kerberos authentication with RDS for Oracle.

Region	RDS for Oracle 21c	RDS for Oracle 19c
US East (N. Virginia)	All versions	All versions
US East (Ohio)	All versions	All versions
US West (N. California)	All versions	All versions
US West (Oregon)	All versions	All versions
Africa (Cape Town) (opt-in Region)	All versions	All versions
Asia Pacific (Hong Kong) (opt-in Region)	All versions	All versions

Region	RDS for Oracle 21c	RDS for Oracle 19c
Asia Pacific (Hyderabad) (opt-in Region)	All versions	All versions
Asia Pacific (Jakarta) (opt-in Region)	All versions	All versions
Asia Pacific (Malaysia)	Not available	Not available
Asia Pacific (Melbourne) (opt-in Region)	All versions	All versions
Asia Pacific (Mumbai)	All versions	All versions
Asia Pacific (New Zealand)	Not available	Not available
Asia Pacific (Osaka)	Not available	Not available
Asia Pacific (Seoul)	All versions	All versions
Asia Pacific (Singapore)	All versions	All versions
Asia Pacific (Sydney)	All versions	All versions
Asia Pacific (Taipei)	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available
Asia Pacific (Tokyo)	All versions	All versions
Canada (Central)	All versions	All versions
Canada West (Calgary)	Not available	Not available
China (Beijing)	Not available	Not available
China (Ningxia)	Not available	Not available
Europe (Frankfurt)	All versions	All versions
Europe (Ireland)	All versions	All versions

Region	RDS for Oracle 21c	RDS for Oracle 19c
Europe (London)	All versions	All versions
Europe (Milan) (opt-in Region)	All versions	All versions
Europe (Paris)	Not available	Not available
Europe (Spain) (opt-in Region)	All versions	All versions
Europe (Stockholm)	All versions	All versions
Europe (Zurich) (opt-in Region)	All versions	All versions
Israel (Tel Aviv) (opt-in Region)	All versions	All versions
Mexico (Central)	Not available	Not available
Middle East (Bahrain) (opt-in Region)	All versions	All versions
Middle East (UAE) (opt-in Region)	All versions	All versions
South America (São Paulo)	All versions	All versions
AWS GovCloud (US-East)	All versions	All versions
AWS GovCloud (US-West)	All versions	All versions

Kerberos authentication with RDS for PostgreSQL

The following Regions and engine versions are available for Kerberos authentication with RDS for PostgreSQL.

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
US East (N. Virginia)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
US East (Ohio)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
US West (N. California)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
US West (Oregon)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Africa (Cape Town)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Asia Pacific (Hong Kong)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Asia Pacific (Hyderabad)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Asia Pacific (Jakarta)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Asia Pacific (Malaysia)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Melbourne)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Asia Pacific (Mumbai	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Osaka)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Seoul)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Asia Pacific (Singapore)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Asia Pacific (Sydney)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Asia Pacific (Taipei)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Tokyo)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Canada (Central)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Canada West (Calgary)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
China (Beijing)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
China (Ningxia)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Europe (Frankfurt)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Europe (Ireland)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Europe (London)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Europe (Milan)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Europe (Paris)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Europe (Spain)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Europe (Stockholm)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Europe (Zurich)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Israel (Tel Aviv)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Mexico (Central)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Middle East (Bahrain)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
Middle East (UAE)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
South America (São Paulo)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
AWS GovCloud (US-East)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions
AWS GovCloud (US-West)	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions	All versions

Kerberos authentication with RDS for SQL Server

The following Regions and engine versions are available for Kerberos authentication with RDS for SQL Server.

Region	RDS for SQL Server 2022	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
US East (N. Virginia)	All versions	All versions	All versions	All versions
US East (Ohio)	All versions	All versions	All versions	All versions
US West (N. California)	All versions	All versions	All versions	All versions
US West (Oregon)	All versions	All versions	All versions	All versions
Africa (Cape Town)	All versions	All versions	All versions	All versions
Asia Pacific (Hong Kong)	All versions	All versions	All versions	All versions
Asia Pacific (Hyderabad)	All versions	All versions	All versions	All versions
Asia Pacific (Malaysia)	Not available	Not available	Not available	Not available
Asia Pacific (Melbourne)	All versions	All versions	All versions	All versions
Asia Pacific (Mumbai)	All versions	All versions	All versions	All versions
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available
Asia Pacific (Osaka)	All versions	All versions	All versions	All versions

Region	RDS for SQL Server 2022	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
Asia Pacific (Seoul)	All versions	All versions	All versions	All versions
Asia Pacific (Singapore)	All versions	All versions	All versions	All versions
Asia Pacific (Sydney)	All versions	All versions	All versions	All versions
Asia Pacific (Taipei)	Not available	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available	Not available
Asia Pacific (Tokyo)	All versions	All versions	All versions	All versions
Canada (Central)	All versions	All versions	All versions	All versions
Canada West (Calgary)	Not available	Not available	Not available	Not available
China (Beijing)	All versions	All versions	All versions	All versions
China (Ningxia)	All versions	All versions	All versions	All versions
Europe (Frankfurt)	All versions	All versions	All versions	All versions
Europe (Ireland)	All versions	All versions	All versions	All versions
Europe (London)	All versions	All versions	All versions	All versions
Europe (Milan)	All versions	All versions	All versions	All versions
Europe (Paris)	All versions	All versions	All versions	All versions

Region	RDS for SQL Server 2022	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
Europe (Spain)	All versions	All versions	All versions	All versions
Europe (Stockholm)	All versions	All versions	All versions	All versions
Europe (Zurich)	All versions	All versions	All versions	All versions
Israel (Tel Aviv)	Not available	Not available	Not available	Not available
Mexico (Central)	Not available	Not available	Not available	Not available
Middle East (Bahrain)	All versions	All versions	All versions	All versions
Middle East (UAE)	All versions	All versions	All versions	All versions
South America (São Paulo)	All versions	All versions	All versions	All versions
AWS GovCloud (US-East)	All versions	All versions	All versions	All versions
AWS GovCloud (US-West)	All versions	All versions	All versions	All versions

Supported Regions and DB engines for Multi-AZ DB clusters in Amazon RDS

A Multi-AZ DB cluster deployment in Amazon RDS provides a high availability deployment mode of Amazon RDS with two readable standby DB instances. A Multi-AZ DB cluster has a writer DB instance and two reader DB instances in three separate Availability Zones in the same Region. Multi-AZ DB clusters provide high availability, increased capacity for read workloads, and lower write latency when compared to Multi-AZ DB instance deployments. For more information, see Multi-AZ DB cluster deployments for Amazon RDS.

Multi-AZ DB clusters aren't available with the following engines:

- RDS for Db2
- RDS for MariaDB
- RDS for Oracle
- RDS for SQL Server

Topics

- Multi-AZ DB clusters with RDS for MySQL
- Multi-AZ DB clusters with RDS for PostgreSQL

Multi-AZ DB clusters with RDS for MySQL

The following Regions and engine versions are available for Multi-AZ DB clusters with RDS for MySQL.

Region	RDS for MySQL 8.4	RDS for MySQL 8.0
US East (N. Virginia)	All available versions	All available versions
US East (Ohio)	All available versions	All available versions
US West (N. California)	Not available	Not available
US West (Oregon)	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0
Asia Pacific (Melbourne)	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available
Asia Pacific (Tokyo)	All available versions	All available versions
Canada (Central)	All available versions	All available versions
Canada (Central)	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions
China (Beijing)	All available versions	All available versions
China (Ningxia)	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions
Europe (London)	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0
Europe (Milan)	All available versions	All available versions
Europe (Paris)	All available versions	All available versions
Europe (Spain)	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions
Mexico (Central)	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions
AWS GovCloud (US-East)	Not available	Not available
AWS GovCloud (US-West)	Not available	Not available

You can list the available versions in a Region for a given DB instance class using the AWS CLI. Change the DB instance class to show the available engine versions for it.

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options \
--engine mysql \
--db-instance-class db.r5d.large \
--query '*[]|[?SupportsClusters == `true`].[EngineVersion]'  \
--output text
```

For Windows:

```
aws rds describe-orderable-db-instance-options ^
--engine mysql ^
--db-instance-class db.r5d.large ^
--query "*[]|[?SupportsClusters == `true`].[EngineVersion]"  ^
--output text
```

Multi-AZ DB clusters with RDS for PostgreSQL

The following Regions and engine versions are available for Multi-AZ DB clusters with RDS for PostgreSQL.

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13
US East (N. Virginia)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
US East (Ohio)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
US West (N. California)	Not available	Not available	Not available	Not available	Not available	Not available
US West (Oregon)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13
Africa (Cape Town)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (Hong Kong)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (Hyderabad)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (Jakarta)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (Malaysia)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13
Asia Pacific (Melbourne)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (Mumbai)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Osaka)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (Seoul)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (Singapore)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13
Asia Pacific (Sydney)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Asia Pacific (Taipei)	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Tokyo)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Canada (Central)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Canada West (Calgary)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13
China (Beijing)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
China (Ningxia)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Europe (Frankfurt)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Europe (Ireland)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Europe (London)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13
Europe (Milan)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Europe (Paris)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Europe (Spain)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Europe (Stockholm)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Europe (Zurich)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13
Israel (Tel Aviv)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Mexico (Central)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Middle East (Bahrain)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
Middle East (UAE)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
South America (São Paulo)	All PostgreSQL 18 versions	All PostgreSQL 17 versions	All PostgreSQL 16 versions	All PostgreSQL 15 versions	Version 14.5 and higher	Version 13.4 and version 13.7 and higher
AWS GovCloud (US-East)	Not available	Not available	Not available	Not available	Not available	Not available

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13
AWS GovCloud (US-West)	Not available	Not available	Not available	Not available	Not available	Not available

You can list the available versions in a Region for a given DB instance class using the AWS CLI. Change the DB instance class to show the available engine versions for it.

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options \
--engine postgres \
--db-instance-class db.r5d.large \
--query '*[]|[?SupportsClusters == `true`].[EngineVersion]'  \
--output text
```

For Windows:

```
aws rds describe-orderable-db-instance-options ^
--engine postgres ^
--db-instance-class db.r5d.large ^
--query "*[]|[?SupportsClusters == `true`].[EngineVersion]"  ^
--output text
```

Supported Regions and DB engines for Performance Insights in Amazon RDS

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.

We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

Performance Insights in Amazon RDS expands on existing Amazon RDS monitoring features to illustrate and help you analyze your database performance. With the Performance Insights dashboard, you can visualize the database load on your Amazon RDS DB instance. You can also filter the load by waits, SQL statements, hosts, or users. For more information, see Monitoring DB load with Performance Insights on Amazon RDS.

Performance Insights is available for all RDS DB engines, except RDS for Db2.

For the available DB engines, Performance Insights is available with all of the available engine versions and in all AWS Regions.

For the Region, DB engine, and instance class support information for Performance Insights features, see Amazon RDS DB engine, Region, and instance class support for Performance Insights features.

Supported Regions and DB engines for RDS Custom

Amazon RDS Custom automates database administration tasks and operations. By using RDS Custom, as a database administrator you can access and customize your database environment and operating system. With RDS Custom, you can customize to meet the requirements of legacy, custom, and packaged applications. For more information, see Amazon RDS Custom.

RDS Custom is supported for the following DB engines only:

Topics

- Supported Regions and DB engines for RDS Custom for Oracle
- Supported Regions and DB engines for RDS Custom for SQL Server

Supported Regions and DB engines for RDS Custom for Oracle

The following Regions and engine versions are available for RDS Custom for Oracle.

Region	Oracle Database 19c	Oracle Database 18c	Oracle Database 12c
US East (N. Virginia)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
US East (Ohio)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
US West (N. California)	Not available	Not available	Not available
US West (Oregon)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Africa (Cape Town)	Not available	Not available	Not available
Asia Pacific (Hong Kong)	Not available	Not available	Not available
Asia Pacific (Jakarta)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Asia Pacific (Malaysia)	Not available	Not available	Not available

Region	Oracle Database 19c	Oracle Database 18c	Oracle Database 12c
Asia Pacific (Melbourne)	Not available	Not available	Not available
Asia Pacific (Mumbai)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Asia Pacific (New Zealand)	Not available	Not available	Not available
Asia Pacific (Osaka)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Asia Pacific (Seoul)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Asia Pacific (Singapore)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Asia Pacific (Sydney)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Asia Pacific (Taipei)	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available
Asia Pacific (Tokyo)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR

Region	Oracle Database 19c	Oracle Database 18c	Oracle Database 12c
Canada (Central)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Canada West (Calgary)	Not available	Not available	Not available
China (Beijing)	Not available	Not available	Not available
China (Ningxia)	Not available	Not available	Not available
Europe (Frankfurt)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Europe (Ireland)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Europe (London)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Europe (Milan)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Europe (Paris)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Europe (Stockholm)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
Israel (Tel Aviv)	Not available	Not available	Not available

Region	Oracle Database 19c	Oracle Database 18c	Oracle Database 12c
Mexico (Central)	Not available	Not available	Not available
Middle East (Bahrain)	Not available	Not available	Not available
Middle East (UAE)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
South America (São Paulo)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
AWS GovCloud (US-East)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR
AWS GovCloud (US-West)	19c with the January 2021 or higher RU/RUR	18c with the January 2021 or higher RU/RUR	12.1 and 12.2 with the January 2021 or higher RU/RUR

Supported Regions and DB engines for RDS Custom for SQL Server

You can deploy RDS Custom for SQL Server by using either an RDS provided engine version (RPEV) or a custom engine version (CEV):

- If you use an RPEV, it includes the default Amazon Machine Image (AMI) and SQL Server installation. RDS Custom for SQL Server also supports operating system updates through maintenance actions for RDS provided engine version (RPEV). If you customize or modify the operating system (OS), your changes might not persist during patching, snapshot restore, or automatic recovery.
- If you use a CEV, you choose your own AMI with either pre-installed Microsoft SQL Server or SQL Server that you install using your own media. When using an AWS provided CEV, you choose the latest Amazon EC2 image (AMI) available by AWS, which has the cumulative update (CU) supported by RDS Custom for SQL Server. With a CEV, you can customize both the OS and SQL Server configuration to meet your enterprise needs.

Currently, you can only use the Windows 2019 operating system.

The following AWS Regions and DB engine versions are available for RDS Custom for SQL Server. The engine version support depends on whether you're using RDS Custom for SQL Server with an RPEV, AWS provided CEV, or customer-provided CEV.

Region	RPEV	AWS provided CEV	Customer-provided CEV
US East (N. Virginia)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
US East (Ohio)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22

Region	RPEV	AWS provided CEV	Customer-provided CEV
	SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	and CU22-GDR. SQL Server 2019 Enterpris e, Standard, Developer , or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
US West (N. California)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterpris e, Standard, Developer , or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

Region	RPEV	AWS provided CEV	Customer-provided CEV
US West (Oregon)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

Region	RPEV	AWS provided CEV	Customer-provided CEV
Africa (Cape Town)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
Asia Pacific (Hong Kong)	Not available	Not available	Not available
Asia Pacific (Hyderabad)	Not available	Not available	Not available
Asia Pacific (Jakarta)	Not available	Not available	Not available
Asia Pacific (Malaysia)	Not available	Not available	Not available
Asia Pacific (Melbourne)	Not available	Not available	Not available

Region	RPEV	AWS provided CEV	Customer-provided CEV
Asia Pacific (Mumbai)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
Asia Pacific (New Zealand)	Not available	Not available	Not available

Region	RPEV	AWS provided CEV	Customer-provided CEV
Asia Pacific (Osaka)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

Region	RPEV	AWS provided CEV	Customer-provided CEV
Asia Pacific (Seoul)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

Region	RPEV	AWS provided CEV	Customer-provided CEV
Asia Pacific (Singapore)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

Region	RPEV	AWS provided CEV	Customer-provided CEV
Asia Pacific (Sydney)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
Asia Pacific (Taipei)	Not available	Not available	Not available
Asia Pacific (Thailand)	Not available	Not available	Not available

Region	RPEV	AWS provided CEV	Customer-provided CEV
Asia Pacific (Tokyo)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

Region	RPEV	AWS provided CEV	Customer-provided CEV
Canada (Central)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
Canada West (Calgary)	Not available	Not available	Not available
China (Beijing)	Not available	Not available	Not available
China (Ningxia)	Not available	Not available	Not available

Region	RPEV	AWS provided CEV	Customer-provided CEV
Europe (Frankfurt)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

Region	RPEV	AWS provided CEV	Customer-provided CEV
Europe (Ireland)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

Region	RPEV	AWS provided CEV	Customer-provided CEV
Europe (London)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
Europe (Milan)	Not available	Not available	Not available

Region	RPEV	AWS provided CEV	Customer-provided CEV
Europe (Paris)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
Europe (Spain)	Not available	Not available	Not available

Region	RPEV	AWS provided CEV	Customer-provided CEV
Europe (Stockholm)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
Europe (Zurich)	Not available	Not available	Not available
Israel (Tel Aviv)	Not available	Not available	Not available
Mexico (Central)	Not available	Not available	Not available
Middle East (Bahrain)	Not available	Not available	Not available
Middle East (UAE)	Not available	Not available	Not available

Region	RPEV	AWS provided CEV	Customer-provided CEV
South America (São Paulo)	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.	SQL Server 2022 Enterprise, Standard, Developer, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR. SQL Server 2019 Enterprise, Standard, Developer, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
AWS GovCloud (US-East)	Not available	Not available	Not available
AWS GovCloud (US-West)	Not available	Not available	Not available

Supported Regions and DB engines for Amazon RDS Proxy

Amazon RDS Proxy is a fully managed, highly available database proxy that makes applications more scalable by pooling and sharing established database connections. For more information, see Amazon RDS Proxy.

RDS Proxy isn't available for the following engines:

- RDS for Db2

- RDS for Oracle

Topics

- RDS Proxy with RDS for MariaDB
- RDS Proxy with RDS for MySQL
- RDS Proxy with RDS for PostgreSQL
- RDS Proxy with RDS for SQL Server

RDS Proxy with RDS for MariaDB

The following Regions and engine versions are available for RDS Proxy with RDS for MariaDB.

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5
US East (N. Virginia)	All available versions	All available versions	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Osaka)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Seoul)	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5
Asia Pacific (Singapore)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Thailand)	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5
Europe (Frankfurt)	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for MariaDB 11.8	RDS for MariaDB 11.4	RDS for MariaDB 10.11	RDS for MariaDB 10.6	RDS for MariaDB 10.5
Mexico (Central)	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions	All available versions	All available versions

RDS Proxy with RDS for MySQL

The following Regions and engine versions are available for RDS Proxy with RDS for MySQL.

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7
US East (N. Virginia)	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7
US West (N. California)	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions	All available versions
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available	Not available
Asia Pacific (Thailand)	All available versions	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7
Canada (Central)	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions
Mexico (Central)	All available versions	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions

Region	RDS for MySQL 8.4	RDS for MySQL 8.0	RDS for MySQL 5.7
AWS GovCloud (US-West)	All available versions	All available versions	All available versions

RDS Proxy with RDS for PostgreSQL

The following Regions and engine versions are available for RDS Proxy with RDS for PostgreSQL.

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
US East (N. Virginia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Asia Pacific (Osaka)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available	Not available
Asia Pacific (Thailand)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Canada West (Calgary)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
Europe (Spain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Mexico (Central)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

Region	RDS for PostgreSQL 18	RDS for PostgreSQL 17	RDS for PostgreSQL 16	RDS for PostgreSQL 15	RDS for PostgreSQL 14	RDS for PostgreSQL 13	RDS for PostgreSQL 12	RDS for PostgreSQL 11	RDS for PostgreSQL 10
AWS GovCloud (US-East)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions	All available versions

RDS Proxy with RDS for SQL Server

The following Regions and engine versions are available for RDS Proxy with RDS for SQL Server.

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
US East (N. Virginia)	All available versions	All available versions	All available versions
US East (Ohio)	All available versions	All available versions	All available versions
US West (N. California)	All available versions	All available versions	All available versions
US West (Oregon)	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	All available versions
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions
Asia Pacific (Hyderabad)	All available versions	All available versions	All available versions

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
Asia Pacific (Jakarta)	All available versions	All available versions	All available versions
Asia Pacific (Malaysia)	All available versions	All available versions	All available versions
Asia Pacific (Melbourne)	All available versions	All available versions	All available versions
Asia Pacific (Mumbai)	All available versions	All available versions	All available versions
Asia Pacific (New Zealand)	Not available	Not available	Not available
Asia Pacific (Osaka)	All available versions	All available versions	All available versions
Asia Pacific (Seoul)	All available versions	All available versions	All available versions
Asia Pacific (Singapore)	All available versions	All available versions	All available versions
Asia Pacific (Sydney)	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available	Not available
Asia Pacific (Thailand)	All available versions	All available versions	All available versions
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions
Canada (Central)	All available versions	All available versions	All available versions
Canada West (Calgary)	All available versions	All available versions	All available versions
China (Beijing)	All available versions	All available versions	All available versions
China (Ningxia)	All available versions	All available versions	All available versions
Europe (Frankfurt)	All available versions	All available versions	All available versions

Region	RDS for SQL Server 2019	RDS for SQL Server 2017	RDS for SQL Server 2016
Europe (Ireland)	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	All available versions
Europe (Paris)	All available versions	All available versions	All available versions
Europe (Spain)	All available versions	All available versions	All available versions
Europe (Stockholm)	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	All available versions
Israel (Tel Aviv)	All available versions	All available versions	All available versions
Mexico (Central)	All available versions	All available versions	All available versions
Middle East (Bahrain)	All available versions	All available versions	All available versions
Middle East (UAE)	All available versions	All available versions	All available versions
South America (São Paulo)	All available versions	All available versions	All available versions
AWS GovCloud (US-East)	All available versions	All available versions	All available versions
AWS GovCloud (US-West)	All available versions	All available versions	All available versions

Supported Regions and DB engines for the Secrets Manager integration with Amazon RDS

With AWS Secrets Manager, you can replace hard-coded credentials in your code, including database passwords, with an API call to Secrets Manager to retrieve the secret programmatically. For more information about Secrets Manager, see AWS Secrets Manager User Guide.

You can specify that Amazon RDS manages the master user password in Secrets Manager for an Amazon RDS DB instance or Multi-AZ DB cluster. RDS generates the password, stores it in Secrets Manager, and rotates it regularly. For more information, see Password management with Amazon RDS and AWS Secrets Manager.

Secrets Manager integration is available in all AWS Regions.

Supported Regions and DB engines for Amazon RDS zero-ETL integrations

RDS zero-ETL integrations is a fully managed solution for making transactional data available in Amazon Redshift or Amazon SageMaker after it's written to an Amazon RDS DB instance. For more information, see *Zero-ETL integrations*.

Topics

- Zero-ETL integrations with RDS for MySQL
- Zero-ETL integrations with RDS for PostgreSQL
- Zero-ETL integrations with RDS for Oracle

Zero-ETL integrations with RDS for MySQL

The following Regions and engine versions are available for RDS for MySQL zero-ETL integrations with Amazon Redshift and Amazon SageMaker.

Region	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.4	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.0	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.4	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.0
US East (N. Virginia	All available versions	All available versions	All available versions	All available versions

Region	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.4	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.0	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.4	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.0
US East (Ohio)	All available versions	All available versions	All available versions	All available versions
US West (N. Californ a)	All available versions	All available versions	Not available	Not available
US West (Orego	All available versions	All available versions	All available versions	All available versions
Africa (Cape Town)	All available versions	All available versions	Not available	Not available
Asia Pacific (Hong Kong)	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Hyder d)	All available versions	All available versions	Not available	Not available
Asia Pacific (Jakart	All available versions	All available versions	Not available	Not available

Region	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.4	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.0	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.4	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.0
Asia Pacific (Malays)	Not available	Not available	Not available	Not available
Asia Pacific (Melbo e)	All available versions	All available versions	Not available	Not available
Asia Pacific (Mumb	All available versions	All available versions	Not available	Not available
Asia Pacific (New Zealan	Not available	Not available	Not available	Not available
Asia Pacific (Osaka	All available versions	All available versions	Not available	Not available
Asia Pacific (Seoul)	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Singap e)	All available versions	All available versions	All available versions	All available versions

Region	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.4	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.0	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.4	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.0
Asia Pacific (Sydne	All available versions	All available versions	All available versions	All available versions
Asia Pacific (Taipei)	Not available	Not available	Not available	Not available
Asia Pacific (Thaila)	Not available	Not available	Not available	Not available
Asia Pacific (Tokyo)	All available versions	All available versions	All available versions	All available versions
Canada (Centra	All available versions	All available versions	All available versions	All available versions
Canada West (Calgar	All available versions	All available versions	Not available	Not available
China (Beijing	Not available	Not available	Not available	Not available
China (Ningxi	Not available	Not available	Not available	Not available

Region	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.4	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.0	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.4	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.0
Europe (Frankfurt)	All available versions	All available versions	All available versions	All available versions
Europe (Ireland)	All available versions	All available versions	All available versions	All available versions
Europe (London)	All available versions	All available versions	All available versions	All available versions
Europe (Milan)	All available versions	All available versions	Not available	Not available
Europe (Paris)	All available versions	All available versions	Not available	Not available
Europe (Spain)	All available versions	All available versions	Not available	Not available
Europe (Stockholm)	All available versions	All available versions	All available versions	All available versions
Europe (Zurich)	All available versions	All available versions	Not available	Not available
Israel (Tel Aviv)	All available versions	All available versions	Not available	Not available
Mexico (Centra	Not available	Not available	Not available	Not available

Region	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.4	Zero-ETL integration with Amazon Redshift for RDS for MySQL 8.0	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.4	Zero-ETL integration with Amazon SageMaker for RDS for MySQL 8.0
Mexico (Centra	Not available	Not available	Not available	Not available
Middle East (Bahrai	All available versions	All available versions	Not available	Not available
Middle East (UAE)	All available versions	All available versions	Not available	Not available
South Americ (São Paulo)	All available versions	All available versions	All available versions	All available versions
AWS GovClo (US-East)	Not available	Not available	Not available	Not available
AWS GovClo (US-West)	Not available	Not available	Not available	Not available

Zero-ETL integrations with RDS for PostgreSQL

The following Regions and engine versions are available for RDS for PostgreSQL zero-ETL integrations with Amazon Redshift.

Region	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 15	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 16	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 17
US East (N. Virginia)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
US East (Ohio)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
US West (N. California)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
US West (Oregon)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Africa (Cape Town)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific (Hong Kong)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific (Hyderabad)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher

Region	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 15	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 16	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 17
(Jakarta)			
Asia Pacific (Malaysia)	Not available	Not available	Not available
Asia Pacific (Melbourne)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific (Mumbai)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific (Osaka)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific (Seoul)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific (Singapore)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific (Sydney)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher

Region	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 15	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 16	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 17
Asia Pacific (Thailand)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Asia Pacific (Tokyo)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Canada (Central)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Canada West (Calgary)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
China (Beijing)	Not available	Not available	Not available
China (Ningxia)	Not available	Not available	Not available
Europe (Frankfurt)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Europe (Ireland)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Europe (London)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Europe (Milan)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher

Region	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 15	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 16	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 17
Europe (Paris)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Europe (Spain)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Europe (Stockholm)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Europe (Zurich)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Israel (Tel Aviv)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Middle East (Bahrain)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
Middle East (UAE)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
South America (São Paulo)	Version 15.11 and higher	Version 16.7 and higher	Version 17.3 and higher
AWS GovCloud (US-East)	Not available	Not available	Not available

Region	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 15	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 16	Zero-ETL integration with Amazon Redshift for RDS for PostgreSQL 17
AWS GovCloud (US-West)	Not available	Not available	Not available

Zero-ETL integrations with RDS for Oracle

The following Regions and engine versions are available for RDS for Oracle zero-ETL integrations with Amazon Redshift.

Region	Zero-ETL integration with Amazon Redshift for RDS for Oracle 19c
US East (N. Virginia)	All available versions
US East (Ohio)	All available versions
US West (N. California)	All available versions
US West (Oregon)	All available versions
Africa (Cape Town)	All available versions
Asia Pacific (Hong Kong)	All available versions
Asia Pacific (Hyderabad)	All available versions
Asia Pacific (Jakarta)	All available versions
Asia Pacific (Malaysia)	Not available
Asia Pacific (Melbourne)	All available versions
Asia Pacific (Mumbai)	All available versions

Region	Zero-ETL integration with Amazon Redshift for RDS for Oracle 19c
Asia Pacific (Osaka)	All available versions
Asia Pacific (Seoul)	All available versions
Asia Pacific (Singapore)	All available versions
Asia Pacific (Sydney)	All available versions
Asia Pacific (Thailand)	All available versions
Asia Pacific (Tokyo)	All available versions
Canada (Central)	All available versions
Canada West (Calgary)	All available versions
China (Beijing)	Not available
China (Ningxia)	Not available
Europe (Frankfurt)	All available versions
Europe (Ireland)	All available versions
Europe (London)	All available versions
Europe (Milan)	All available versions
Europe (Paris)	All available versions
Europe (Spain)	All available versions
Europe (Stockholm)	All available versions
Europe (Zurich)	All available versions
Israel (Tel Aviv)	All available versions
Middle East (Bahrain)	All available versions

Region	Zero-ETL integration with Amazon Redshift for RDS for Oracle 19c
Middle East (UAE)	All available versions
South America (São Paulo)	All available versions
AWS GovCloud (US-East)	Not available
AWS GovCloud (US-West)	Not available

Engine-native features in Amazon RDS

Amazon RDS database engines also support many of the most common engine-native features and functionality. These features are different than the Amazon RDS-native features listed on this page. Some engine-native features might have limited support or restricted privileges.

For more information on engine-native features, see:

- Amazon RDS for Db2 features
- MariaDB feature support on Amazon RDS
- MySQL feature support on Amazon RDS
- RDS for Oracle features
- Working with PostgreSQL features supported by Amazon RDS for PostgreSQL
- Microsoft SQL Server features on Amazon RDS

DB instance billing for Amazon RDS

Amazon RDS instances are billed based on the following components:

- DB instance hours (per hour) – Based on the DB instance class of the DB instance (for example, db.t2.small or db.m4.large). Pricing is listed on a per-hour basis, but bills are calculated down to the second and show times in decimal form. RDS usage is billed in 1-second increments, with a minimum of 10 minutes. For more information, see DB instance classes.
- Storage (per GiB per month) – Storage capacity that you have provisioned to your DB instance. If you scale your provisioned storage capacity within the month, your bill is prorated. For more information, see Amazon RDS DB instance storage.
- Input/output (I/O) requests (per 1 million requests) – Total number of storage I/O requests that you have made in a billing cycle, for Amazon RDS magnetic storage only.
- Provisioned IOPS (per IOPS per month) – Provisioned IOPS rate, regardless of IOPS consumed, for Amazon RDS Provisioned IOPS (SSD) and General Purpose (SSD) gp3 storage. Provisioned storage for EBS volumes are billed in 1-second increments, with a minimum of 10 minutes.
- Backup storage (per GiB per month) – *Backup storage* is the storage that is associated with automated database backups and any active database snapshots that you have taken. Increasing your backup retention period or taking additional database snapshots increases the backup storage consumed by your database. Per second billing doesn't apply to backup storage (metered in GB-month).

  For more information, see Backing up, restoring, and exporting data.
- Data transfer (per GB) – Data transfer in and out of your DB instance from or to the internet and other AWS Regions. For useful examples, see the AWS blog post Exploring Data Transfer Costs for AWS Managed Databases.

Amazon RDS provides the following purchasing options to enable you to optimize your costs based on your needs:

- **On-Demand instances** – Pay by the hour for the DB instance hours that you use. Pricing is listed on a per-hour basis, but bills are calculated down to the second and show times in decimal form. RDS usage is now billed in 1-second increments, with a minimum of 10 minutes.
- **Reserved instances** – Reserve a DB instance for a one-year or three-year term and get a significant discount compared to the on-demand DB instance pricing. With Reserved Instance

usage, you can launch, delete, start, or stop multiple instances within an hour and get the Reserved Instance benefit for all of the instances.

For Amazon RDS pricing information, see the Amazon RDS pricing page.

Topics

- On-Demand DB instances for Amazon RDS
- Reserved DB instances for Amazon RDS

On-Demand DB instances for Amazon RDS

Amazon RDS on-demand DB instances are billed based on the class of the DB instance (for example, db.t3.small or db.m5.large). For Amazon RDS pricing information, see the Amazon RDS product page.

Billing starts for a DB instance as soon as the DB instance is available. Pricing is listed on a per-hour basis, but bills are calculated down to the second and show times in decimal form. Amazon RDS usage is billed in one-second increments, with a minimum of 10 minutes. In the case of billable configuration change, such as scaling compute or storage capacity, you're charged a 10-minute minimum. Billing continues until the DB instance terminates, which occurs when you delete the DB instance or if the DB instance fails.

If you no longer want to be charged for your DB instance, you must stop or delete it to avoid being billed for additional DB instance hours. For more information about the DB instance states for which you are billed, see Viewing Amazon RDSDB instance status.

Stopped DB instances

While your DB instance is stopped, you're charged for provisioned storage, including Provisioned IOPS. You are also charged for backup storage, including storage for manual snapshots and automated backups within your specified retention window. You aren't charged for DB instance hours.

Multi-AZ DB instances

A Multi-AZ setup enhances data durability and availability by automatically provisioning and maintaining a synchronous standby replica in a different Availability Zone. Due to the additional resources and increased availability, Multi-AZ deployments are priced higher than Single-AZ deployments, and can cost approximately twice as much due to the additional standby instance and associated resources.

Consider the following important details about Multi-AZ pricing:

- **Compute costs**: Billed per DB instance-hour for both the primary and standby instances.
- **Storage costs**: Charged per GB-month for the storage provisioned for both the primary and standby instances.
- **Data transfer costs**: Replication between the primary and standby instances is included in the cost, but other data transfer charges might apply based on your usage.

To accurately estimate your monthly costs based on your specific use case and AWS Region, you can use the AWS Pricing Calculator. This tool lets you to input your configuration details and provides a comprehensive cost breakdown.

> **Note**
>
> Pricing is subject to change. See the Amazon RDS Pricing page for the most up-to-date information.

Reserved DB instances for Amazon RDS

Using reserved DB instances, you can reserve a DB instance for a one- or three-year term. Reserved DB instances provide you with a significant discount compared to on-demand DB instance pricing. Reserved DB instances are not physical instances, but rather a billing discount applied to the use of certain on-demand DB instances in your account. Discounts for reserved DB instances are tied to instance type and AWS Region.

The general process for working with reserved DB instances is: First get information about available reserved DB instance offerings, then purchase a reserved DB instance offering, and finally get information about your existing reserved DB instances.

For information about purchasing reserved DB instances and viewing the billing for reserved DB instances, see the following sections.

- Purchasing reserved DB instances for Amazon RDS
- Viewing the billing for reserved DB instances for Amazon RDS

Overview of reserved DB instances

When you purchase a reserved DB instance in Amazon RDS, you purchase a commitment to getting a discounted rate, on a specific DB instance type, for the duration of the reserved DB instance. To use an Amazon RDS reserved DB instance, you create a new DB instance just like you do for an on-demand instance.

The new DB instance that you create must have the same specifications as the reserved DB instance for the following:

- AWS Region
- DB engine (The DB engine's version number doesn't need to match.)
- DB instance type
- DB instance size (RDS for Db2, RDS for SQL Server, and RDS for Oracle License Included)
- Edition (RDS for Db2, RDS for SQL Server, and RDS for Oracle)
- License type (license-included or bring-your-own-license)

If the specifications of the new DB instance match an existing reserved DB instance for your account, you are billed at the discounted rate offered for the reserved DB instance. Otherwise, the DB instance is billed at an on-demand rate.

You can modify a DB instance that you're using as a reserved DB instance. If the modification is within the specifications of the reserved DB instance, part or all of the discount still applies to the modified DB instance. If the modification is outside the specifications, such as changing the instance class, the discount no longer applies. For more information, see Size-flexible reserved DB instances.

Topics

- Offering types
- Size-flexible reserved DB instances
- Reserved DB instance billing example
- Reserved DB instances for a Multi-AZ DB cluster
- Deleting a reserved DB instance

For more information about reserved DB instances, including pricing, see Amazon RDS reserved instances.

Offering types

Reserved DB instances are available in three varieties—No Upfront, Partial Upfront, and All Upfront—that let you optimize your Amazon RDS costs based on your expected usage.

> **Note**
>
> Not all RDS instance classes support all Reserved Instance offering types. For example, some instance classes might not offer the No Upfront option. To confirm availability, review the Reserved Instance offerings in the AWS Management Console or use the `describe-reserved-db-instances-offerings` AWS CLI command.

No Upfront

This option provides access to a reserved DB instance without requiring an upfront payment. Your No Upfront reserved DB instance bills a discounted hourly rate for every hour within the

term, regardless of usage, and no upfront payment is required. This option is only available as a one-year reservation.

Partial Upfront

This option requires a part of the reserved DB instance to be paid upfront. The remaining hours in the term are billed at a discounted hourly rate, regardless of usage. This option is the replacement for the previous Heavy Utilization option.

All Upfront

Full payment is made at the start of the term, with no other costs incurred for the remainder of the term regardless of the number of hours used.

If you are using consolidated billing, all the accounts in the organization are treated as one account. This means that all accounts in the organization can receive the hourly cost benefit of reserved DB instances that are purchased by any other account. For more information about consolidated billing, see Amazon RDS reserved DB instances in the *AWS Billing and Cost Management User Guide*.

Size-flexible reserved DB instances

When you purchase a reserved DB instance, one thing that you specify is the instance class, for example db.r5.large. For more information about DB instance classes, see DB instance classes.

If you have a DB instance, and you need to scale it to larger capacity, your reserved DB instance is automatically applied to your scaled DB instance. That is, your reserved DB instances are automatically applied across all DB instance class sizes. Size-flexible reserved DB instances are available for DB instances with the same AWS Region and database engine. Size-flexible reserved DB instances can only scale in their instance class type. For example, a reserved DB instance for a db.r6i.large can apply to a db.r6i.xlarge, but not to a db.r6id.large or db.r7g.large, because db.r6id.large and db.r7g.large are different instance class types.

Reserved DB instance benefits apply to both Multi-AZ and Single-AZ configurations. This means that you can move freely between configurations within the same DB instance class type. For example, you can move from a Single-AZ deployment running on one large DB instance (four normalized units per hour) to a Multi-AZ deployment running on two medium DB instances (2+2 = 4 normalized units per hour).

Size-flexible reserved DB instances are available for the following Amazon RDS database engines:

- RDS for Db2
- RDS for MariaDB
- RDS for MySQL
- RDS for Oracle, Bring Your Own License
- RDS for PostgreSQL

Size flexibility does not apply to RDS for SQL Server and RDS for Oracle License Included.

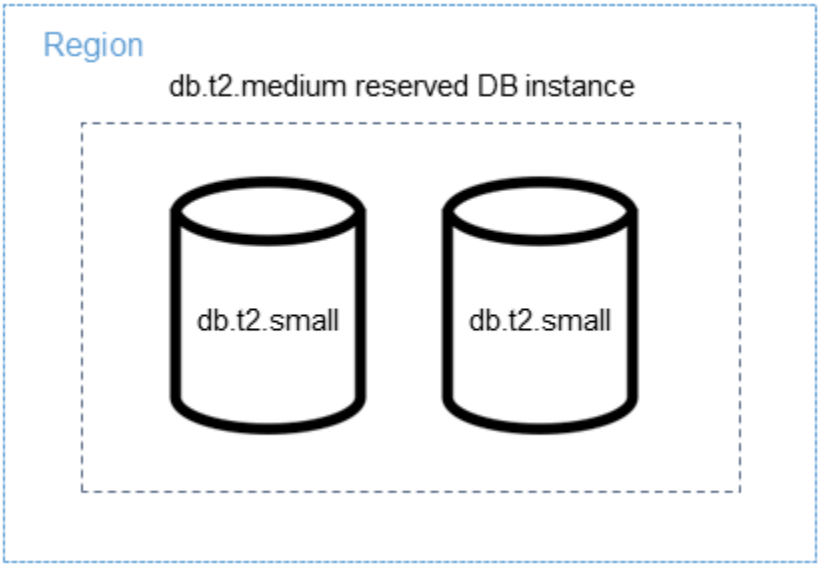
For details about using size-flexible reserved instances with Aurora, see Reserved DB instances for Aurora.

You can compare usage for different reserved DB instance sizes by using normalized units per hour. For example, one unit of usage on two db.r3.large DB instances is equivalent to eight normalized units per hour of usage on one db.r3.small. The following table shows the number of normalized units per hour for each DB instance size.

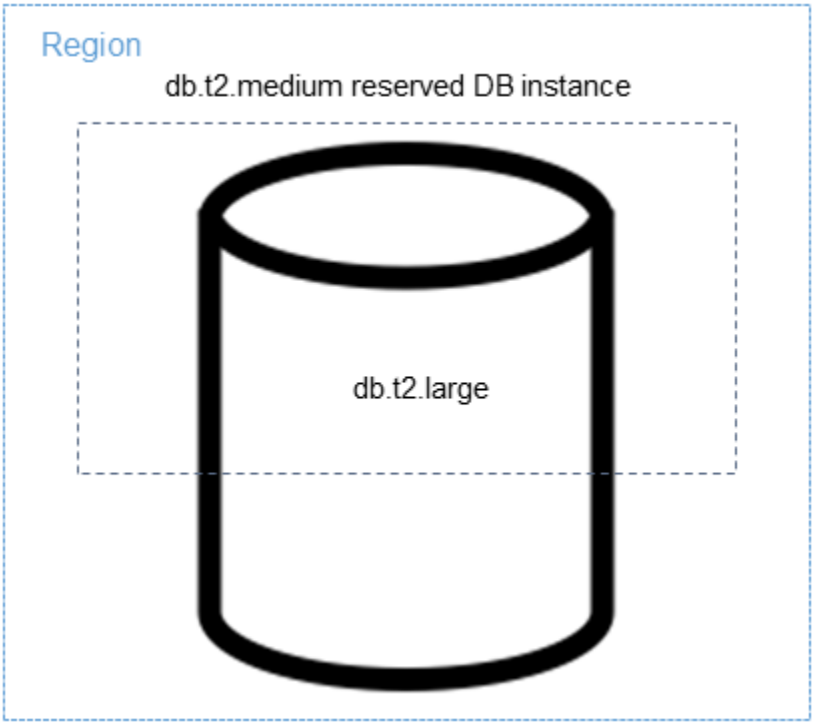
Instance size	Single-AZ normalized units per hour (deployment with one DB instance)	Multi-AZ DB instance normalized units per hour (deployment with one DB instance and one standby)	Multi-AZ DB cluster normalized units per hour (deployment with one DB instance and two standbys)
micro	0.5	1	1.5
small	1	2	3
medium	2	4	6
large	4	8	12
xlarge	8	16	24
2xlarge	16	32	48
4xlarge	32	64	96
6xlarge	48	96	144
8xlarge	64	128	192

Instance size	Single-AZ normalized units per hour (deployment with one DB instance)	Multi-AZ DB instance normalized units per hour (deployment with one DB instance and one standby)	Multi-AZ DB cluster normalized units per hour (deployment with one DB instance and two standbys)
10xlarge	80	160	240
12xlarge	96	192	288
16xlarge	128	256	384
24xlarge	192	384	576
32xlarge	256	512	768

For example, suppose that you purchase a `db.t2.medium` reserved DB instance, and you have two running `db.t2.small` DB instances in your account in the same AWS Region. In this case, the billing benefit is applied in full to both instances.



Alternatively, if you have one `db.t2.large` instance running in your account in the same AWS Region, the billing benefit is applied to 50 percent of the usage of the DB instance.



Reserved DB instance billing example

The price for a reserved DB instance doesn't provide a discount for the costs associated with storage, backups, and I/O. It provides a discount only on the hourly, on-demand instance usage. The following example illustrates the total cost per month for a reserved DB instance:

- An RDS for MySQL reserved Single-AZ db.r5.large DB instance class in US East (N. Virginia) with the No Upfront option at a cost of $0.12 for the instance, or $90 per month
- 400 GiB of General Purpose SSD (gp2) storage at a cost of 0.115 per GiB per month, or $45.60 per month
- 600 GiB of backup storage at $0.095, or $19 per month (400 GiB free)

Add all of these charges ($90 + $45.60 + $19) with the reserved DB instance, and the total cost per month is $154.60.

If you choose to use an on-demand DB instance instead of a reserved DB instance, an RDS for MySQL Single-AZ db.r5.large DB instance class in US East (N. Virginia) costs $0.1386 per hour, or $101.18 per month. So, for an on-demand DB instance, add all of these options ($101.18 + $45.60 + $19), and the total cost per month is $165.78. You save a little over $11 per month by using the reserved DB instance.

> **Note**
>
> The prices in this example are sample prices and might not match actual prices. For Amazon RDS pricing information, see Amazon RDS pricing.

Reserved DB instances for a Multi-AZ DB cluster

To purchase the equivalent reserved DB instances for a Multi-AZ DB cluster, you can do one of the following:

- Reserve three Single-AZ DB instances that are the same size as the instances in the cluster.
- Reserve one Multi-AZ DB instance and one Single-AZ DB instance that are the same size as the DB instances in the cluster.

For example, suppose that you have one cluster consisting of three db.m6gd.large DB instances. In this case, you can either purchase three db.m6gd.large Single-AZ reserved DB instances, or one db.m6gd.large Multi-AZ reserved DB instance and one db.m6gd.large Single-AZ reserved DB instance. Either of these options reserves the maximum reserved instance discount for the Multi-AZ DB cluster.

Alternately, you can use size-flexible DB instances and purchase a larger DB instance to cover smaller DB instances in one or more clusters. For example, if you have two clusters with six total db.m6gd.large DB instances, you can purchase three db.m6gd.xl Single-AZ reserved DB instances. Doing so reserves all six DB instances in the two clusters. For more information, see Size-flexible reserved DB instances.

You might reserve DB instances that are the same size as the DB instances in the cluster, but reserve fewer DB instances than the total number of DB instances in the cluster. However, if you do so, the cluster is only partially reserved. For example, suppose that you have one cluster with three db.m6gd.large DB instances, and you purchase one db.m6gd.large Multi-AZ reserved DB instance. In this case, the cluster is only partially reserved, because only two of the three instances in the cluster are covered by reserved DB instances. The remaining DB instance is charged at the on-demand db.m6gd.large hourly rate.

For more information about Multi-AZ DB clusters, see Multi-AZ DB cluster deployments for Amazon RDS.

Deleting a reserved DB instance

The terms for a reserved DB instance involve a one-year or three-year commitment. You can't cancel a reserved DB instance. However, you can delete a DB instance that is covered by a reserved DB instance discount. The process for deleting a DB instance that is covered by a reserved DB instance discount is the same as for any other DB instance.

You're billed for the upfront costs regardless of whether you use the resources.

If you delete a DB instance that is covered by a reserved DB instance discount, you can launch another DB instance with compatible specifications. In this case, you continue to get the discounted rate during the reservation term (one or three years).

Purchasing reserved DB instances for Amazon RDS

You can use the AWS Management Console, the AWS CLI, and the RDS API to work with reserved DB instances.

Console

You can use the AWS Management Console to work with reserved DB instances as shown in the following procedures.

To get pricing and information about available reserved DB instance offerings

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Reserved instances**.
3. Choose **Purchase Reserved DB Instance**.
4. For **Product description**, choose the DB engine and licensing type.
5. For **DB instance class**, choose the DB instance class.
6. For **Deployment Option**, choose whether you want a Single-AZ or Multi-AZ DB instance deployment.

> **Note**
>
> To purchase the equivalent reserved DB instances for a Multi-AZ DB cluster deployment, either purchase three Single-AZ reserved DB instances, or one Multi-

AZ and one Single-AZ reserved DB instance. For more information, see Reserved DB instances for a Multi-AZ DB cluster.

7. For **Term**, choose the length of time to reserve the the DB instance.
8. For **Offering type**, choose the offering type.

   After you select the offering type, you can see the pricing information.

   Important

   Choose **Cancel** to avoid purchasing the reserved DB instance and incurring any charges.

After you have information about the available reserved DB instance offerings, you can use the information to purchase an offering as shown in the following procedure.

To purchase a reserved DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Reserved instances**.
3. **Important**

   Before proceeding, verify that you are in the correct AWS Region. Reserved DB instances are Region-specific and cannot be transferred between Regions. Check the Region selector in the upper-right corner of the console to ensure you are purchasing the reserved instance in the intended Region.
4. Choose **Purchase reserved DB instance**.
5. For **Product description**, choose the DB engine and licensing type.
6. For **DB instance class**, choose the DB instance class.
7. For **Multi-AZ deployment**, choose whether you want a Single-AZ or Multi-AZ DB instance deployment.

> **Note**
>
> To purchase the equivalent reserved DB instances for a Multi-AZ DB cluster deployment, either purchase three Single-AZ reserved DB instances, or one Multi-AZ and one Single-AZ reserved DB instance. For more information, see Reserved DB instances for a Multi-AZ DB cluster.

8. For **Term**, choose the length of time you want the DB instance reserved.
9. For **Offering type**, choose the offering type.

   After you choose the offering type, you can see the pricing information.
   Purchase reserved DB instance console
10. (Optional) You can assign your own identifier to the reserved DB instances that you purchase to help you track them. For **Reserved Id**, type an identifier for your reserved DB instance.
11. Choose **Submit**.

    Your reserved DB instance is purchased, then displayed in the **Reserved instances** list.

After you have purchased reserved DB instances, you can get information about your reserved DB instances as shown in the following procedure.

To get information about reserved DB instances for your AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the **Navigation** pane, choose **Reserved instances**.

   The reserved DB instances for your account appear. To see detailed information about a particular reserved DB instance, choose that instance in the list. You can then see detailed information about that instance in the detail pane at the bottom of the console.

AWS CLI

You can use the AWS CLI to work with reserved DB instances as shown in the following examples.

Example of getting available reserved DB instance offerings

To get information about available reserved DB instance offerings, call the AWS CLI command `describe-reserved-db-instances-offerings`.

```
aws rds describe-reserved-db-instances-offerings
```

This call returns output similar to the following:

```
OFFERING  OfferingId                            Class         Multi-AZ  Duration  Fixed
 Price  Usage Price  Description  Offering Type
OFFERING  438012d3-4052-4cc7-b2e3-8d3372e0e706  db.r3.large   y         1y
 1820.00 USD  0.368 USD    mysql        Partial  Upfront
OFFERING  649fd0c8-cf6d-47a0-bfa6-060f8e75e95f  db.r3.small   n         1y
 227.50 USD  0.046 USD    mysql        Partial  Upfront
OFFERING  123456cd-ab1c-47a0-bfa6-12345667232f  db.r3.small   n         1y
 162.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   0.123   USD       Hourly
OFFERING  123456cd-ab1c-37a0-bfa6-12345667232d  db.r3.large   y         1y
 700.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   1.25    USD       Hourly
OFFERING  123456cd-ab1c-17d0-bfa6-12345667234e  db.r3.xlarge  n         1y
 4242.00 USD   2.42 USD    mysql        No       Upfront
```

After you have information about the available reserved DB instance offerings, you can use the information to purchase an offering.

To purchase a reserved DB instance, use the AWS CLI command `purchase-reserved-db-instances-offering` with the following parameters:

- `--reserved-db-instances-offering-id` – The ID of the offering that you want to purchase. See the preceding example to get the offering ID.
- `--reserved-db-instance-id` – You can assign your own identifier to the reserved DB instances that you purchase to help track them.

Example of purchasing a reserved DB instance

The following example purchases the reserved DB instance offering with ID *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f*, and assigns the identifier of *MyReservation*.

For Linux, macOS, or Unix:

```
aws rds purchase-reserved-db-instances-offering \
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
    --reserved-db-instance-id MyReservation
```

For Windows:

```
aws rds purchase-reserved-db-instances-offering ^
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
    --reserved-db-instance-id MyReservation
```

The command returns output similar to the following:

```
RESERVATION  ReservationId      Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State            Description  Offering Type
RESERVATION  MyReservation      db.r3.small  y         2011-12-19T00:30:23.247Z  1y
    455.00 USD   0.092 USD    1      payment-pending  mysql        Partial  Upfront
```

After you have purchased reserved DB instances, you can get information about your reserved DB instances.

To get information about reserved DB instances for your AWS account, call the AWS CLI command `describe-reserved-db-instances`, as shown in the following example.

Example of getting your reserved DB instances

```
aws rds describe-reserved-db-instances
```

The command returns output similar to the following:

```
RESERVATION  ReservationId     Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State    Description  Offering Type
RESERVATION  MyReservation     db.r3.small  y         2011-12-09T23:37:44.720Z  1y
   455.00 USD   0.092 USD    1      retired  mysql        Partial  Upfront
```

RDS API

You can use the RDS API to work with reserved DB instances:

- To get information about available reserved DB instance offerings, call the Amazon RDS API operation `DescribeReservedDBInstancesOfferings`.
- After you have information about the available reserved DB instance offerings, you can use the information to purchase an offering. Call the `PurchaseReservedDBInstancesOffering` RDS API operation with the following parameters:
  - `--reserved-db-instances-offering-id` – The ID of the offering that you want to purchase.
  - `--reserved-db-instance-id` – You can assign your own identifier to the reserved DB instances that you purchase to help track them.
- After you have purchased reserved DB instances, you can get information about your reserved DB instances. Call the `DescribeReservedDBInstances` RDS API operation.

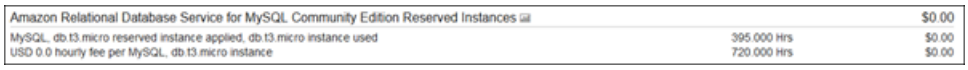
Viewing the billing for reserved DB instances for Amazon RDS

You can view the billing for your reserved DB instances in the Billing Dashboard in the AWS Management Console.

To view reserved DB instance billing

1. Sign in to the AWS Management Console.
2. From the **account menu** at the upper right, choose **Billing Dashboard**.
3. Choose **Bill Details** at the upper right of the dashboard.
4. Under **AWS Service Charges**, expand **Relational Database Service**.
5. Expand the AWS Region where your reserved DB instances are, for example **US West (Oregon)**.

   Your reserved DB instances and their hourly charges for the current month are shown under **Amazon Relational Database Service for *`Database Engine`* Reserved Instances**.

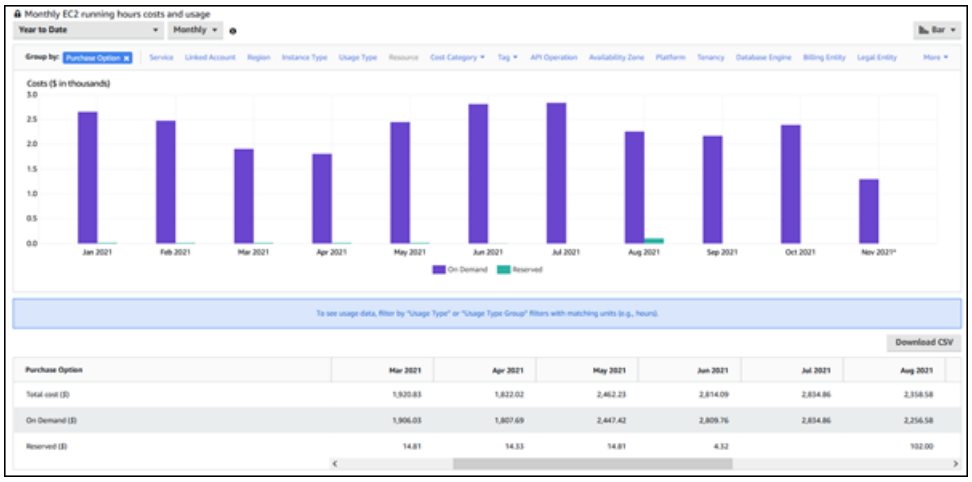
   

   The reserved DB instance in this example was purchased All Upfront, so there are no hourly charges.
6. Choose the **Cost Explorer** (bar graph) icon next to the **Reserved Instances** heading.

The Cost Explorer displays the **Monthly EC2 running hours costs and usage** graph.

7. Clear the **Usage Type Group** filter to the right of the graph.
8. Choose the time period and time unit for which you want to examine usage costs.

   The following example shows usage costs for on-demand and reserved DB instances for the year to date by month.



The reserved DB instance costs from January through June 2021 are monthly charges for a Partial Upfront instance, while the cost in August 2021 is a one-time charge for an All Upfront instance.

The reserved instance discount for the Partial Upfront instance expired in June 2021, but the DB instance wasn't deleted. After the expiration date, it was simply charged at the on-demand rate.

Setting up your Amazon RDS environment

This page provides a comprehensive guide for setting up Amazon Relational Database Service, including account configuration, security, and resource management. It walks you through the essential steps to create, manage, and secure your database environments efficiently. Whether you're new to Amazon RDS or setting up for specific requirements, these sections help ensure your setup is optimized and compliant with best practices.

Topics

- Sign up for an AWS account
- Create a user with administrative access
- Grant programmatic access
- Determine requirements
- Provide access to your DB instance in your VPC by creating a security group

If you already have an AWS account, know your Amazon RDS requirements, and prefer to use the defaults for IAM and VPC security groups, skip ahead to Getting started with Amazon RDS.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.

Which user needs programmatic access?	To	By
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Determine requirements

The basic building block of Amazon RDS is the DB instance. In a DB instance, you create your databases. A DB instance provides a network address called an *endpoint*. Your applications use this endpoint to connect to your DB instance. When you create a DB instance, you specify details like storage, memory, database engine and version, network configuration, security, and maintenance periods. You control network access to a DB instance through a security group.

Before you create a DB instance and a security group, you must know your DB instance and network needs. Here are some important things to consider:

- **Resource requirements** – What are the memory and processor requirements for your application or service? You use these settings to help you determine what DB instance class to use. For specifications about DB instance classes, see DB instance classes.

- **VPC, subnet, and security group –** Your DB instance will most likely be in a virtual private cloud (VPC). To connect to your DB instance, you need to set up security group rules. These rules are set up differently depending on what kind of VPC you use and how you use it. For example, you can use: a default VPC or a user-defined VPC.

  The following list describes the rules for each VPC option:

  - **Default VPC** – If your AWS account has a default VPC in the current AWS Region, that VPC is configured to support DB instances. If you specify the default VPC when you create the DB instance, do the following:
    - Make sure to create a *VPC security group* that authorizes connections from the application or service to the Amazon RDS DB instance. Use the **Security Group** option on the VPC console or the AWS CLI to create VPC security groups. For information, see Step 3: Create a VPC security group.
    - Specify the default DB subnet group. If this is the first DB instance you have created in this AWS Region, Amazon RDS creates the default DB subnet group when it creates the DB instance.
  - **User-defined VPC –** If you want to specify a user-defined VPC when you create a DB instance, be aware of the following:
    - Make sure to create a *VPC security group* that authorizes connections from the application or service to the Amazon RDS DB instance. Use the **Security Group** option on the VPC console or the AWS CLI to create VPC security groups. For information, see Step 3: Create a VPC security group.
    - The VPC must meet certain requirements in order to host DB instances, such as having at least two subnets, each in a separate Availability Zone. For information, see Amazon VPC and Amazon RDS.
    - Make sure to specify a DB subnet group that defines which subnets in that VPC can be used by the DB instance. For information, see the DB subnet group section in Working with a DB instance in a VPC.
- **High availability –** Do you need failover support? On Amazon RDS, a Multi-AZ deployment creates a primary DB instance and a secondary standby DB instance in another Availability Zone for failover support. We recommend Multi-AZ deployments for production workloads to maintain high availability. For development and test purposes, you can use a deployment that isn't Multi-AZ. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

- **IAM policies** – Does your AWS account have policies that grant the permissions needed to perform Amazon RDS operations? If you are connecting to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS operations. For more information, see Identity and access management for Amazon RDS.
- **Open ports** – What TCP/IP port does your database listen on? The firewalls at some companies might block connections to the default port for your database engine. If your company firewall blocks the default port, choose another port for the new DB instance. When you create a DB instance that listens on a port you specify, you can change the port by modifying the DB instance.
- **AWS Region** – What AWS Region do you want your database in? Having your database in close proximity to your application or web service can reduce network latency. For more information, see Regions, Availability Zones, and Local Zones.
- **DB disk subsystem** – What are your storage requirements? Amazon RDS provides three storage types:
  - General Purpose (SSD)
  - Provisioned IOPS (PIOPS)
  - Magnetic (also known as standard storage)

  For more information on Amazon RDS storage, see Amazon RDS DB instance storage.

When you have the information you need to create the security group and the DB instance, continue to the next step.

Provide access to your DB instance in your VPC by creating a security group

VPC security groups provide access to DB instances in a VPC. They act as a firewall for the associated DB instance, controlling both inbound and outbound traffic at the DB instance level. DB instances are created by default with a firewall and a default security group that protect the DB instance.

Before you can connect to your DB instance, you must add rules to a security group that enable you to connect. Use your network and configuration information to create rules to allow access to your DB instance.

For example, suppose that you have an application that accesses a database on your DB instance in a VPC. In this case, you must add a custom TCP rule that specifies the port range and IP addresses that your application uses to access the database. If you have an application on an Amazon EC2 instance, you can use the security group that you set up for the Amazon EC2 instance.

You can configure connectivity between an Amazon EC2 instance a DB instance when you create the DB instance. For more information, see Configure automatic network connectivity with an EC2 instance.

> **Tip**
>
> You can set up network connectivity between an Amazon EC2 instance and a DB instance automatically when you create the DB instance. For more information, see Configure automatic network connectivity with an EC2 instance.

For information about how to connect resources in Amazon Lightsail to your DB instances, see Connect Lightsail resources to AWS services using VPC peering.

For information about common scenarios for accessing a DB instance, see Scenarios for accessing a DB instance in a VPC.

To create a VPC security group

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc.

   > **Note**
   >
   > Make sure you are in the VPC console, not the RDS console.

2. In the upper-right corner of the AWS Management Console, choose the AWS Region where you want to create your VPC security group and DB instance. In the list of Amazon VPC resources for that AWS Region, you should see at least one VPC and several subnets. If you don't, you don't have a default VPC in that AWS Region.
3. In the navigation pane, choose **Security Groups**.
4. Choose **Create security group**.

   The **Create security group** page appears.

5. In **Basic details**, enter the **Security group name** and **Description**. For **VPC**, choose the VPC that you want to create your DB instance in.
6. In **Inbound rules**, choose **Add rule**.
   a. For **Type**, choose **Custom TCP**.
   b. For **Port range**, enter the port value to use for your DB instance.
   c. For **Source**, choose a security group name or type the IP address range (CIDR value) from where you access the DB instance. If you choose **My IP**, this allows access to the DB instance from the IP address detected in your browser.
7. If you need to add more IP addresses or different port ranges, choose **Add rule** and enter the information for the rule.
8. (Optional) In **Outbound rules**, add rules for outbound traffic. By default, all outbound traffic is allowed.
9. Choose **Create security group**.

You can use the VPC security group that you just created as the security group for your DB instance when you create it.

> **Note**
>
> If you use a default VPC, a default subnet group spanning all of the VPC's subnets is created for you. When you create a DB instance, you can select the default VPC and use **default** for **DB Subnet Group**.

After you have completed the setup requirements, you can create a DB instance using your requirements and security group. To do so, follow the instructions in Creating an Amazon RDS DB instance. For information about getting started by creating a DB instance that uses a specific DB engine, see the relevant documentation in the following table.

Database engine	Documentation
MariaDB	Creating and connecting to a MariaDB DB instance
Microsoft SQL Server	Creating and connecting to a Microsoft SQL Server DB instance

Database engine	Documentation
MySQL	Creating and connecting to a MySQL DB instance
Oracle	Creating and connecting to an Oracle DB instance
PostgreSQL	Creating and connecting to a PostgreSQL DB instance

Note

If you can't connect to a DB instance after you create it, see the troubleshooting information in Can't connect to Amazon RDS DB instance.

Getting started with Amazon RDS

In the following examples, you can find how to create and connect to a DB instance using Amazon Relational Database Service (Amazon RDS). You can create a DB instance that uses Db2, MariaDB, MySQL, Microsoft SQL Server, Oracle, or PostgreSQL.

> **Important**
>
> Before you can create or connect to a DB instance, make sure to complete the tasks in Setting up your Amazon RDS environment.

Creating a DB instance and connecting to a database on a DB instance is slightly different for each of the DB engines. Choose one of the following DB engines that you want to use for detailed information on creating and connecting to the DB instance. After you have created and connected to your DB instance, there are instructions to help you delete the DB instance.

Topics

- Creating and connecting to a MariaDB DB instance
- Creating and connecting to a Microsoft SQL Server DB instance
- Creating and connecting to a MySQL DB instance
- Creating and connecting to an Oracle DB instance
- Creating and connecting to a PostgreSQL DB instance
- Tutorial: Create a web server and an Amazon RDS DB instance
- Tutorial: Using a Lambda function to access an Amazon RDS database

Creating and connecting to a MariaDB DB instance

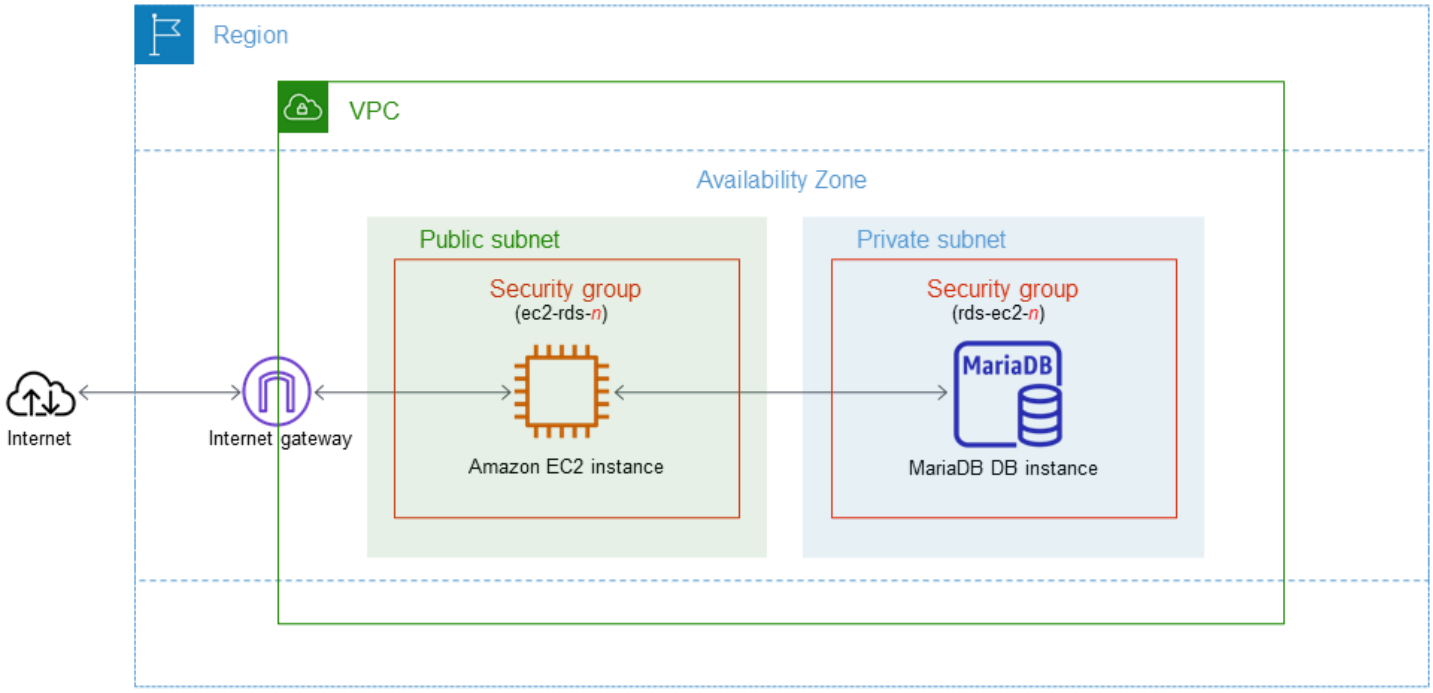
This tutorial creates an EC2 instance and an RDS for MariaDB DB instance. The tutorial shows you how to access the DB instance from the EC2 instance using a standard MySQL client. As a best practice, this tutorial creates a private DB instance in a virtual private cloud (VPC). In most cases, other resources in the same VPC, such as EC2 instances, can access the DB instance, but resources outside of the VPC can't access it.

After you complete the tutorial, there is a public and private subnet in each Availability Zone in your VPC. In one Availability Zone, the EC2 instance is in the public subnet, and the DB instance is in the private subnet.

Important

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the resources you use. You can delete these resources after you complete the tutorial if they are no longer needed.

The following diagram shows the configuration when the tutorial is complete.



This tutorial allows you to create your resources by using one of the following methods:

1. Use the AWS Management Console - Create an EC2 instance and Create a MariaDB DB instance
2. Use CloudFormation to create the database instance and EC2 instance - (Optional) Create VPC, EC2 instance, and MariaDB instance using CloudFormation

The first method uses **Easy create** to create a private MariaDB DB instance with the AWS Management Console. Here, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default settings for the other configuration options.

When you use **Standard create** instead, you can specify more configuration options when you create a DB instance. These options include settings for availability, security, backups, and maintenance. To create a public DB instance, you must use **Standard create**. For information, see Creating an Amazon RDS DB instance.

Topics

- Prerequisites
- Create an EC2 instance
- Create a MariaDB DB instance
- (Optional) Create VPC, EC2 instance, and MariaDB instance using CloudFormation
- Connect to a MariaDB DB instance
- Delete the EC2 instance and DB instance
- (Optional) Delete the EC2 instance and DB instance created with CloudFormation
- (Optional) Connect your DB instance to a Lambda function

Prerequisites

Before you begin, complete the steps in the following sections:

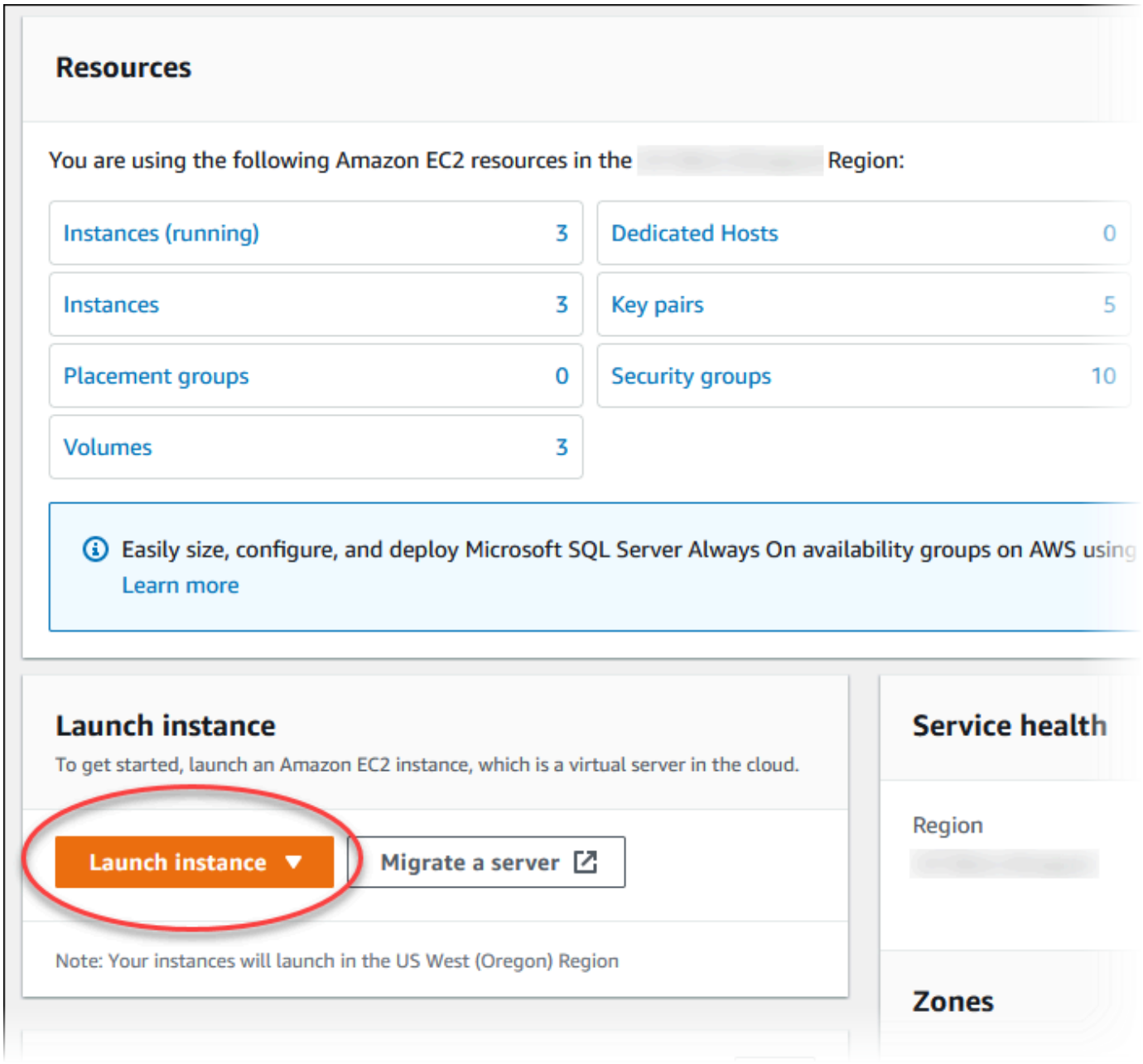
- Sign up for an AWS account
- Create a user with administrative access

Create an EC2 instance

Create an Amazon EC2 instance that you will use to connect to your database.

To create an EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you want to create the EC2 instance.
3. Choose **EC2 Dashboard**, and then choose **Launch instance**, as shown in the following image.

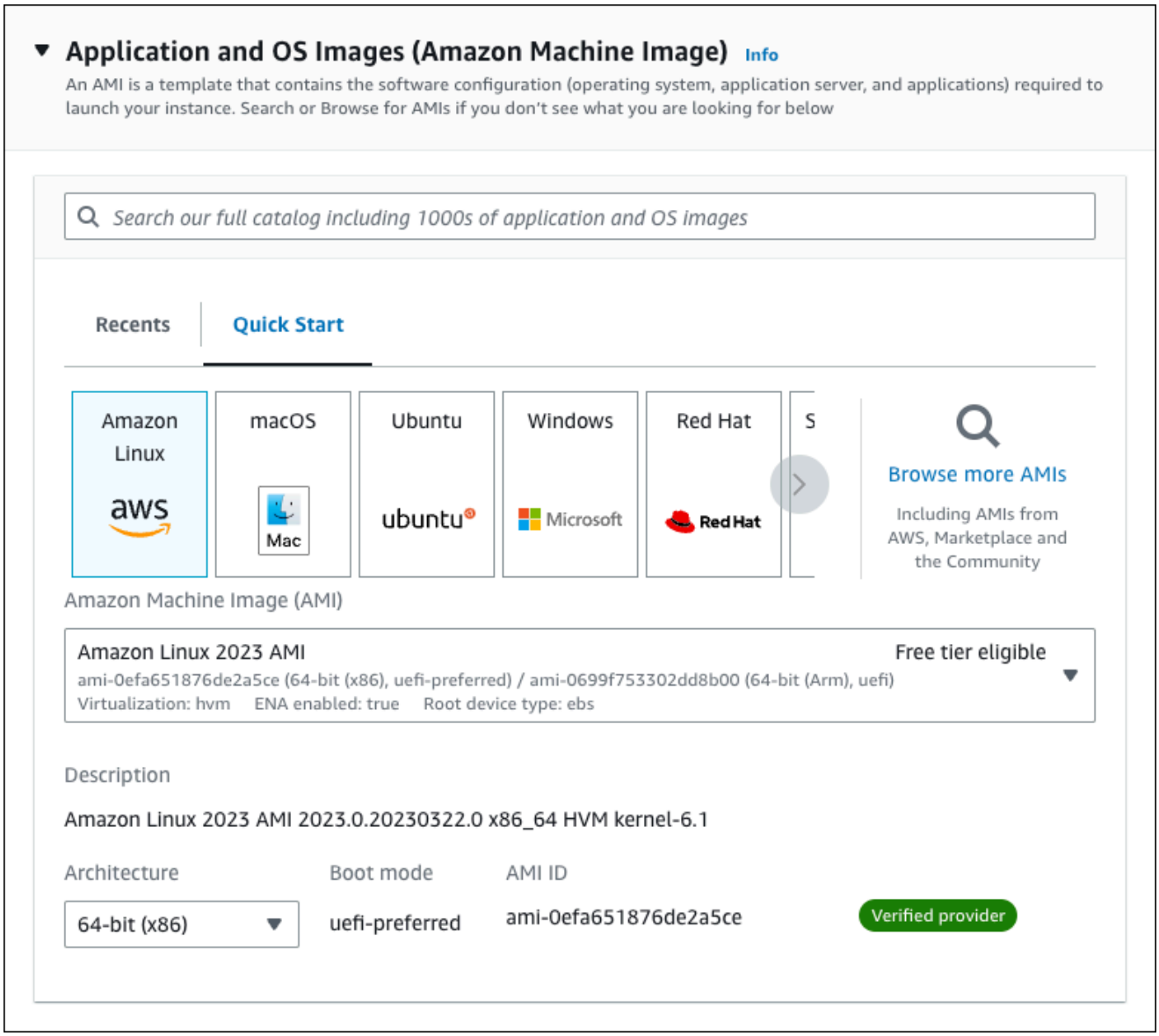


The **Launch an instance** page opens.

4. Choose the following settings on the **Launch an instance** page.

   a. Under **Name and tags**, for **Name**, enter **`ec2-database-connect`**.

   b. Under **Application and OS Images (Amazon Machine Image)**, choose **Amazon Linux**, and then choose the **Amazon Linux 2023 AMI**. Keep the default selections for the other choices.



   c. Under **Instance type**, choose **t2.micro**.

   d. Under **Key pair (login)**, choose a **Key pair name** to use an existing key pair. To create a new key pair for the Amazon EC2 instance, choose **Create new key pair** and then use the **Create key pair** window to create it.

For more information about creating a new key pair, see Create a key pair in the *Amazon EC2 User Guide*.

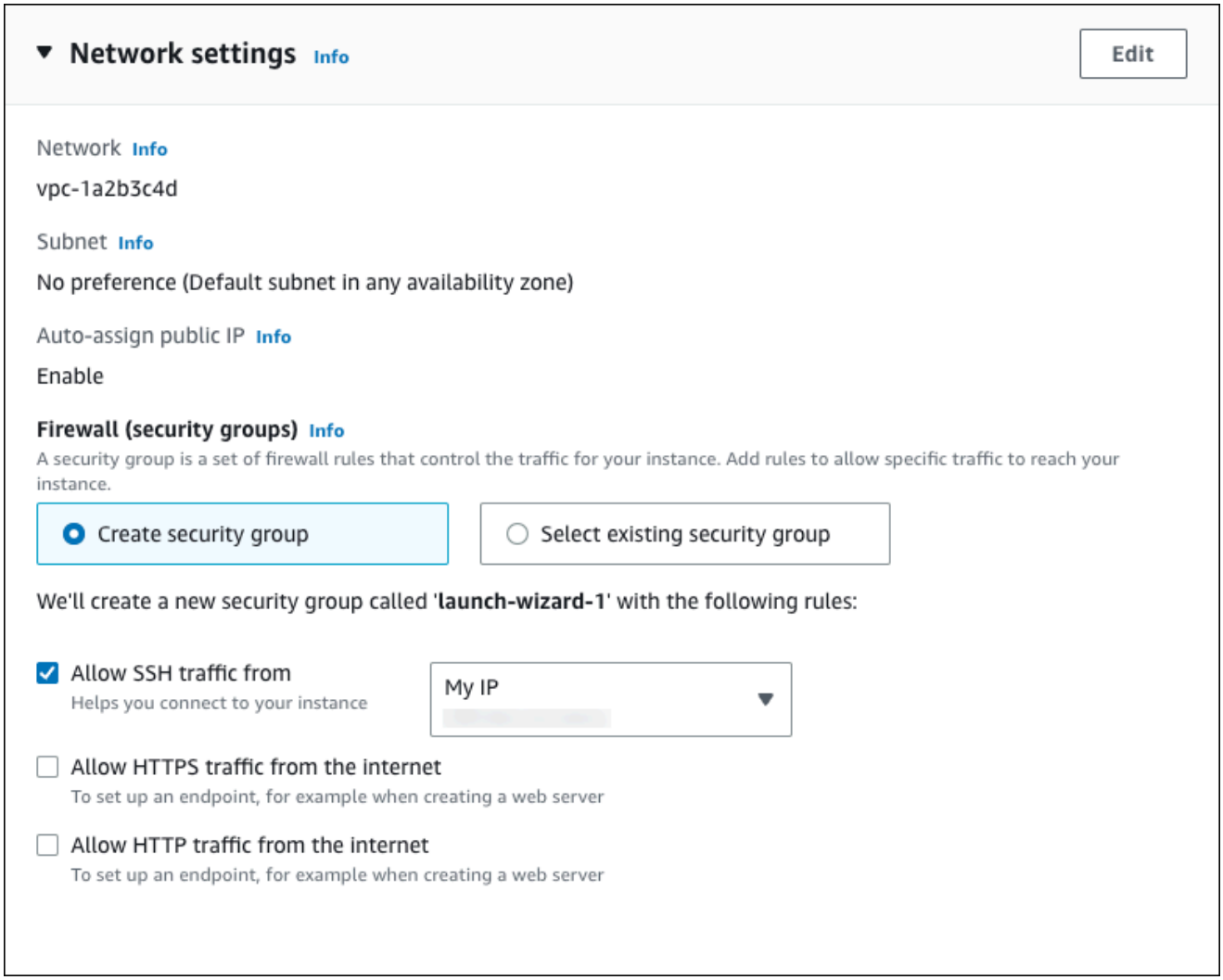
e. For **Allow SSH traffic** in **Network settings**, choose the source of SSH connections to the EC2 instance.

You can choose **My IP** if the displayed IP address is correct for SSH connections. Otherwise, you can determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH). To determine your public IP address, in a different browser window or tab, you can use the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, make sure to determine the range of IP addresses used by client computers.

> **Warning**
>
> If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

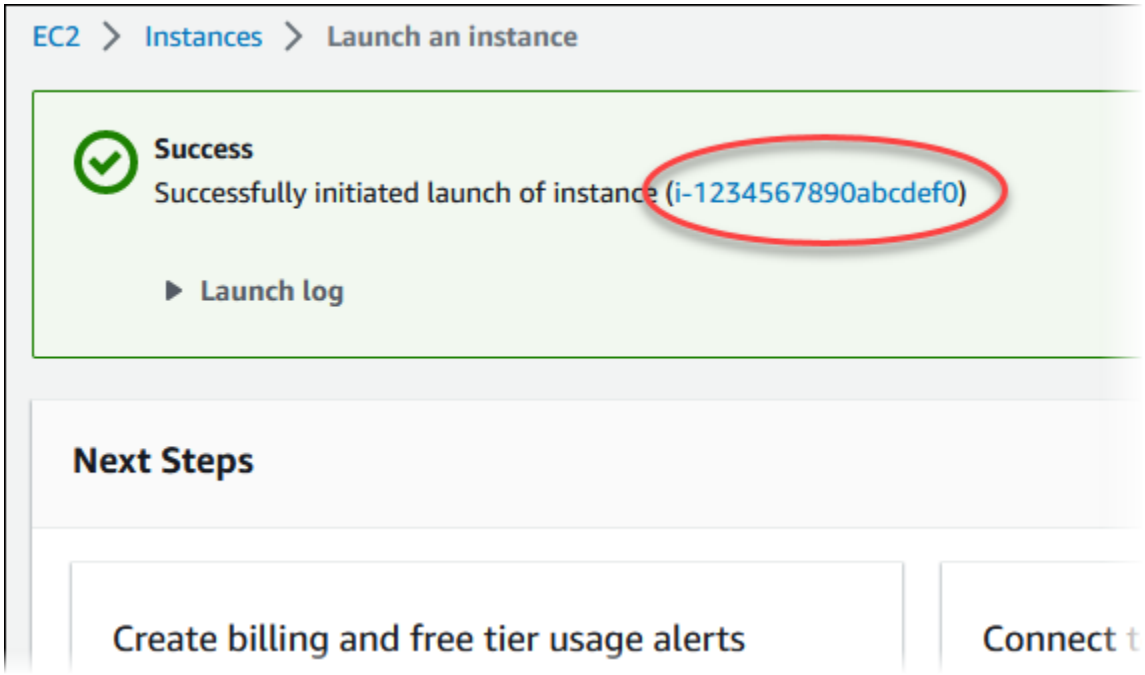
The following image shows an example of the **Network settings** section.



f. Leave the default values for the remaining sections.

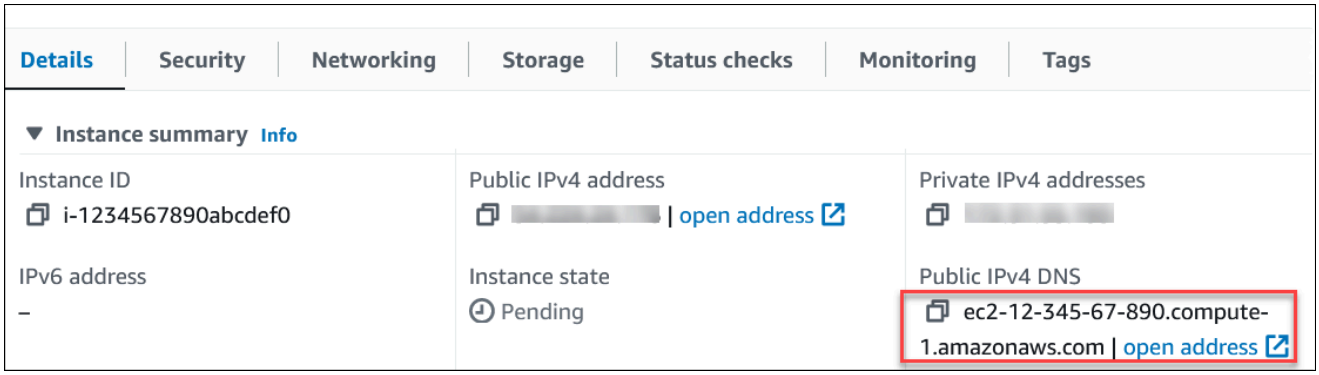
g. Review a summary of your EC2 instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

5. On the **Launch Status** page, note the identifier for your new EC2 instance, for example: `i-1234567890abcdef0`.

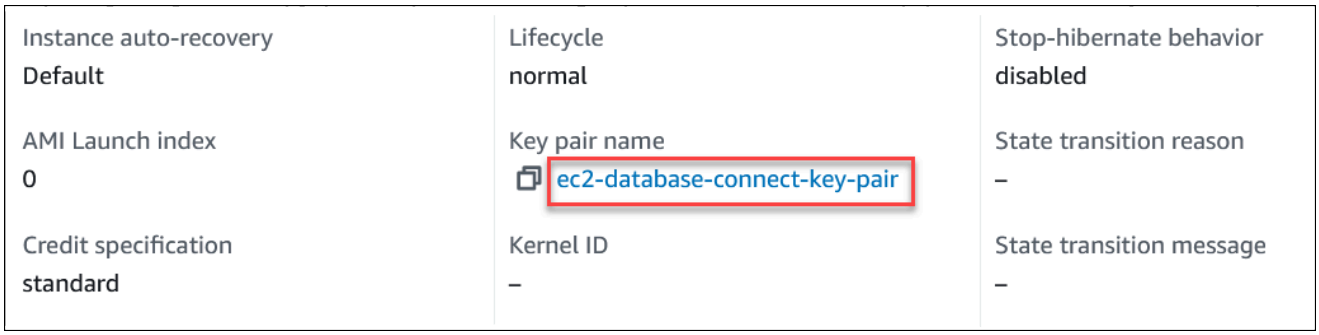


6. Choose the EC2 instance identifier to open the list of EC2 instances, and then select your EC2 instance.
7. In the **Details** tab, note the following values, which you need when you connect using SSH:

   a. In **Instance summary**, note the value for **Public IPv4 DNS**.



   b. In **Instance details**, note the value for **Key pair name**.



8. Wait until the **Instance state** for your EC2 instance has a status of **Running** before continuing.

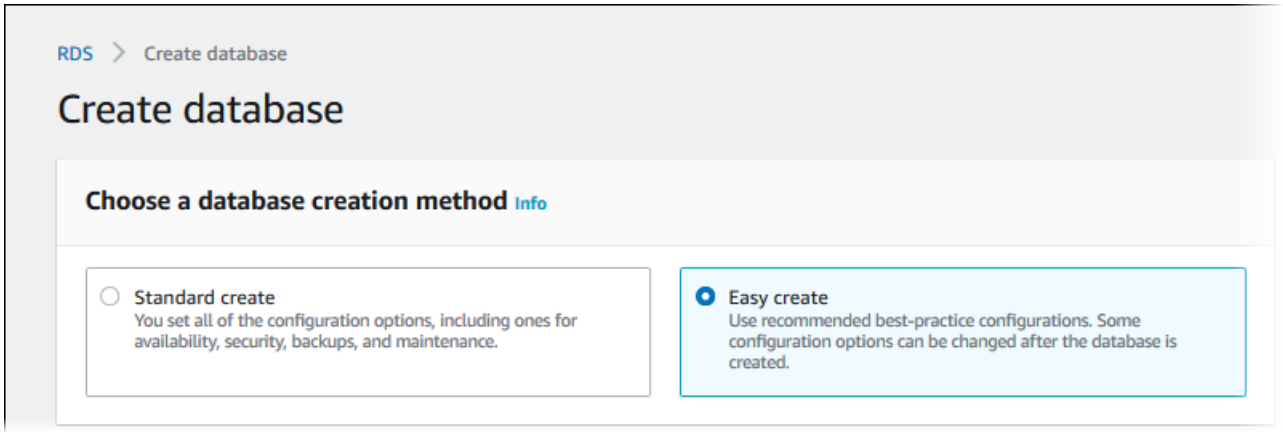
Create a MariaDB DB instance

The basic building block of Amazon RDS is the DB instance. This environment is where you run your MariaDB databases.

In this example, you use **Easy create** to create a DB instance running the MariaDB database engine with a db.t4g.micro DB instance class.

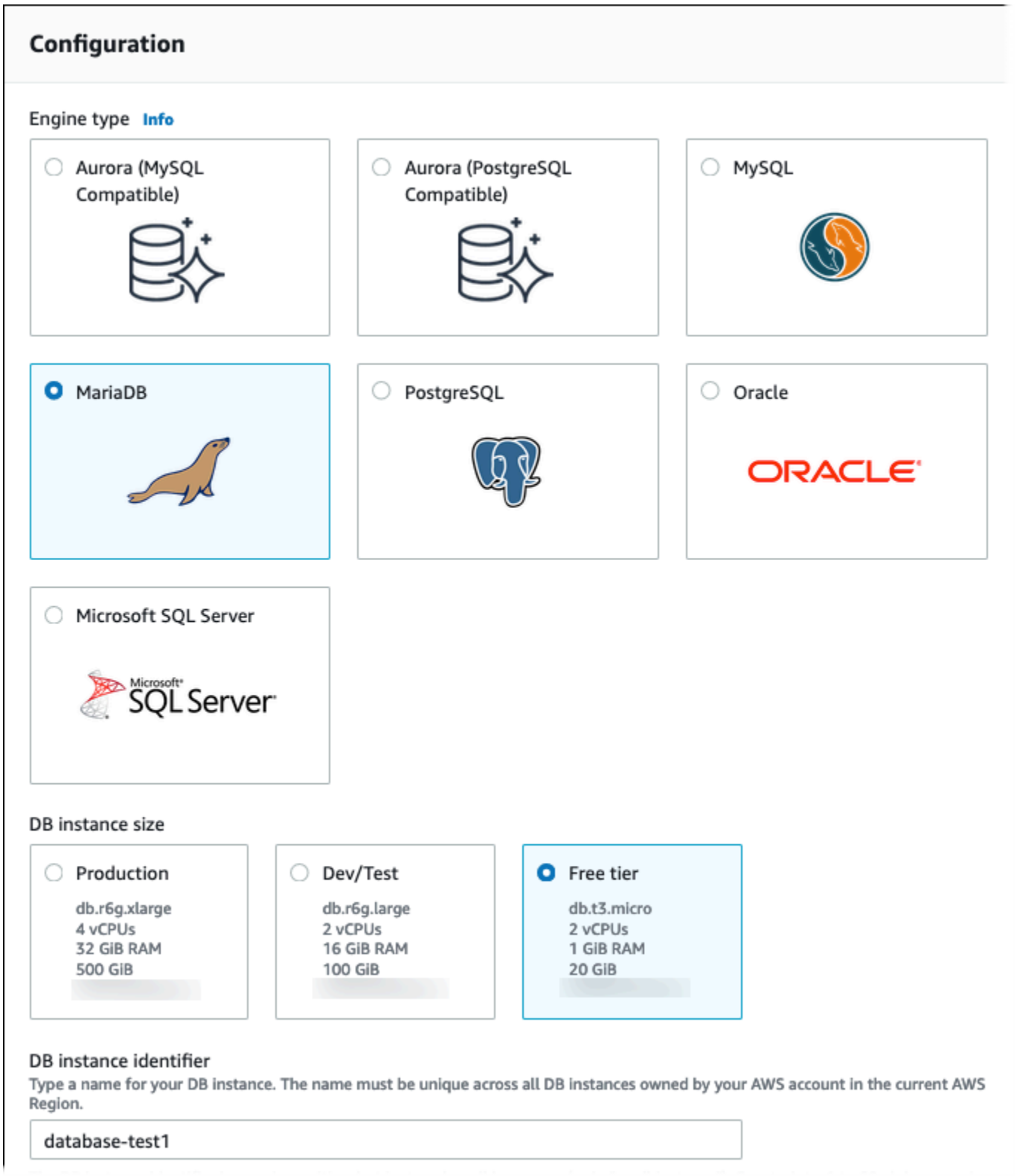
To create a MariaDB DB instance with Easy create

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database** and select **Easy create**.



5. In **Configuration**, choose **MariaDB**.
6. For **DB instance size**, choose **Free tier** or **Sandbox**. **Free tier** appears for free plan accounts. **Sandbox** appears for paid plan accounts.
7. For **DB instance identifier**, enter **`database-test1`**.
8. For **Master username**, enter a name for the master user, or keep the default name.

   The **Create database** page should look similar to the following image. For free plan accounts, **Free tier** appears. For paid plan accounts, **Sandbox** appears.

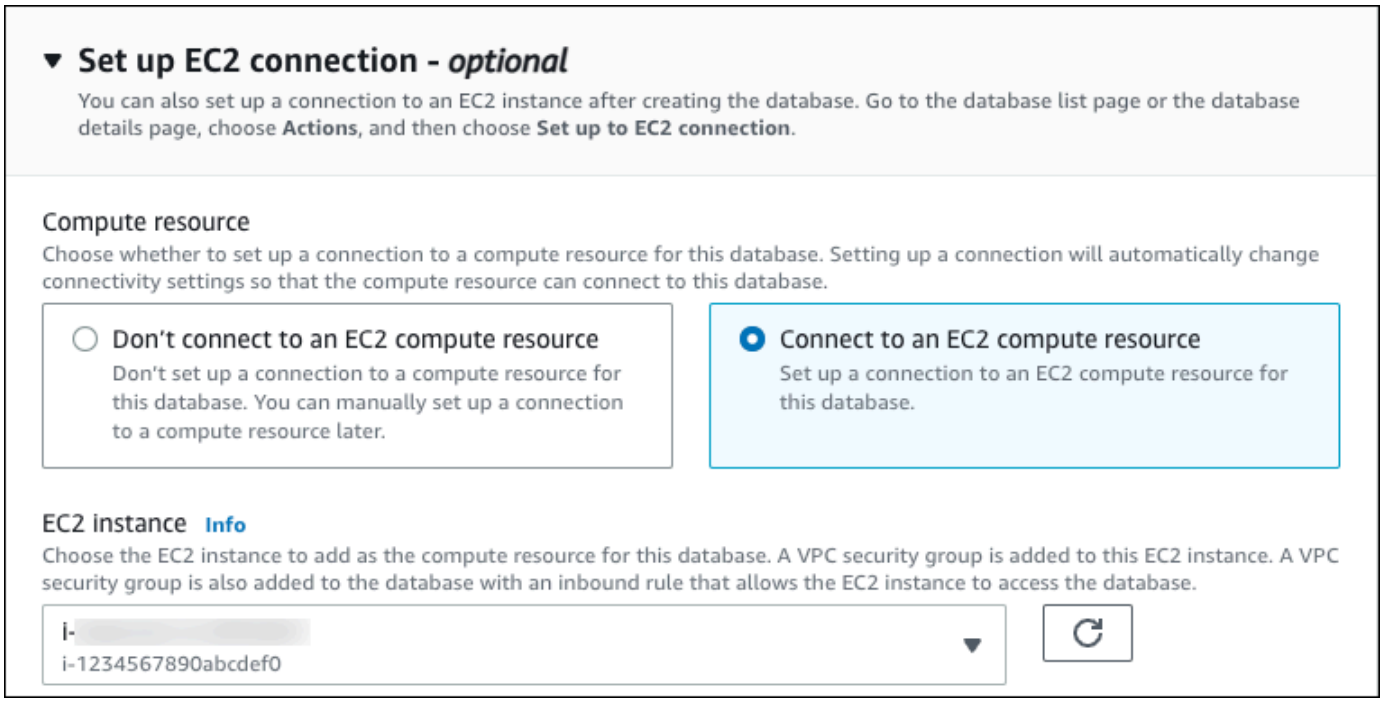


9. To use an automatically generated master password for the DB instance, select **Auto generate a password**.

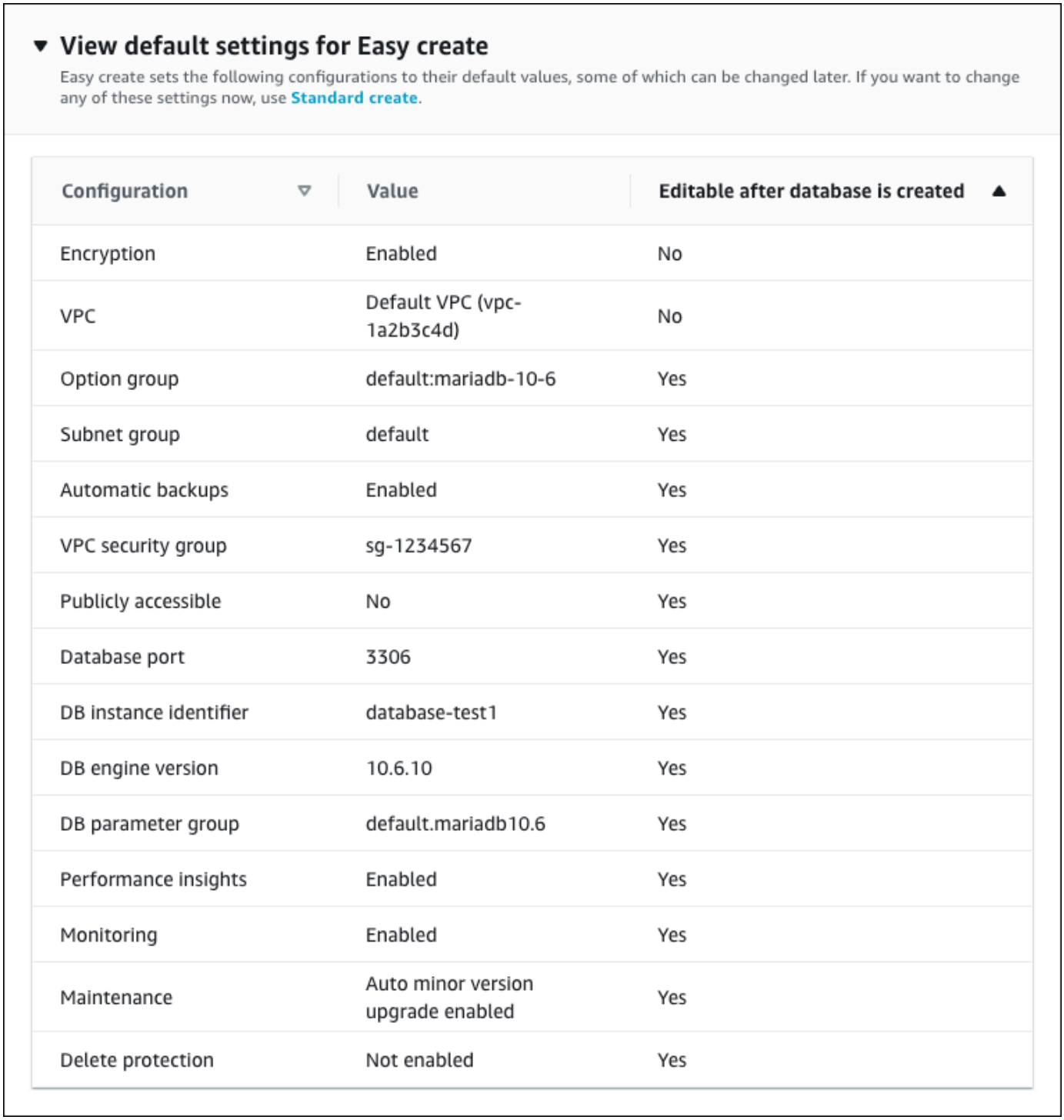
To enter your master password, clear **Auto generate a password**, and then enter the same password in **Master password** and **Confirm master password**.

10. To set up a connection with the EC2 instance you created previously, expand **Set up EC2 connection - *optional***.

    Select **Connect to an EC2 compute resource**. Choose the EC2 instance you created previously.



11. Expand **View default settings for Easy create**.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:mariadb-10-6	Yes
Subnet group	default	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	3306	Yes
DB instance identifier	database-test1	Yes
DB engine version	10.6.10	Yes
DB parameter group	default.mariadb10.6	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

You can examine the default settings used with **Easy create**. The **Editable after database is created** column shows which options you can change after you create the database.

- If a setting has **No** in that column, and you want a different setting, you can use **Standard create** to create the DB instance.

- If a setting has **Yes** in that column, and you want a different setting, you can either use **Standard create** to create the DB instance, or modify the DB instance after you create it to change the setting.

12. Choose **Create database**.

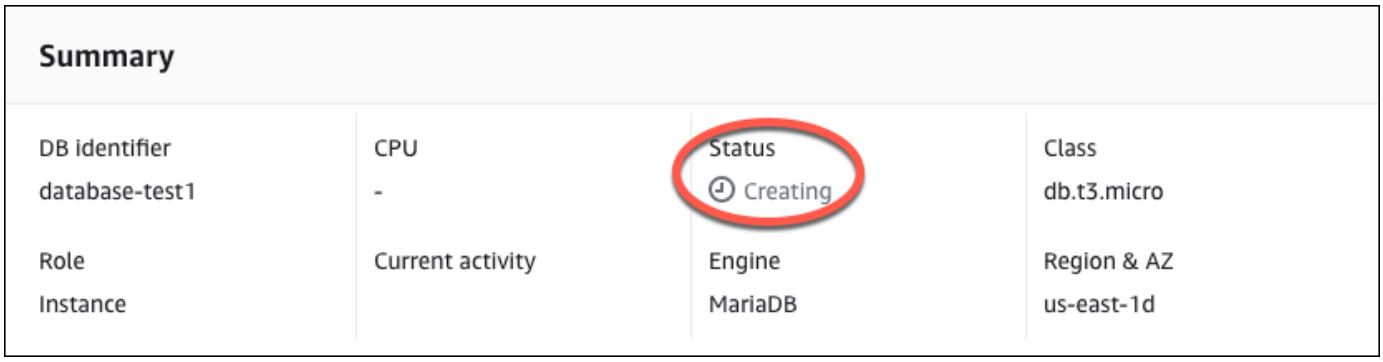
    To view the master username and password for the DB instance, choose **View credential details**.

    You can use the username and password that appears to connect to the DB instance as the master user.

    > **Important**
    >
    > You can't view the master user password again. If you don't record it, you might have to change it.
    > If you need to change the master user password after the DB instance is available, you can modify the DB instance to do so. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

13. In the **Databases** list, choose the name of the new MariaDB DB instance to show its details.

    The DB instance has a status of **Creating** until it is ready to use.



    When the status changes to **Available**, you can connect to the DB instance. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new instance is available.

(Optional) Create VPC, EC2 instance, and MariaDB instance using CloudFormation

Instead of using the console to create your VPC, EC2 instance, and MariaDB instance, you can use CloudFormation to provision AWS resources by treating infrastructure as code. To help you organize your AWS resources into smaller and more manageable units, you can use the CloudFormation nested stack functionality. For more information, see Creating a stack on the CloudFormation console and Working with nested stacks.

Important

CloudFormation is free, but the resources that CloudFormation creates are live. You incur the standard usage fees for these resources until you terminate them. For more information, see RDS for MariaDB pricing.

- Download the CloudFormation template
- Configure your resources using CloudFormation

Download the CloudFormation template

A CloudFormation template is a JSON or YAML text file that contains the configuration information about the resources you want to create in the stack. This template also creates a VPC and a bastion host for you along with the RDS instance.

To download the template file, open the following link, MariaDB CloudFormation template.

In the Github page, click the *Download raw file* button to save the template YAML file.

Configure your resources using CloudFormation

Note

Before starting this process, make sure you have a Key pair for an EC2 instance in your AWS account. For more information, see Amazon EC2 key pairs and Linux instances.

When you use the CloudFormation template, you must select the correct parameters to make sure your resources are created properly. Follow the steps below:

1. Sign in to the AWS Management Console and open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Choose **Create Stack**.
3. In the Specify template section, select **Upload a template file from your computer**, and then choose **Next**.
4. In the **Specify stack details** page, set the following parameters:
   a. Set **Stack name** to **MariaDBTestStack**.
   b. Under **Parameters**, set **Availability Zones** by selecting three availability zones.
   c. Under **Linux Bastion Host configuration**, for **Key Name**, select a key pair to login to your EC2 instance.
   d. In **Linux Bastion Host configuration** settings, set the **Permitted IP range** to your IP address. To connect to EC2 instances in your VPC using Secure Shell (SSH), determine your public IP address using the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

      > **Warning**
      >
      > If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

   e. Under **Database General configuration**, set **Database instance class** to **db.t3.micro**.
   f. Set **Database name** to **`database-test1`**.
   g. For **Database master username**, enter a name for the master user.
   h. Set **Manage DB master user password with Secrets Manager** to `false` for this tutorial.
   i. For **Database password**, set a password of your choice. Remember this password for further steps in the tutorial.
   j. Under **Database Storage configuration**, set **Database storage type** to **gp2**.
   k. Under **Database Monitoring configuration**, set **Enable RDS Performance Insights** to false.

l. Leave all other settings as the default values. Click **Next** to continue.

5. In the **Review stack** page, select **Submit** after checking the database and Linux bastion host options.

After the stack creation process completes, view the stacks with names *BastionStack* and *RDSNS* to note the information you need to connect to the database. For more information, see Viewing CloudFormation stack data and resources on the AWS Management Console.

Connect to a MariaDB DB instance

You can use any standard SQL client application to connect to the DB instance. In this example, you connect to a MariaDB DB instance using the mysql command-line client.

To connect to a MariaDB DB instance

1. Find the endpoint (DNS name) and port number for your DB instance.

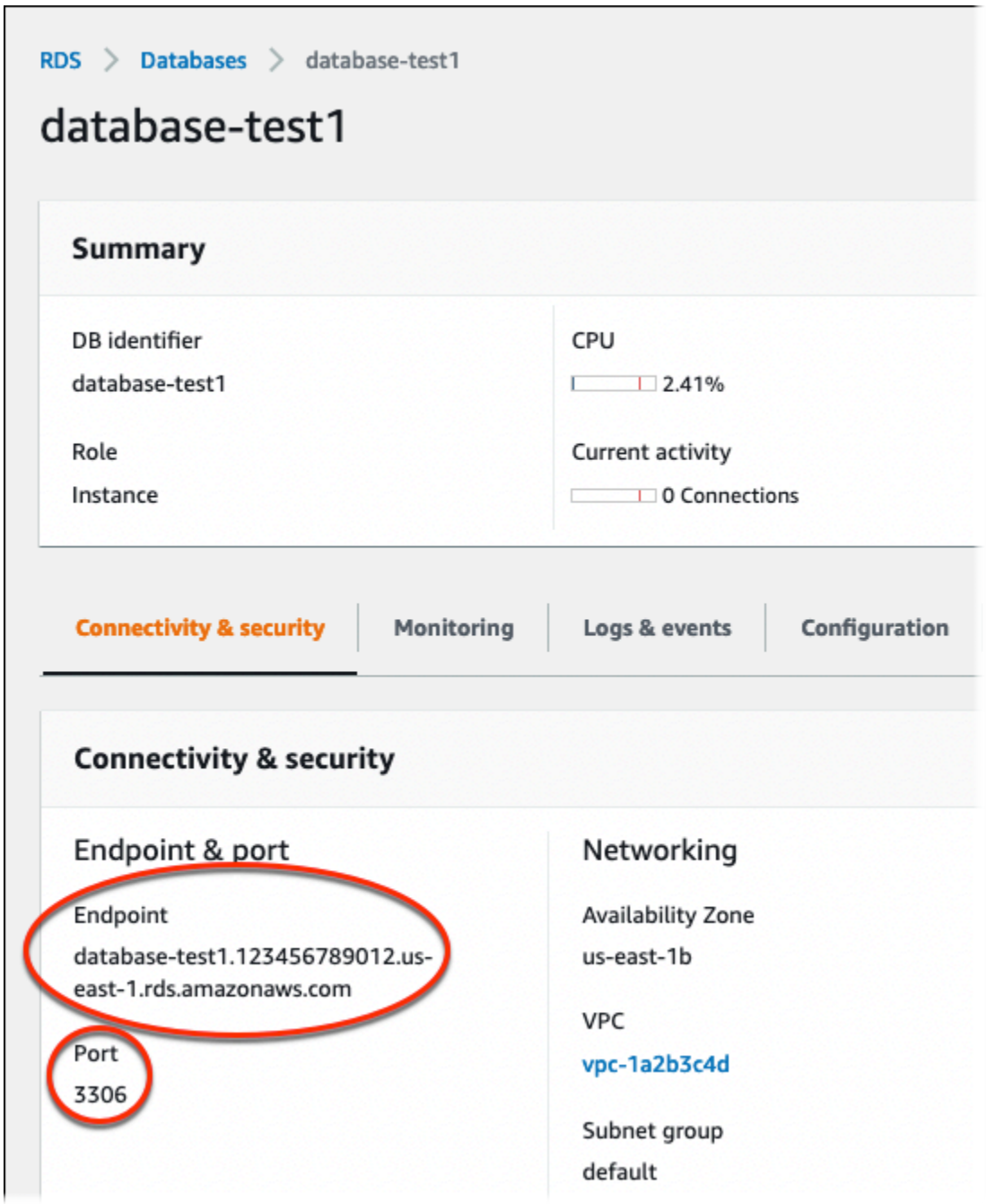
   a. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

   b. In the upper-right corner of the Amazon RDS console, choose the AWS Region for the DB instance.

   c. In the navigation pane, choose **Databases**.

   d. Choose the MariaDB DB instance name to display its details.

   e. On the **Connectivity & security** tab, copy the endpoint. Also note the port number. You need both the endpoint and the port number to connect to the DB instance.



2. Connect to the EC2 instance that you created earlier by following the steps in Connect to your Linux instance in the *Amazon EC2 User Guide*.

   We recommend that you connect to your EC2 instance using SSH. If the SSH client utility is installed on Windows, Linux, or Mac, you can connect to the instance using the following command format:

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

For example, assume that `ec2-database-connect-key-pair.pem` is stored in `/dir1` on Linux, and the public IPv4 DNS for your EC2 instance is `ec2-12-345-678-90.compute-1.amazonaws.com`. Your SSH command would look as follows:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. Get the latest bug fixes and security updates by updating the software on your EC2 instance. To do this, use the following command.

> **Note**
>
> The `-y` option installs the updates without asking for confirmation. To examine updates before installing, omit this option.

```
sudo dnf update -y
```

4. Install the mysql command-line client from MariaDB.

   To install the MariaDB command-line client on Amazon Linux 2023, run the following command:

```
sudo dnf install mariadb105
```

5. Connect to the MariaDB DB instance. For example, enter the following command. This action lets you connect to the MariaDB DB instance using the MySQL client.

   Substitute the DB instance endpoint (DNS name) for *`endpoint`*, and substitute the master username that you used for *`admin`*. Provide the master password that you used when prompted for a password.

```
mysql -h endpoint -P 3306 -u admin -p
```

   After you enter the password for the user, you should see output similar to the following.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
```

```
Your MariaDB connection id is 156
Server version: 10.6.10-MariaDB-log managed by https://aws.amazon.com/rds/

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

For more information about connecting to a MariaDB DB instance, see Connecting to your MariaDB DB instance. If you can't connect to your DB instance, see Can't connect to Amazon RDS DB instance.

For security, it is a best practice to use encrypted connections. Only use an unencrypted MariaDB connection when the client and server are in the same VPC and the network is trusted. For information about using encrypted connections, see Connecting to your MariaDB DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted).

6. Run SQL commands.

   For example, the following SQL command shows the current date and time:

```
SELECT CURRENT_TIMESTAMP;
```

Delete the EC2 instance and DB instance

After you connect to and explore the sample EC2 instance and DB instance that you created, delete them so you're no longer charged for them.

If you used CloudFormation to create resources, skip this step and go to the next step.

To delete the EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. Select the EC2 instance, and choose **Instance state, Terminate instance**.
4. Choose **Terminate** when prompted for confirmation.

For more information about deleting an EC2 instance, see Terminate your instance in the *Amazon EC2 User Guide*.

To delete the DB instance with no final DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance you want to delete.
4. For **Actions**, choose **Delete**.
5. Clear **Create final snapshot?** and **Retain automated backups**.
6. Complete the acknowledgement and choose **Delete**.

(Optional) Delete the EC2 instance and DB instance created with CloudFormation

If you used CloudFormation to create resources, delete the CloudFormation stack after you connect to and explore the sample EC2 instance and DB instance, so you're no longer charged for them.

To delete the CloudFormation resources

1. Open the CloudFormation console.
2. On the **Stacks** page in the CloudFormation console, select the root stack (the stack without the name VPCStack, BastionStack or RDSNS).
3. Choose **Delete**.
4. Select **Delete stack** when prompted for confirmation.

For more information about deleting a stack in CloudFormation, see Deleting a stack on the CloudFormation console in the *AWS CloudFormation User Guide*.

(Optional) Connect your DB instance to a Lambda function

You can also connect your RDS for MariaDB DB instance to a Lambda serverless compute resource. Lambda functions allow you to run code without provisioning or managing infrastructure. A Lambda function also allows you to automatically respond to code execution requests at any scale,

from a dozen events a day to hundreds of per second. For more information, see Automatically connecting a Lambda function and a DB instance.

Creating and connecting to a Microsoft SQL Server DB instance

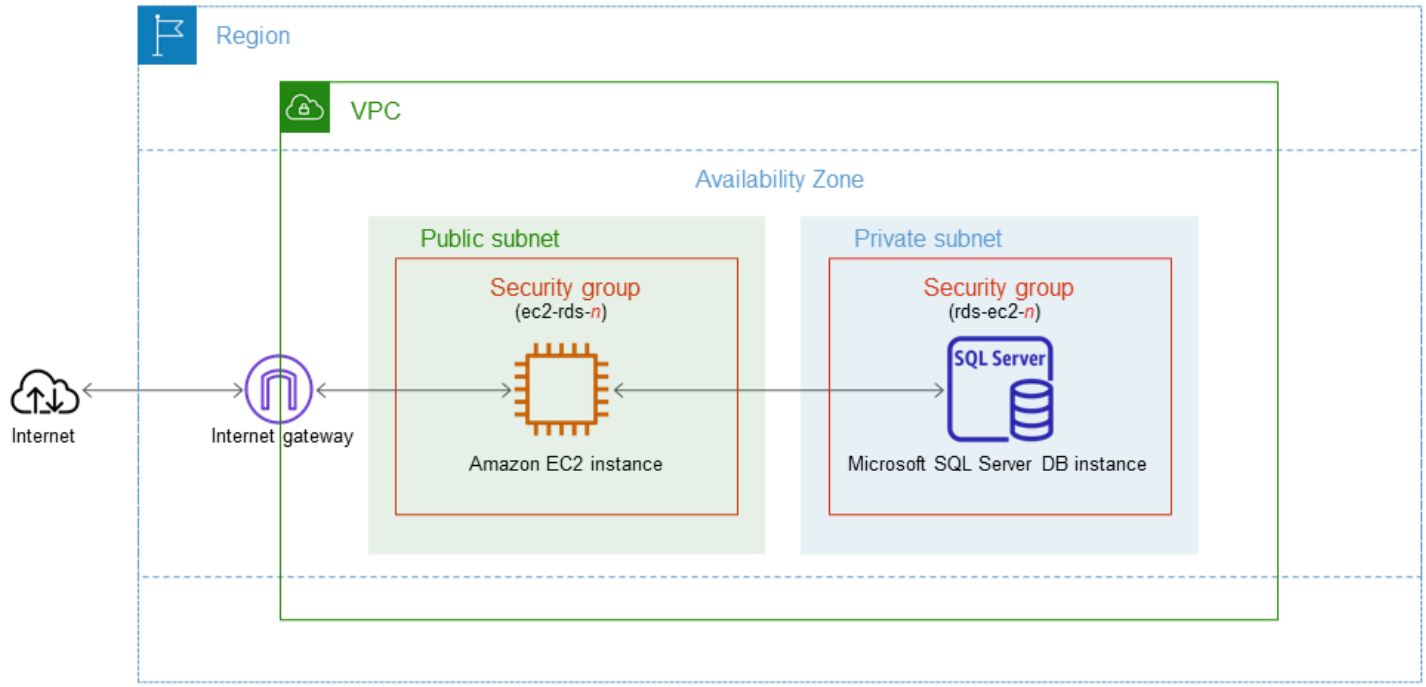
This tutorial creates an EC2 instance and an RDS for Microsoft SQL Server DB instance. The tutorial shows you how to access the DB instance from the EC2 instance using the Microsoft SQL Server Management Studio client. As a best practice, this tutorial creates a private DB instance in a virtual private cloud (VPC). In most cases, other resources in the same VPC, such as EC2 instances, can access the DB instance, but resources outside of the VPC can't access it.

After you complete the tutorial, there is a public and private subnet in each Availability Zone in your VPC. In one Availability Zone, the EC2 instance is in the public subnet, and the DB instance is in the private subnet.

Important

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources you use. You can delete these resources after you complete the tutorial if they are no longer needed.

The following diagram shows the configuration when the tutorial is complete.



This tutorial allows you to create your resources by using one of the following methods:

1. Use the AWS Management Console - Create a SQL Server DB instance and Create an EC2 instance
2. Use CloudFormation to create the database instance and EC2 instance - (Optional) Create VPC, EC2 instance, and SQL Server instance using CloudFormation

The first method uses **Easy create** to create a private SQL Server DB instance with the AWS Management Console. Here, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default settings for the other configuration options.

When you use **Standard create** instead, you can specify more configuration options when you create a DB instance. These options include settings for availability, security, backups, and maintenance. To create a public DB instance, you must use **Standard create**. For information, see Creating an Amazon RDS DB instance.

Topics

- Prerequisites
- Create an EC2 instance
- Create a SQL Server DB instance
- (Optional) Create VPC, EC2 instance, and SQL Server instance using CloudFormation
- Connect to your SQL Server DB instance
- Explore your sample SQL Server DB instance
- Delete the EC2 instance and DB instance
- (Optional) Delete the EC2 instance and DB instance created with CloudFormation
- (Optional) Connect your DB instance to a Lambda function

Prerequisites

Before you begin, complete the steps in the following sections:

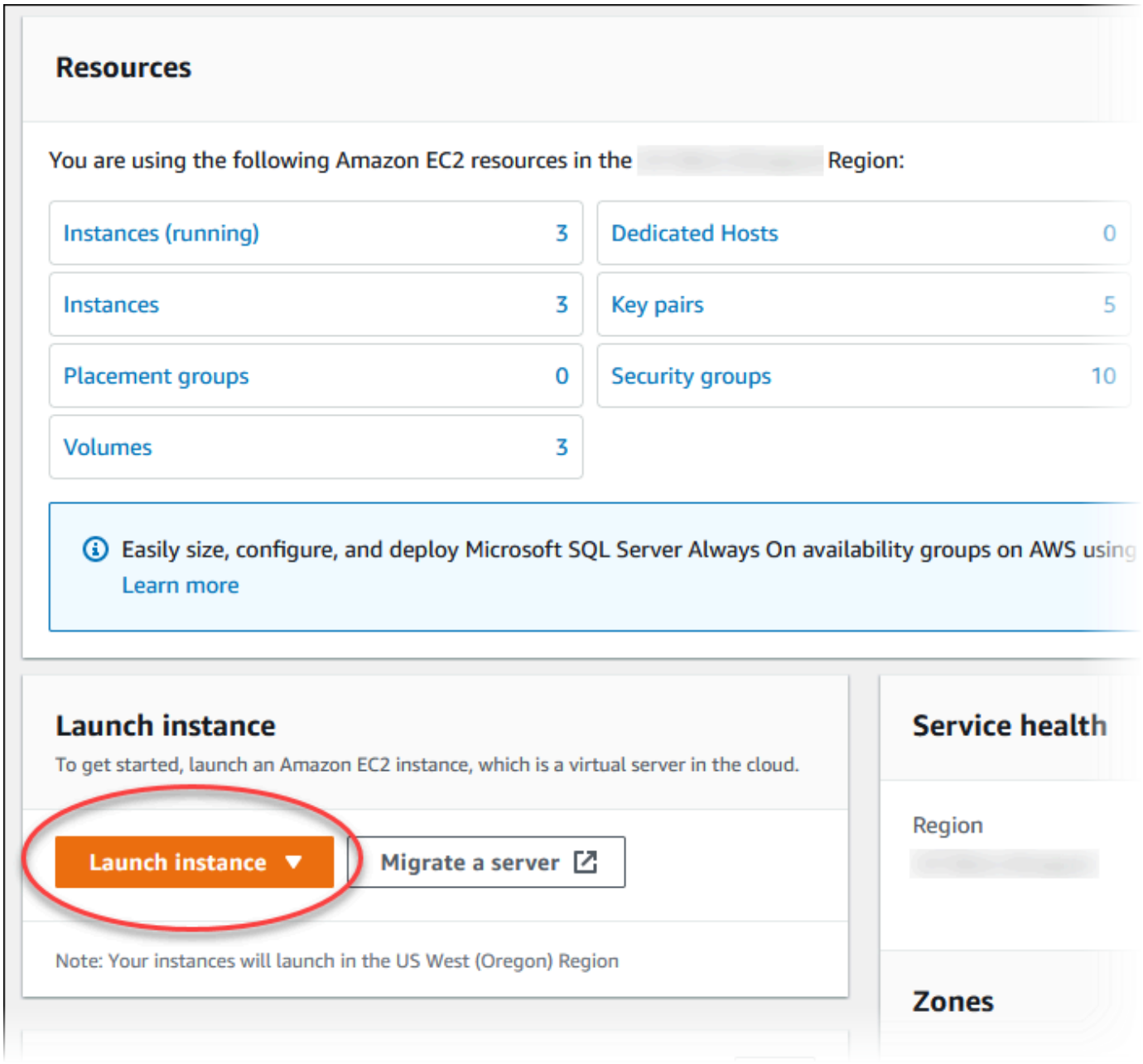
- Sign up for an AWS account
- Create a user with administrative access

Create an EC2 instance

Create an Amazon EC2 instance that you will use to connect to your database.

To create an EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region you used for the database previously.
3. Choose **EC2 Dashboard**, and then choose **Launch instance**, as shown in the following image.

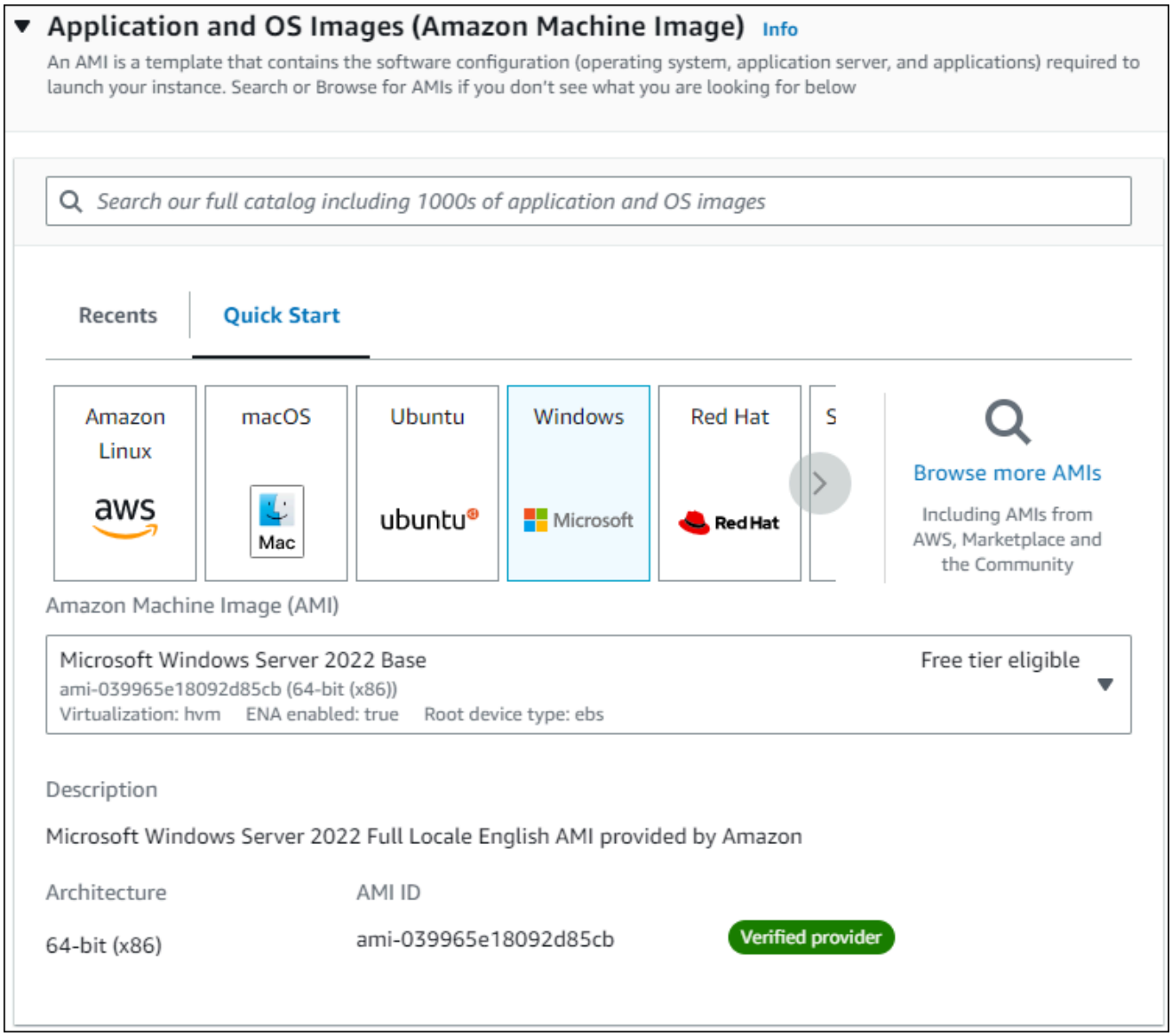


The **Launch an instance** page opens.

4. Choose the following settings on the **Launch an instance** page.

   a. Under **Name and tags**, for **Name**, enter **`ec2-database-connect`**.

   b. Under **Application and OS Images (Amazon Machine Image)**, choose **Windows**, and then choose the **Microsoft Windows Server 2022 Base**. Keep the default selections for the other choices.



   c. Under **Instance type**, choose **t2.micro**.

   d. Under **Key pair (login)**, choose a **Key pair name** to use an existing key pair. To create a new key pair for the Amazon EC2 instance, choose **Create new key pair** and then use the **Create key pair** window to create it.

For more information about creating a new key pair, see Create a key pair in the *Amazon EC2 User Guide for Windows Instances*.

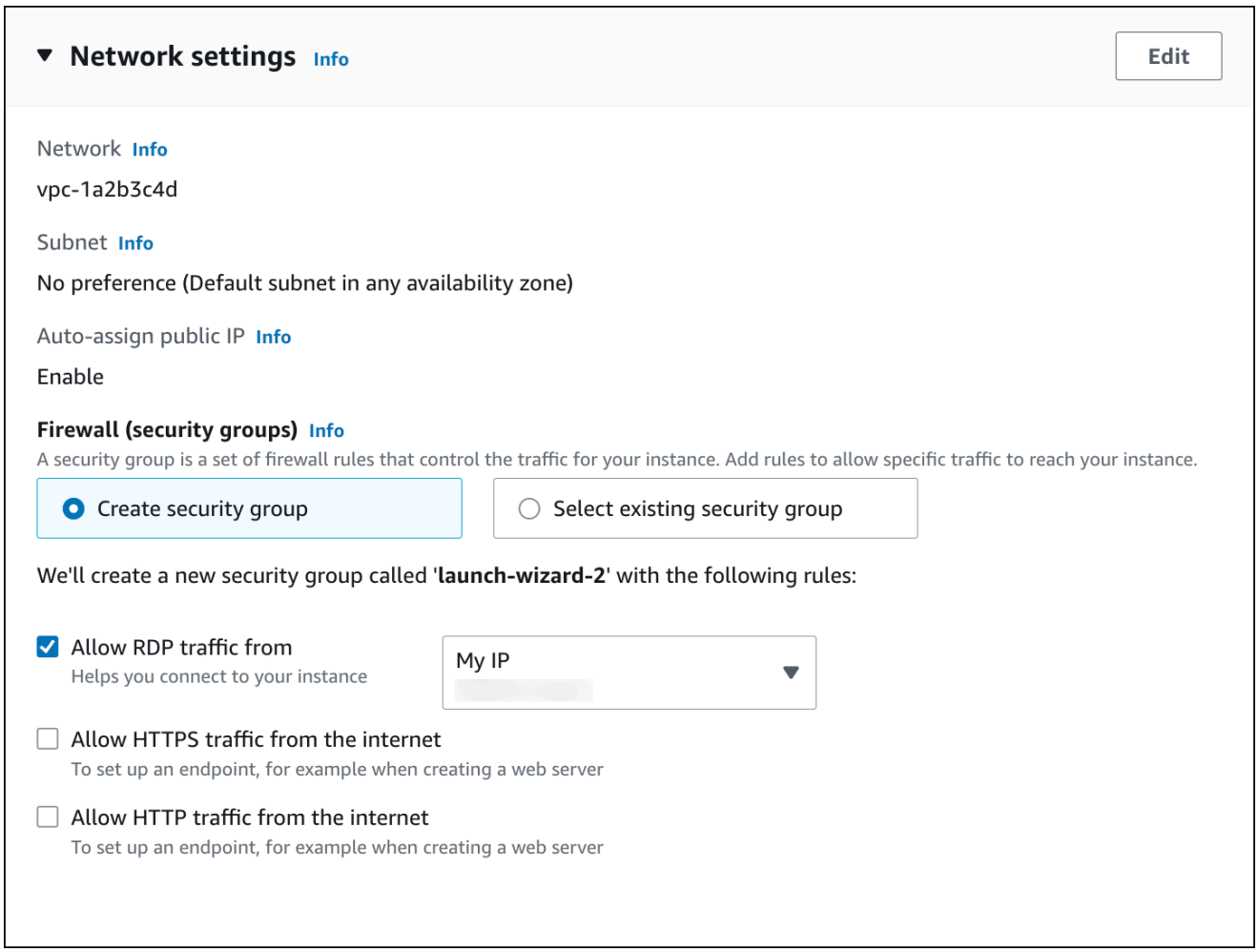
e. For **Firewall (security groups)** in **Network settings**, choose **Allow RDP traffic from** to connect to the EC2 instance.

You can choose **My IP** if the displayed IP address is correct for RDP connections. Otherwise, you can determine the IP address to use to connect to EC2 instances in your VPC using RDP. To determine your public IP address, in a different browser window or tab, you can use the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, make sure to determine the range of IP addresses used by client computers.

> **Warning**
>
> If you use `0.0.0.0/0` for RDP access, you make it possible for all IP addresses to access your public EC2 instances using RDP. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using RDP.

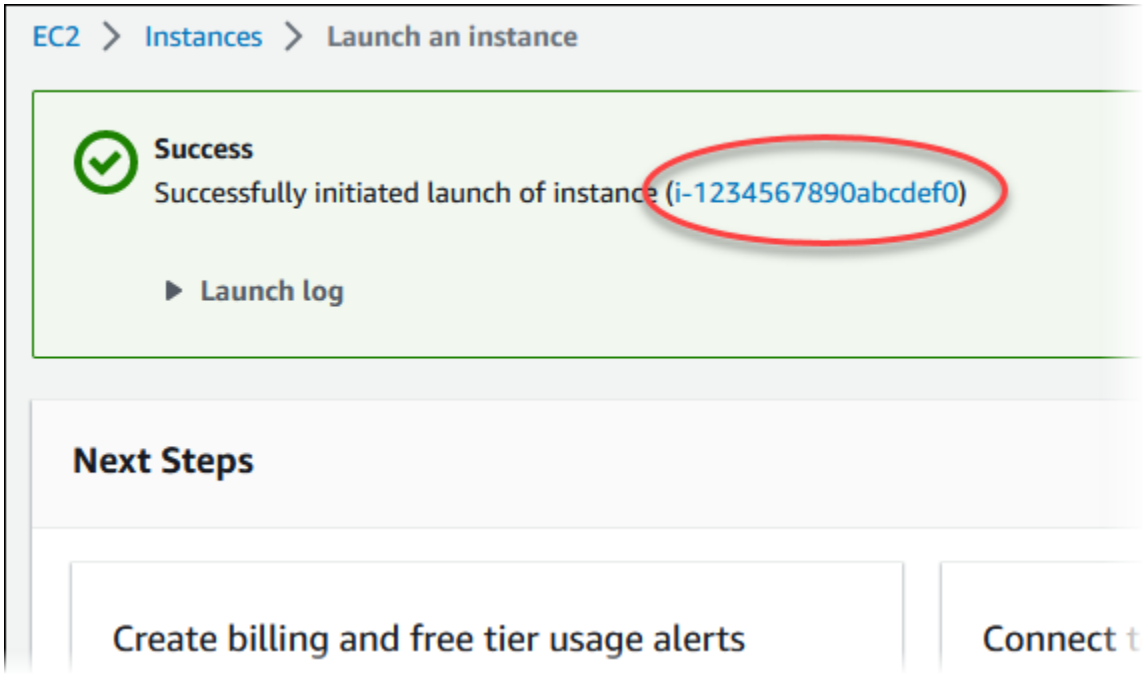
The following image shows an example of the **Network settings** section.



f. Keep the default values for the remaining sections.

g. Review a summary of your EC2 instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

5. On the **Launch Status** page, note the identifier for your new EC2 instance, for example: `i-1234567890abcdef0`.



6. Choose the EC2 instance identifier to open the list of EC2 instances.
7. Wait until the **Instance state** for your EC2 instance has a status of **Running** before continuing.

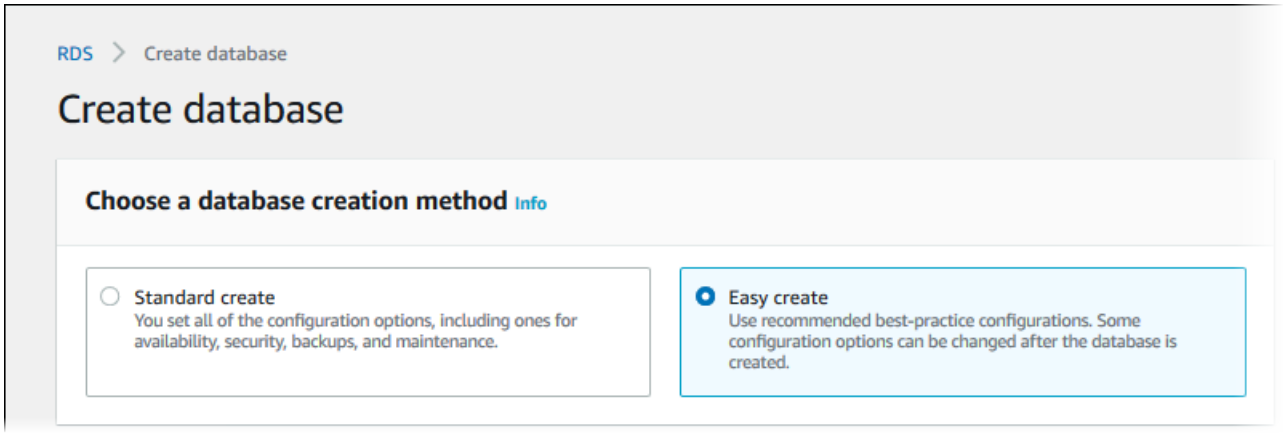
Create a SQL Server DB instance

The basic building block of Amazon RDS is the DB instance. This environment is where you run your SQL Server databases.

In this example, you use **Easy create** to create a DB instance running the SQL Server database engine with a db.t2.micro DB instance class.

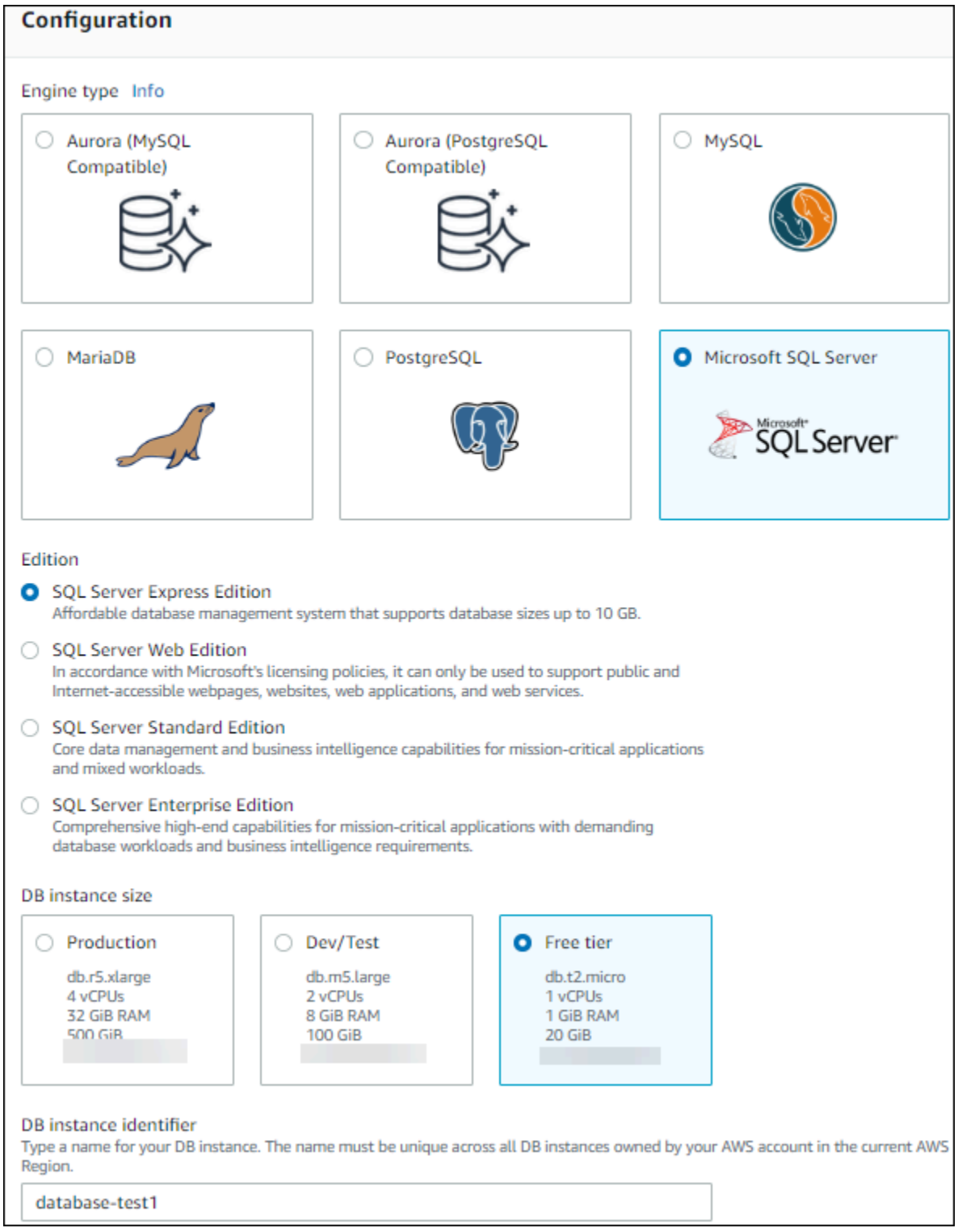
To create a Microsoft SQL Server DB instance with Easy create

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database** and make sure that **Easy create** is chosen.



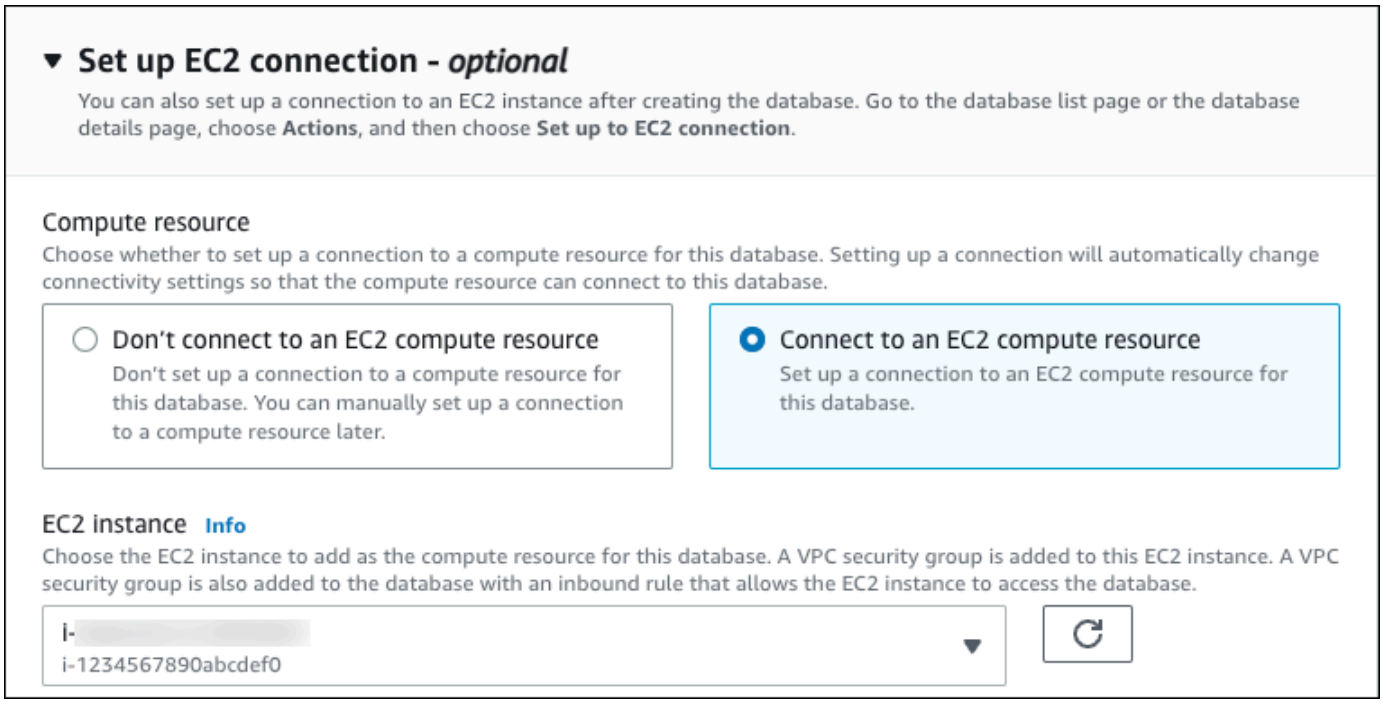
5. In **Configuration**, choose **Microsoft SQL Server**.
6. For **Edition**, choose **SQL Server Express Edition**.
7. For **DB instance size**, choose **Free tier**. **Free tier** appears for free plan accounts. **Sandbox** appears for paid plan accounts.
8. For **DB instance identifier**, enter **`database-test1`**.

   The **Create database** page should look similar to the following image. For free plan accounts, **Free tier** appears. For paid plan accounts, **Sandbox** appears.



9. For **Master username**, enter a name for the master user, or keep the default name.
10. To set up a connection with the EC2 instance you created previously, open **Set up EC2 connection - *optional***.

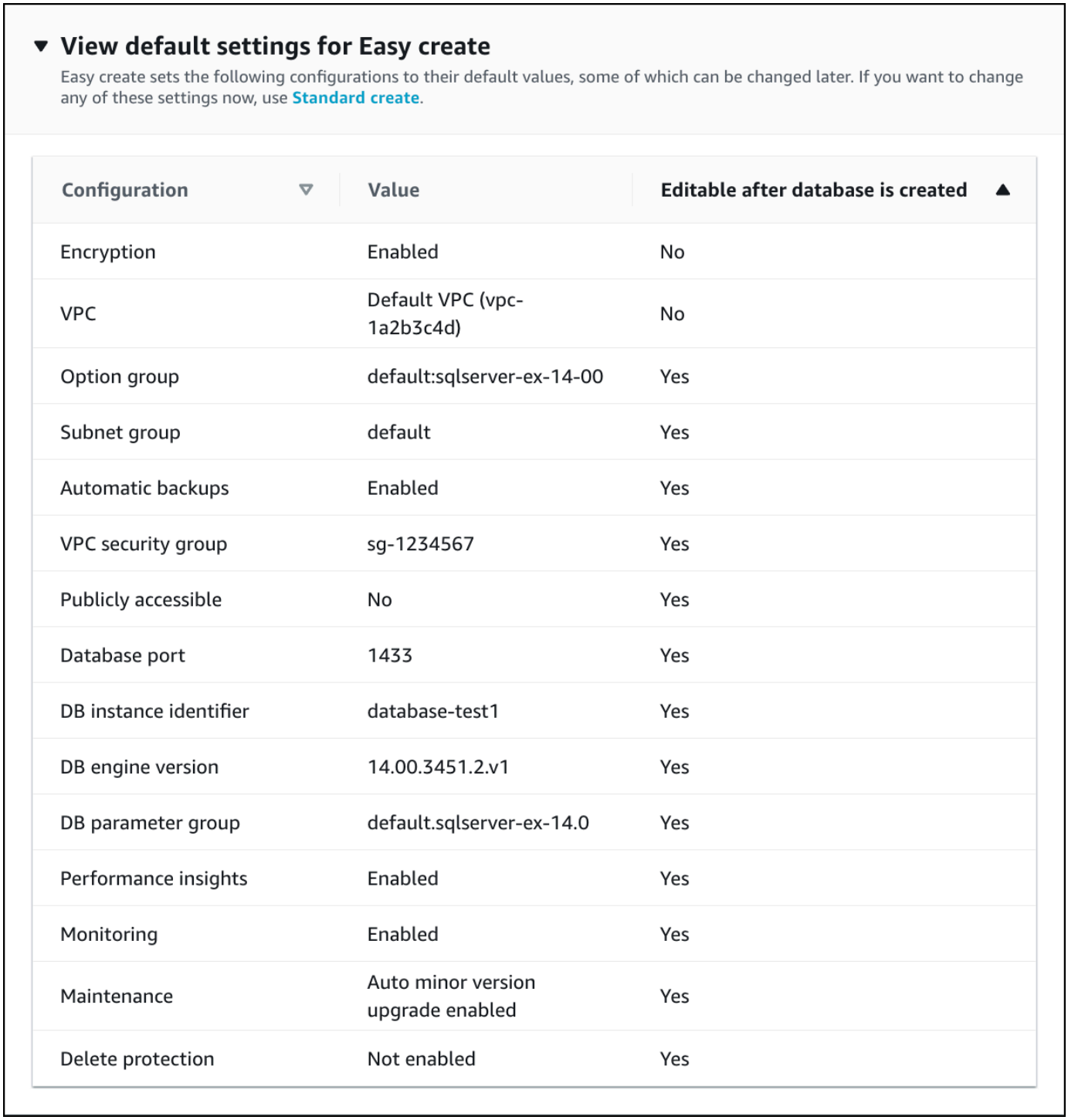
Select **Connect to an EC2 compute resource**. Choose the EC2 instance you created previously.



11. To use an automatically generated master password for the DB instance, select the **Auto generate a password** box.

    To enter your master password, clear the **Auto generate a password** box, and then enter the same password in **Master password** and **Confirm password**.

12. Open **View default settings for Easy create**.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:sqlserver-ex-14-00	Yes
Subnet group	default	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	1433	Yes
DB instance identifier	database-test1	Yes
DB engine version	14.00.3451.2.v1	Yes
DB parameter group	default.sqlserver-ex-14.0	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

You can examine the default settings used with **Easy create**. The **Editable after database is created** column shows which options you can change after you create the database.

- If a setting has **No** in that column, and you want a different setting, you can use **Standard create** to create the DB instance.

- If a setting has **Yes** in that column, and you want a different setting, you can either use **Standard create** to create the DB instance, or modify the DB instance after you create it to change the setting.

13. Choose **Create database**.

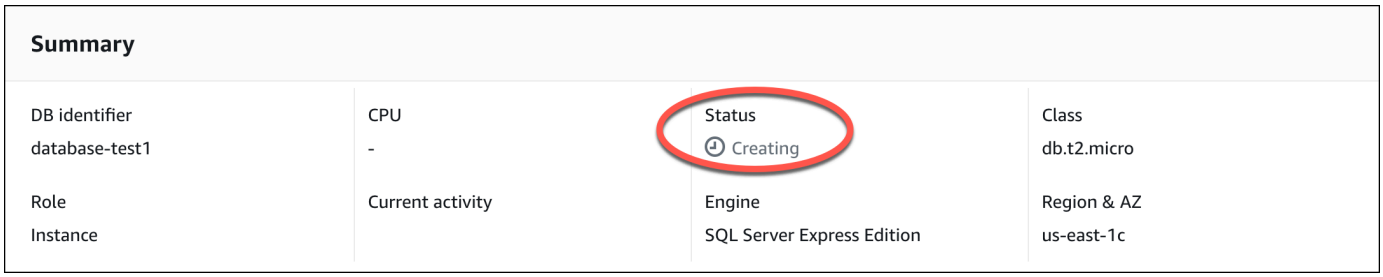
    To view the master username and password for the DB instance, choose **View credential details**.

    You can use the username and password that appears to connect to the DB instance as the master user.

    > **Important**
    >
    > You can't view the master user password again. If you don't record it, you might have to change it.
    > If you need to change the master user password after the DB instance is available, you can modify the DB instance to do so. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

14. In the **Databases** list, choose the name of the new SQL Server DB instance to show its details.

    The DB instance has a status of **Creating** until it is ready to use.



    When the status changes to **Available**, you can connect to the DB instance. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new instance is available.

(Optional) Create VPC, EC2 instance, and SQL Server instance using CloudFormation

Instead of using the console to create your VPC, EC2 instance, and SQL Server instance, you can use CloudFormation to provision AWS resources by treating infrastructure as code. To help you organize your AWS resources into smaller and more manageable units, you can use the CloudFormation nested stack functionality. For more information, see Creating a stack on the CloudFormation console and Working with nested stacks..

Important

CloudFormation is free, but the resources that CloudFormation creates are live. You incur the standard usage fees for these resources until you terminate them. For more information, see RDS for SQL Server pricing.

To create your resources using the CloudFormation console, complete the following steps:

- Download the CloudFormation template
- Configure your resources using CloudFormation

Download the CloudFormation template

A CloudFormation template is a JSON or YAML text file that contains the configuration information about the resources you want to create in the stack. This template also creates a VPC and a bastion host for you along with the RDS instance.

To download the template file, open the following link, SQL Server CloudFormation template.

In the Github page, click the *Download raw file* button to save the template YAML file.

Configure your resources using CloudFormation

Note

Before starting this process, make sure you have a Key pair for an EC2 instance in your AWS account. For more information, see Amazon EC2 key pairs and Linux instances.

When you use the CloudFormation template, you must select the correct parameters to make sure your resources are created properly. Follow the steps below:

1. Sign in to the AWS Management Console and open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Choose **Create Stack**.
3. In the Specify template section, select **Upload a template file from your computer**, and then choose **Next**.
4. In the **Specify stack details** page, set the following parameters:
   a. Set **Stack name** to **SQLServerTestStack**.
   b. Under **Parameters**, set **Availability Zones** by selecting three availability zones.
   c. Under **Linux Bastion Host configuration**, for **Key Name**, select a key pair to login to your EC2 instance.
   d. In **Linux Bastion Host configuration** settings, set the **Permitted IP range** to your IP address. To connect to EC2 instances in your VPC using Secure Shell (SSH), determine your public IP address using the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

      > **Warning**
      >
      > If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

   e. Under **Database General configuration**, set **Database instance class** to **db.t3.micro**.
   f. Set **Database name** to **`database-test1`**.
   g. For **Database master username**, enter a name for the master user.
   h. Set **Manage DB master user password with Secrets Manager** to `false` for this tutorial.
   i. For **Database password**, set a password of your choice. Remember this password for further steps in the tutorial.
   j. Under **Database Storage configuration**, set **Database storage type** to **gp2**.
   k. Under **Database Monitoring configuration**, set **Enable RDS Performance Insights** to false.

l. Leave all other settings as the default values. Click **Next** to continue.

5. In the **Configure stack options** page, leave all the default options. Click **Next** to continue.
6. In the **Review stack** page, select **Submit** after checking the database and Linux bastion host options.

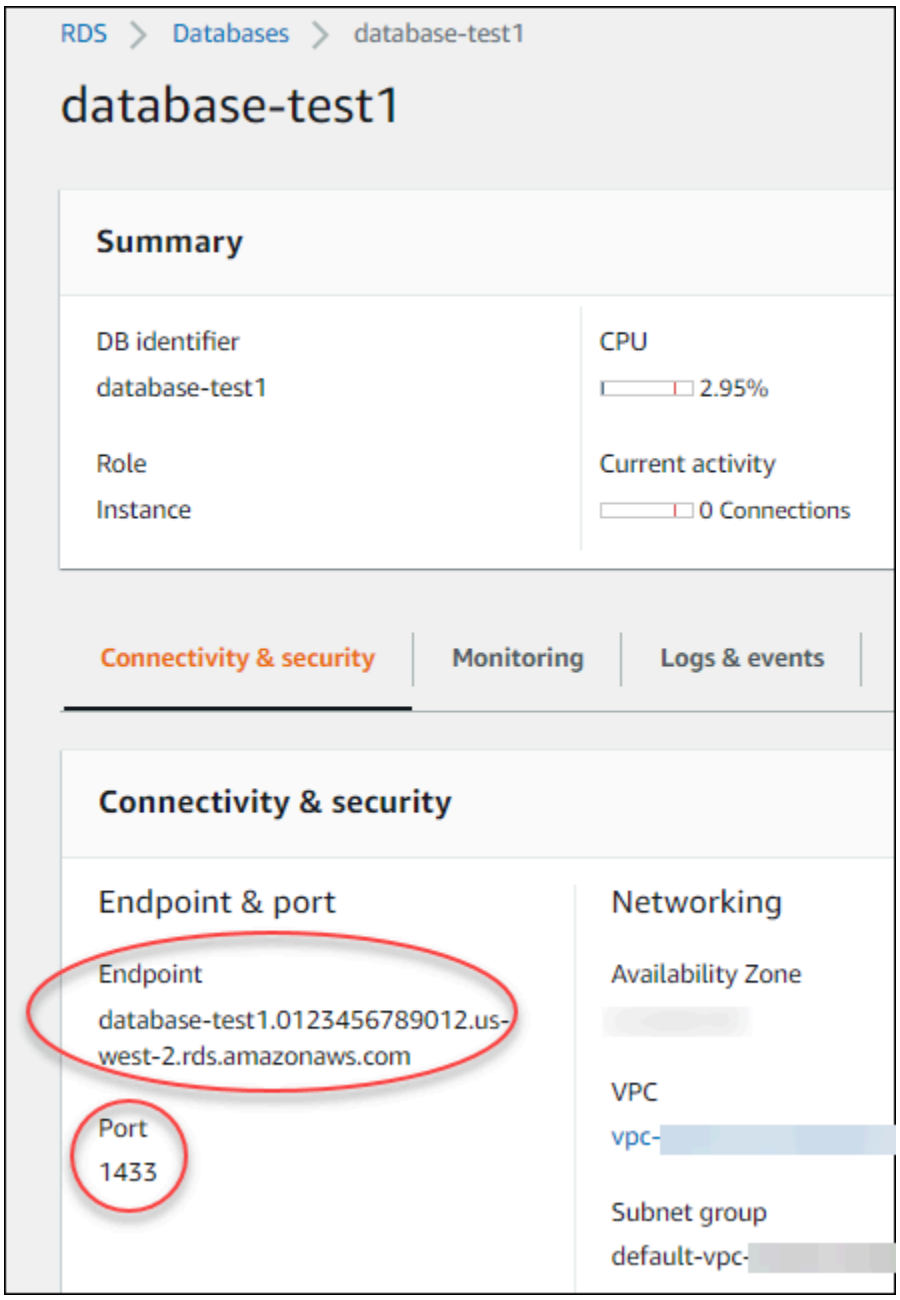
After the stack creation process completes, view the stacks with names *BastionStack* and *RDSNS* to note the information you need to connect to the database. For more information, see Viewing CloudFormation stack data and resources on the AWS Management Console.

Connect to your SQL Server DB instance

In the following procedure, you connect to your DB instance by using Microsoft SQL Server Management Studio (SSMS).

To connect to an RDS for SQL Server DB instance using SSMS

1. Find the endpoint (DNS name) and port number for your DB instance.
   a. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
   b. In the upper-right corner of the Amazon RDS console, choose the AWS Region for the DB instance.
   c. In the navigation pane, choose **Databases**.
   d. Choose the SQL Server DB instance name to display its details.
   e. On the **Connectivity** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



2. Connect to the EC2 instance that you created earlier by following the steps in Connect to your Microsoft Windows instance in the *Amazon EC2 User Guide for Windows Instances*.
3. Install the SQL Server Management Studio (SSMS) client from Microsoft.

   To download a standalone version of SSMS to your EC2 instance, see Download SQL Server Management Studio (SSMS) in the Microsoft documentation.

   a. Use the Start menu to open Internet Explorer.

b. Use Internet Explorer to download and install a standalone version of SSMS. If you are prompted that the site isn't trusted, add the site to the list of trusted sites.

4. Start SQL Server Management Studio (SSMS).

   The **Connect to Server** dialog box appears.

5. Provide the following information for your sample DB instance:

   a. For **Server type**, choose **Database Engine**.

   b. For **Server name**, enter the DNS name, followed by a comma and the port number (the default port is 1433). For example, your server name should look as follows:

   ```
   database-test1.0123456789012.us-west-2.rds.amazonaws.com,1433
   ```

   c. For **Authentication**, choose **SQL Server Authentication**.

   d. For **Login**, enter the username that you chose to use for your sample DB instance. This is also known as the master username.

   e. For **Password**, enter the password that you chose earlier for your sample DB instance. This is also known as the master user password.

6. Choose **Connect**.

   After a few moments, SSMS connects to your DB instance. For security, it is a best practice to use encrypted connections. Only use an unencrypted SQL Server connection when the client and server are in the same VPC and the network is trusted. For information about using encrypted connections, see Using SSL with a Microsoft SQL Server DB instance

For more information about connecting to a Microsoft SQL Server DB instance, see Connecting to your Microsoft SQL Server DB instance.

For information about connection issues, see Can't connect to Amazon RDS DB instance.

Explore your sample SQL Server DB instance

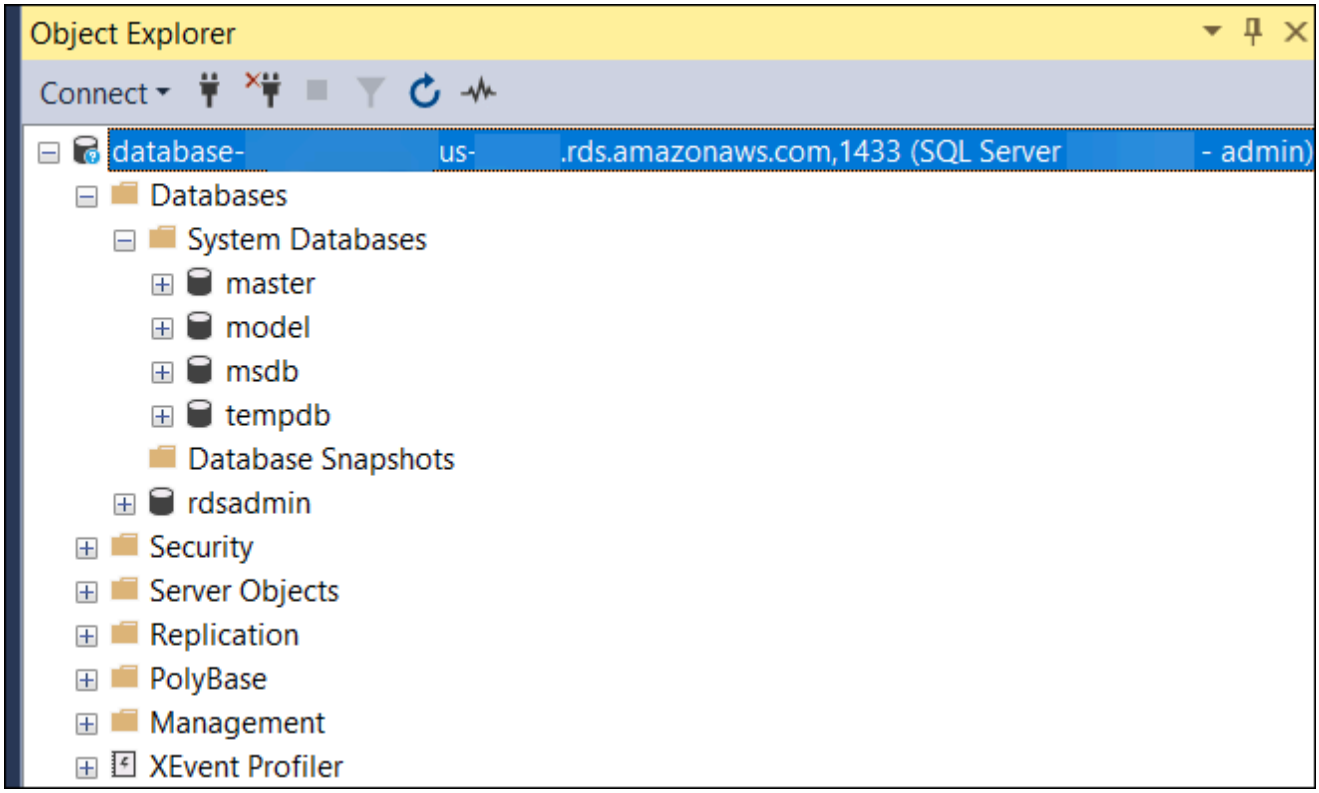
You can explore your sample DB instance by using Microsoft SQL Server Management Studio (SSMS).

To explore a DB instance using SSMS

1. Your SQL Server DB instance comes with SQL Server's standard built-in system databases (master, model, msdb, and tempdb). To explore the system databases, do the following:

   a. In SSMS, on the **View** menu, choose **Object Explorer**.

   b. Expand your DB instance, expand **Databases**, and then expand **System Databases** as shown.



   Your SQL Server DB instance also comes with a database named `rdsadmin`. Amazon RDS uses this database to store the objects that it uses to manage your database. The `rdsadmin` database also includes stored procedures that you can run to perform advanced tasks.

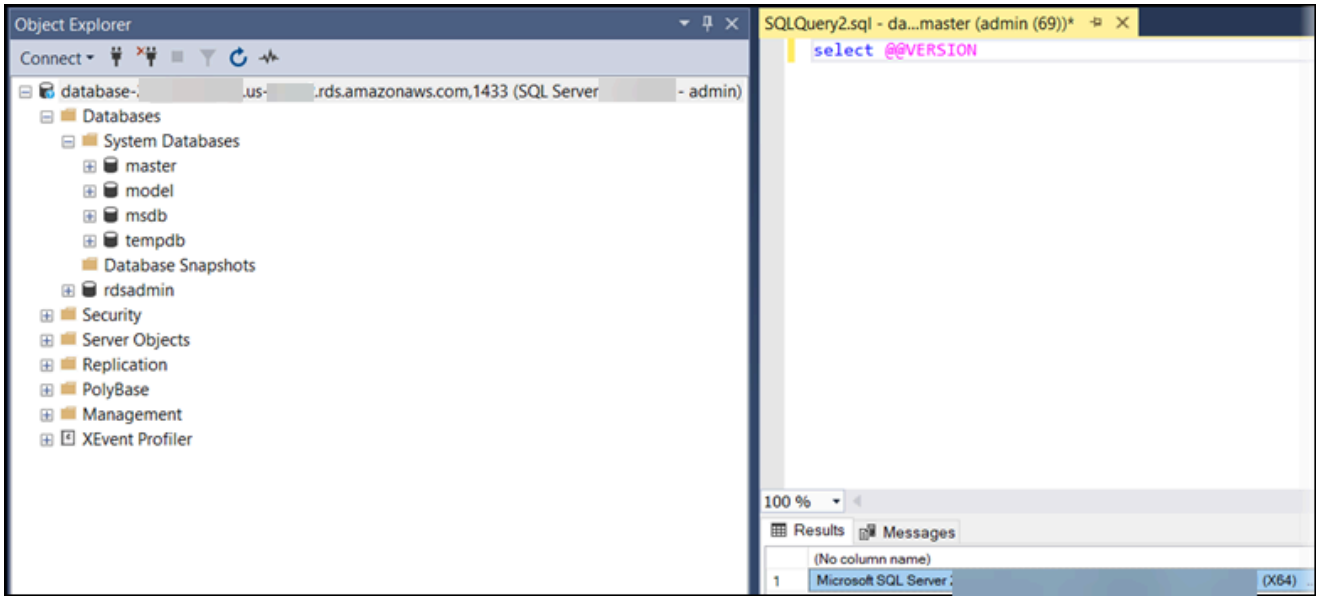
2. Start creating your own databases and running queries against your DB instance and databases as usual. To run a test query against your sample DB instance, do the following:

   a. In SSMS, on the **File** menu, point to **New** and then choose **Query with Current Connection**.

   b. Enter the following SQL query:

```
select @@VERSION
```

c. Run the query. SSMS returns the SQL Server version of your Amazon RDS DB instance.



Delete the EC2 instance and DB instance

After you connect to and explore the sample EC2 instance and DB instance that you created, delete them so you're no longer charged for them.

If you used CloudFormation to create resources, skip this step and go to the next step.

To delete the EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. Select the EC2 instance, and choose **Instance state, Terminate instance**.
4. Choose **Terminate** when prompted for confirmation.

For more information about deleting an EC2 instance, see Terminate your instance in the *User Guide for Windows Instances*.

To delete the DB instance with no final DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to delete.
4. For **Actions**, choose **Delete**.
5. Clear **Create final snapshot?** and **Retain automated backups**.
6. Complete the acknowledgement and choose **Delete**.

(Optional) Delete the EC2 instance and DB instance created with CloudFormation

If you used CloudFormation to create resources, delete the CloudFormation stack after you connect to and explore the sample EC2 instance and DB instance, so you're no longer charged for them.

To delete the CloudFormation resources

1. Open the CloudFormation console.
2. On the **Stacks** page in the CloudFormationconsole, select the root stack (the stack without the name VPCStack, BastionStack or RDSNS).
3. Choose **Delete**.
4. Select **Delete stack** when prompted for confirmation.

For more information about deleting a stack in CloudFormation, see Deleting a stack on the CloudFormation console in the *AWS CloudFormation User Guide*.

(Optional) Connect your DB instance to a Lambda function

You can also connect your RDS for SQL Server DB instance to a Lambda serverless compute resource. Lambda functions allow you to run code without provisioning or managing infrastructure. A Lambda function also allows you to automatically respond to code execution requests at any scale, from a dozen events a day to hundreds of per second. For more information, see Automatically connecting a Lambda function and a DB instance.

Creating and connecting to a MySQL DB instance

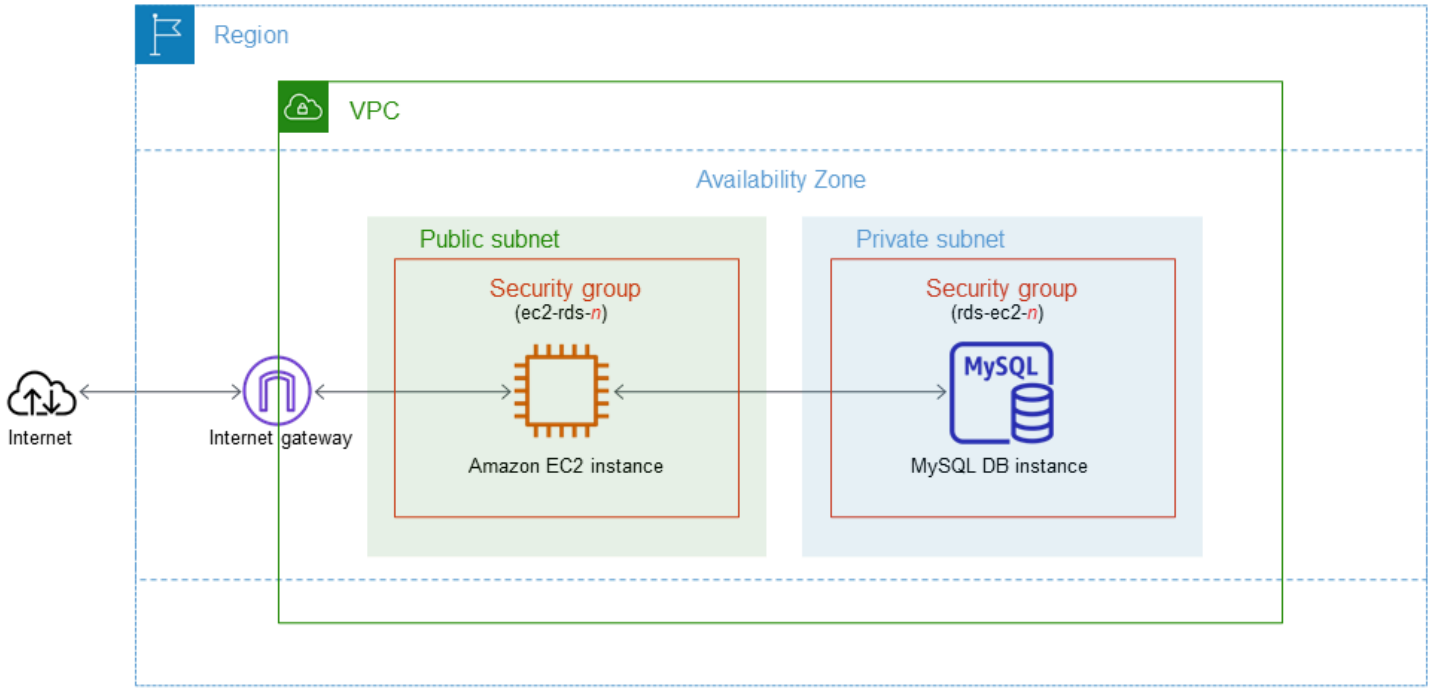
This tutorial creates an EC2 instance and an RDS for MySQL DB instance. The tutorial shows you how to access the DB instance from the EC2 instance using a standard MySQL client. As a best practice, this tutorial creates a private DB instance in a virtual private cloud (VPC). In most cases, other resources in the same VPC, such as EC2 instances, can access the DB instance, but resources outside of the VPC can't access it.

After you complete the tutorial, there is a public and private subnet in each Availability Zone in your VPC. In one Availability Zone, the EC2 instance is in the public subnet, and the DB instance is in the private subnet.

Important

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources you use. You can delete these resources after you complete the tutorial if they are no longer needed.

The following diagram shows the configuration when the tutorial is complete.



This tutorial allows you to create your resources by using one of the following methods:

1. Use the AWS Management Console - Create a MySQL DB instance and Create an EC2 instance
2. Use CloudFormation to create the database instance and EC2 instance - (Optional) Create VPC, EC2 instance, and MySQL instance using CloudFormation

The first method uses **Easy create** to create a private MySQL DB instance with the AWS Management Console. Here, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default settings for the other configuration options.

When you use **Standard create** instead, you can specify more configuration options when you create a DB instance. These options include settings for availability, security, backups, and maintenance. To create a public DB instance, you must use **Standard create**. For information, see Creating an Amazon RDS DB instance.

Topics

- Prerequisites
- Create an EC2 instance
- Create a MySQL DB instance
- (Optional) Create VPC, EC2 instance, and MySQL instance using CloudFormation
- Connect to a MySQL DB instance
- Delete the EC2 instance and DB instance
- (Optional) Delete the EC2 instance and DB instance created with CloudFormation
- (Optional) Connect your DB instance to a Lambda function

Prerequisites

Before you begin, complete the steps in the following sections:

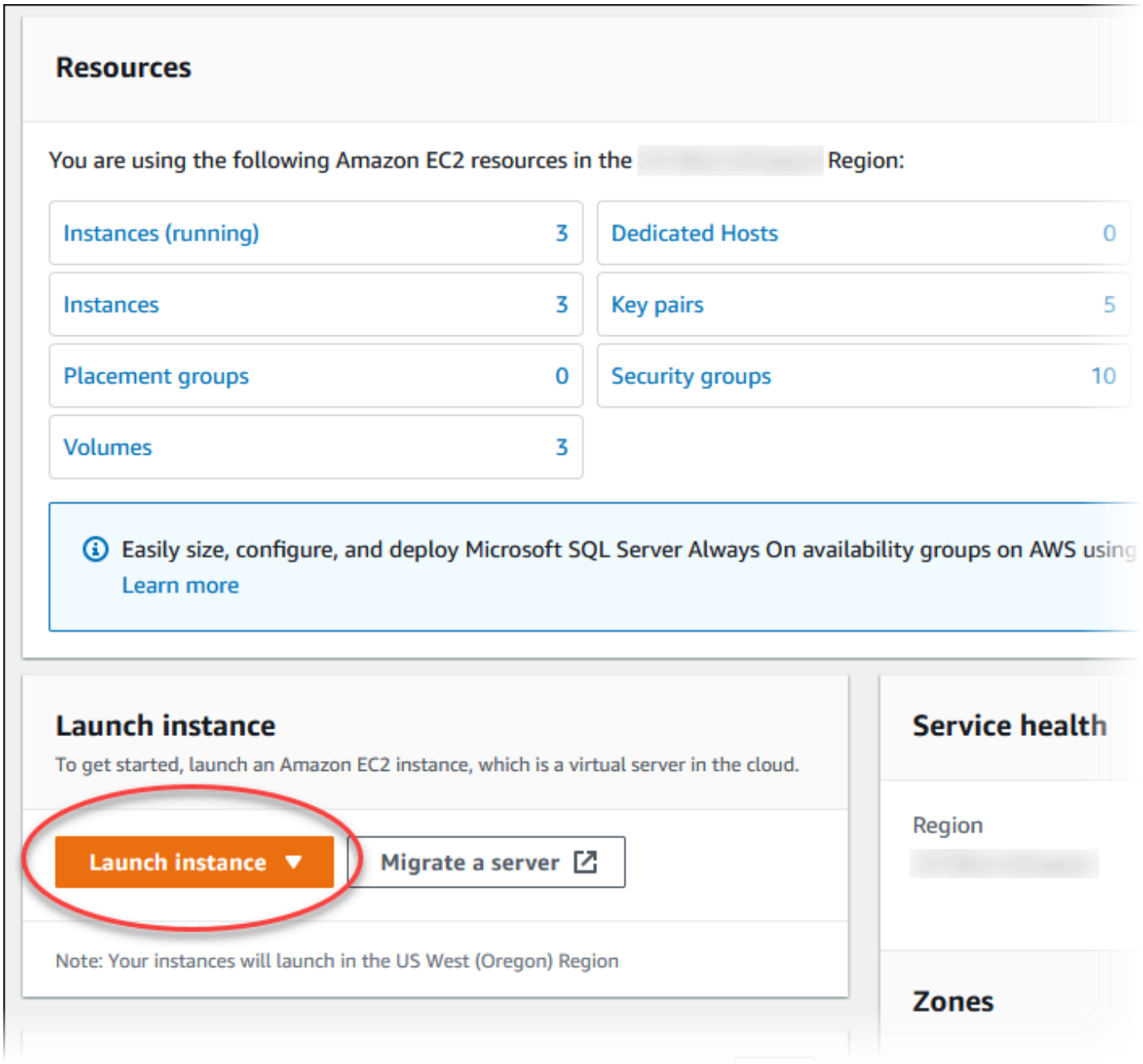
- Sign up for an AWS account
- Create a user with administrative access

Create an EC2 instance

Create an Amazon EC2 instance that you will use to connect to your database.

To create an EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you want to create the EC2 instance.
3. Choose **EC2 Dashboard**, and then choose **Launch instance**, as shown in the following image.

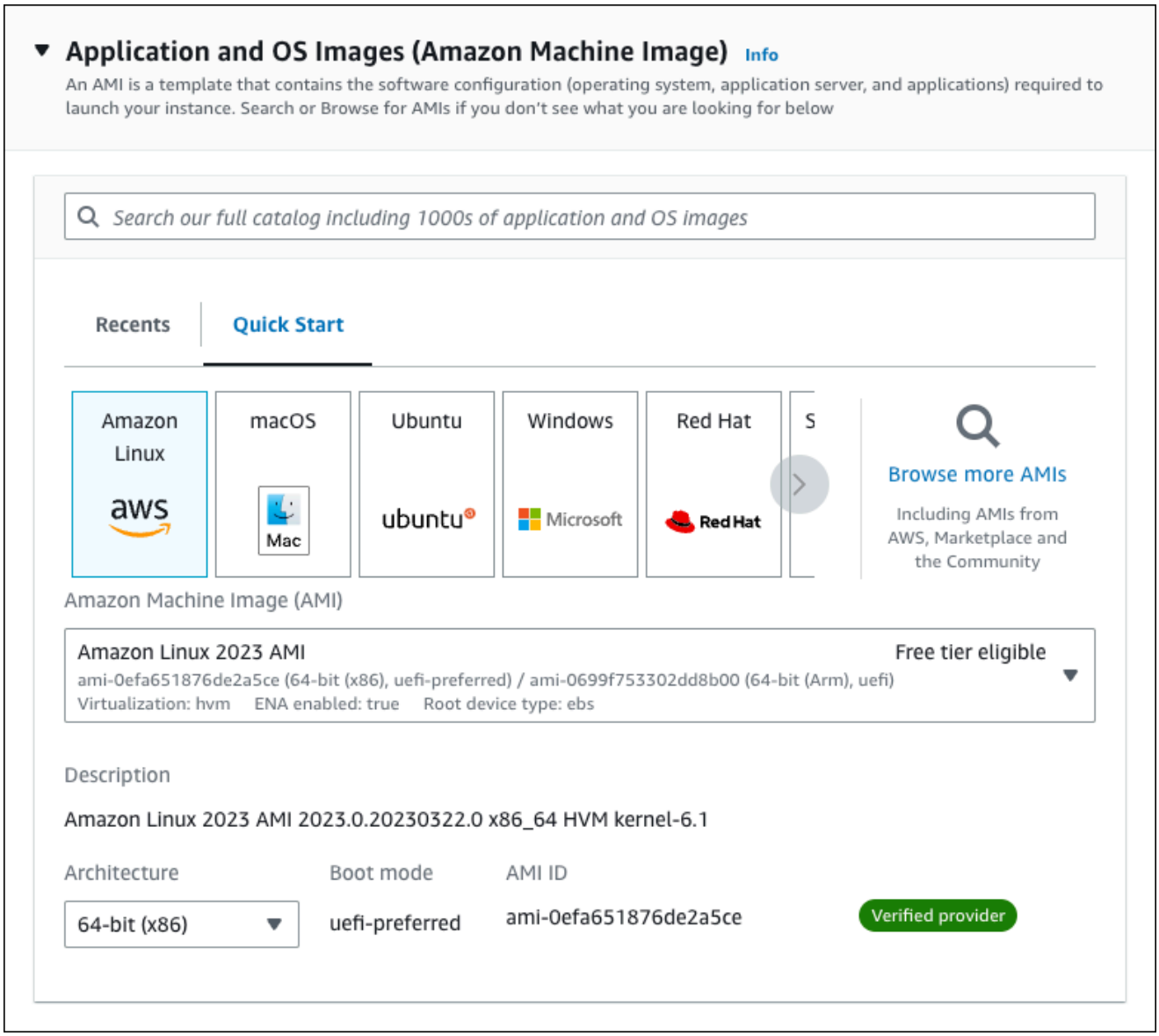


The **Launch an instance** page opens.

4. Choose the following settings on the **Launch an instance** page.

   a. Under **Name and tags**, for **Name**, enter **`ec2-database-connect`**.

   b. Under **Application and OS Images (Amazon Machine Image)**, choose **Amazon Linux**, and then choose the **Amazon Linux 2023 AMI**. Keep the default selections for the other choices.



   c. Under **Instance type**, choose **t2.micro**.

   d. Under **Key pair (login)**, choose a **Key pair name** to use an existing key pair. To create a new key pair for the Amazon EC2 instance, choose **Create new key pair** and then use the **Create key pair** window to create it.

For more information about creating a new key pair, see Create a key pair in the *Amazon EC2 User Guide*.

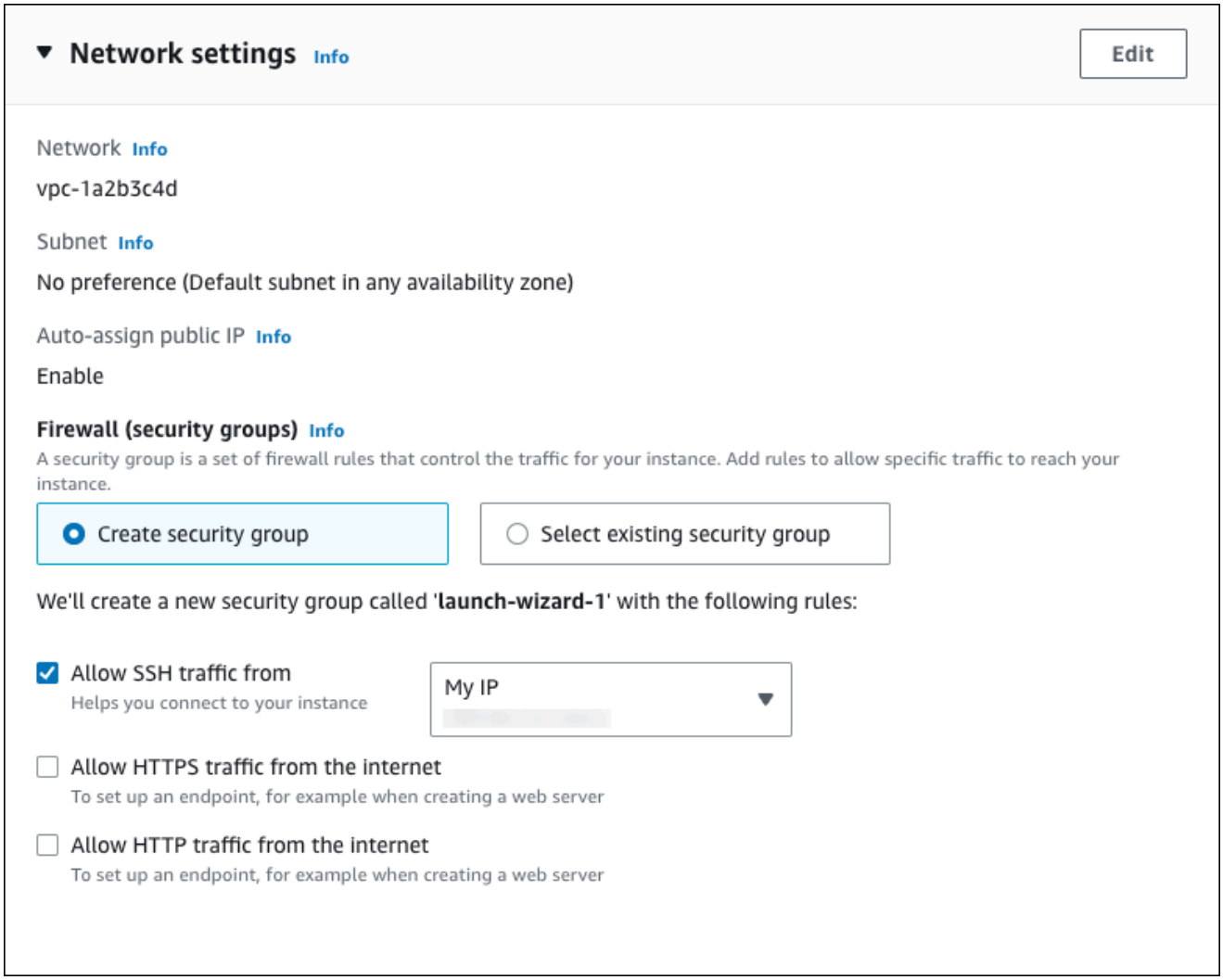
e. For **Allow SSH traffic** in **Network settings**, choose the source of SSH connections to the EC2 instance.

You can choose **My IP** if the displayed IP address is correct for SSH connections. Otherwise, you can determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH). To determine your public IP address, in a different browser window or tab, you can use the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, make sure to determine the range of IP addresses used by client computers.

> **Warning**
>
> If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

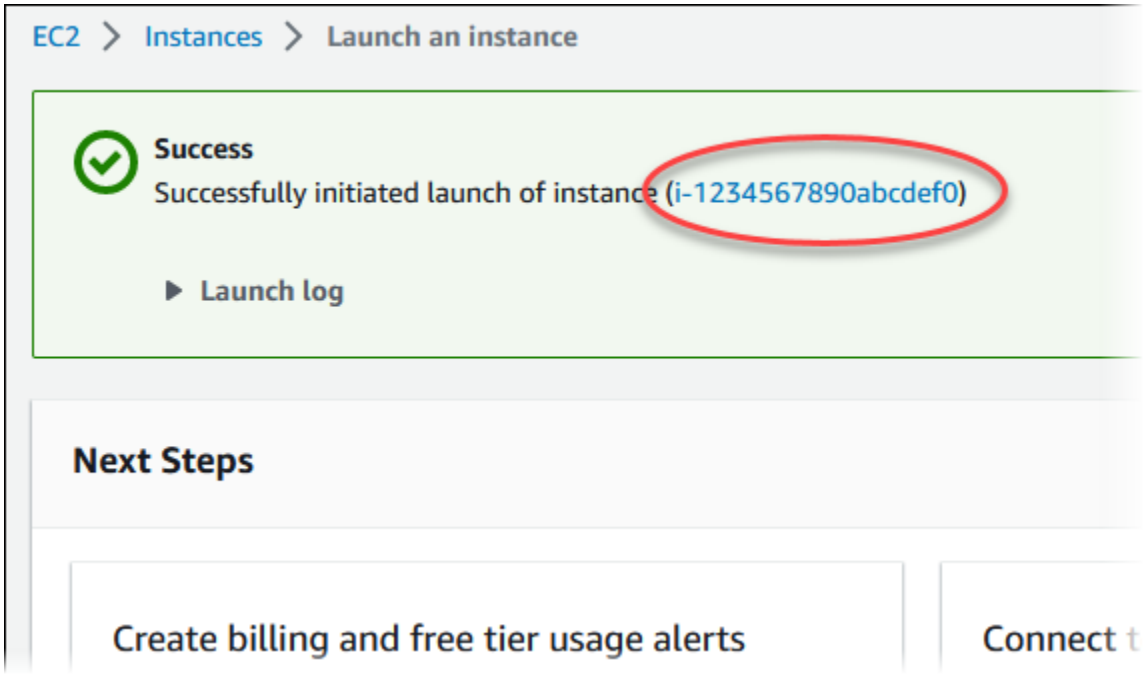
The following image shows an example of the **Network settings** section.



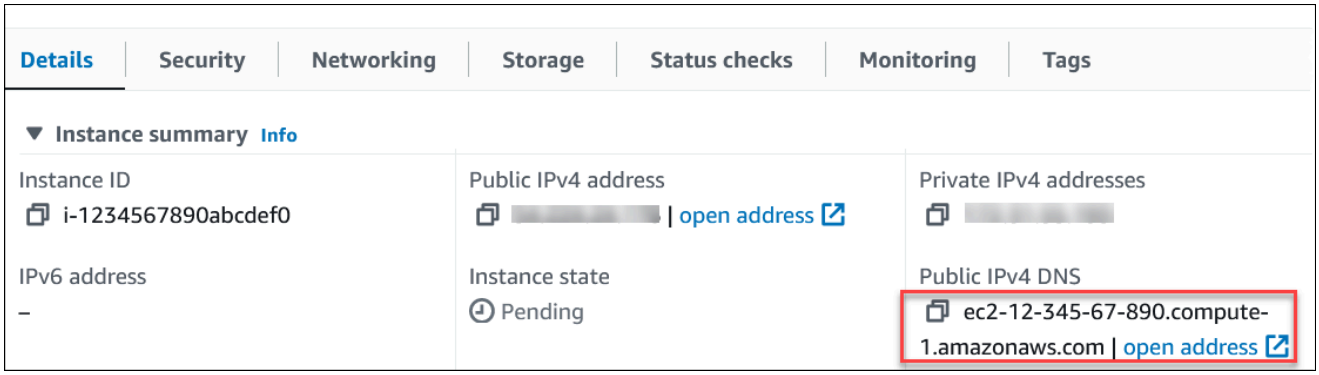
f. Leave the default values for the remaining sections.

g. Review a summary of your EC2 instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

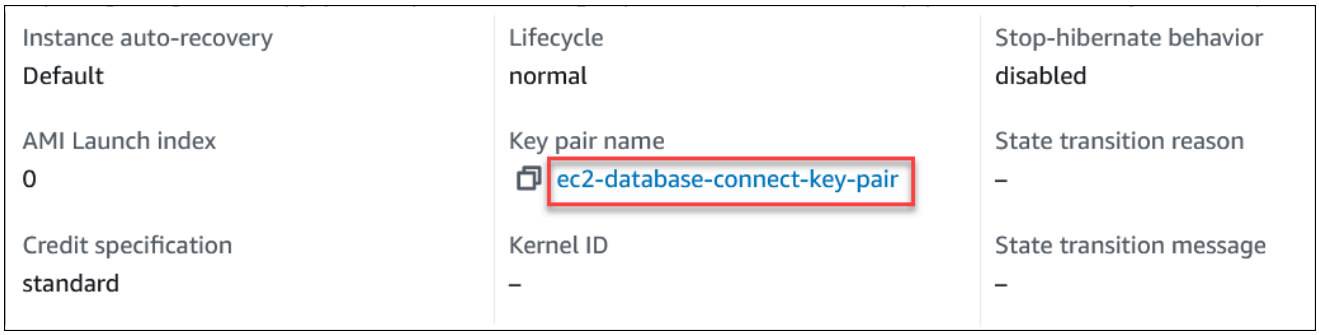
5. On the **Launch Status** page, note the identifier for your new EC2 instance, for example: `i-1234567890abcdef0`.



6. Choose the EC2 instance identifier to open the list of EC2 instances, and then select your EC2 instance.
7. In the **Details** tab, note the following values, which you need when you connect using SSH:
   a. In **Instance summary**, note the value for **Public IPv4 DNS**.



   b. In **Instance details**, note the value for **Key pair name**.



8. Wait until the **Instance state** for your EC2 instance has a status of **Running** before continuing.

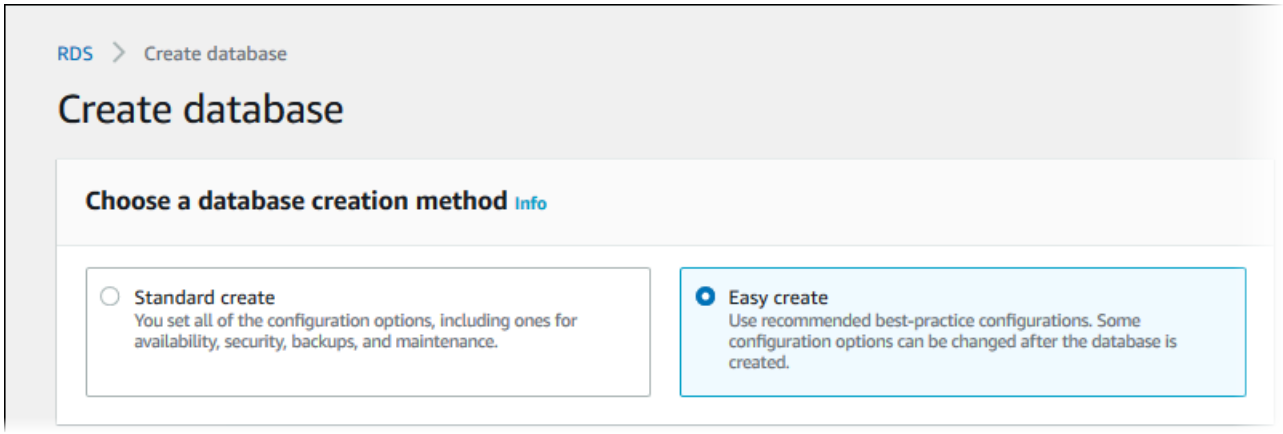
Create a MySQL DB instance

The basic building block of Amazon RDS is the DB instance. This environment is where you run your MySQL databases.

In this example, you use **Easy create** to create a DB instance running the MySQL database engine with a db.t3.micro DB instance class.

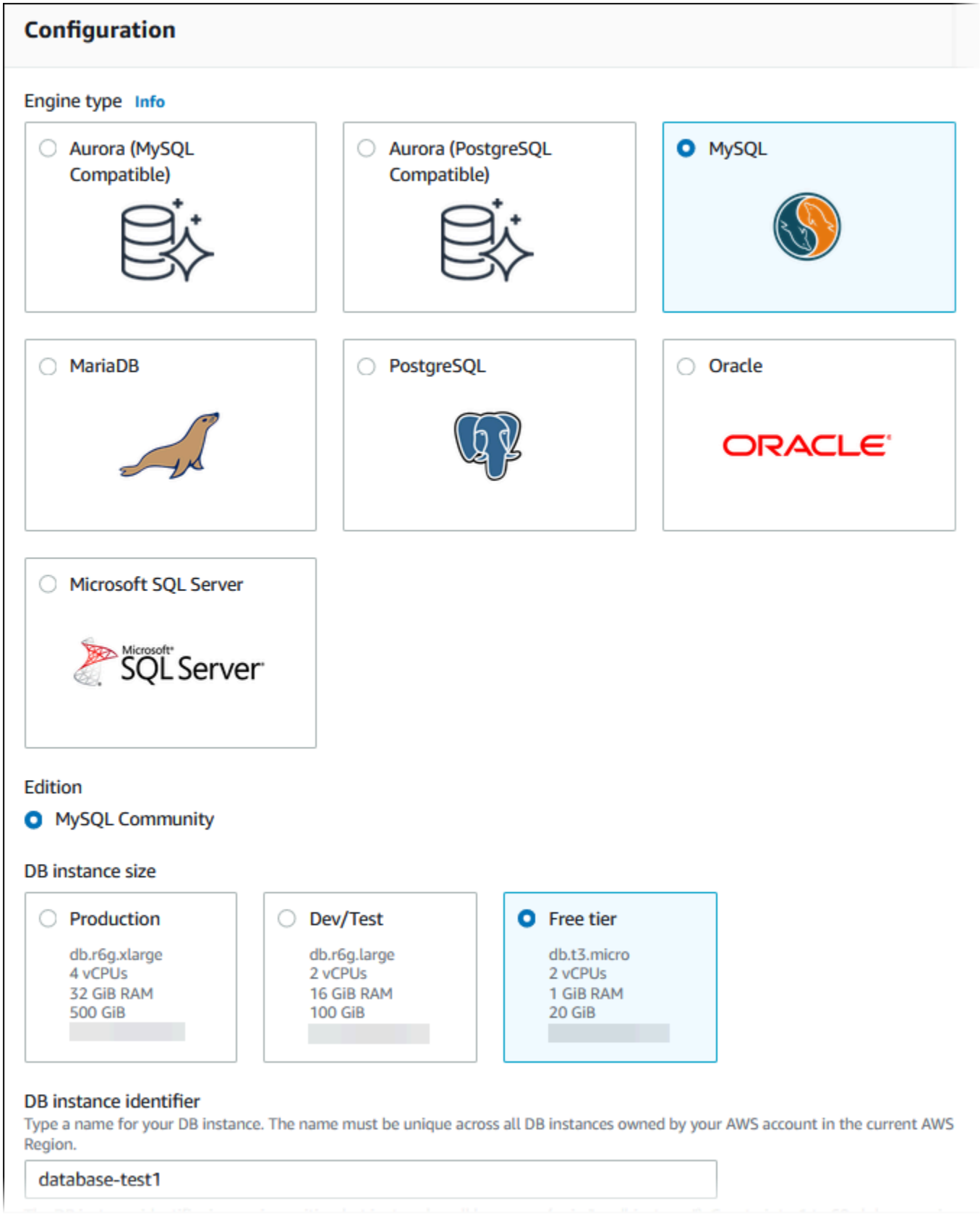
To create a MySQL DB instance with Easy create

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region you used for the EC2 instance previously.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database** and make sure that **Easy create** is chosen.



5. In **Configuration**, choose **MySQL**.
6. For **DB instance size**, choose **Free tier**. **Free tier** appears for free plan accounts. **Sandbox** appears for paid plan accounts.
7. For **DB instance identifier**, enter **`database-test1`**.
8. For **Master username**, enter a name for the master user, or keep the default name.

   The **Create database** page should look similar to the following image. For free plan accounts, **Free tier** appears. For paid plan accounts, **Sandbox** appears.

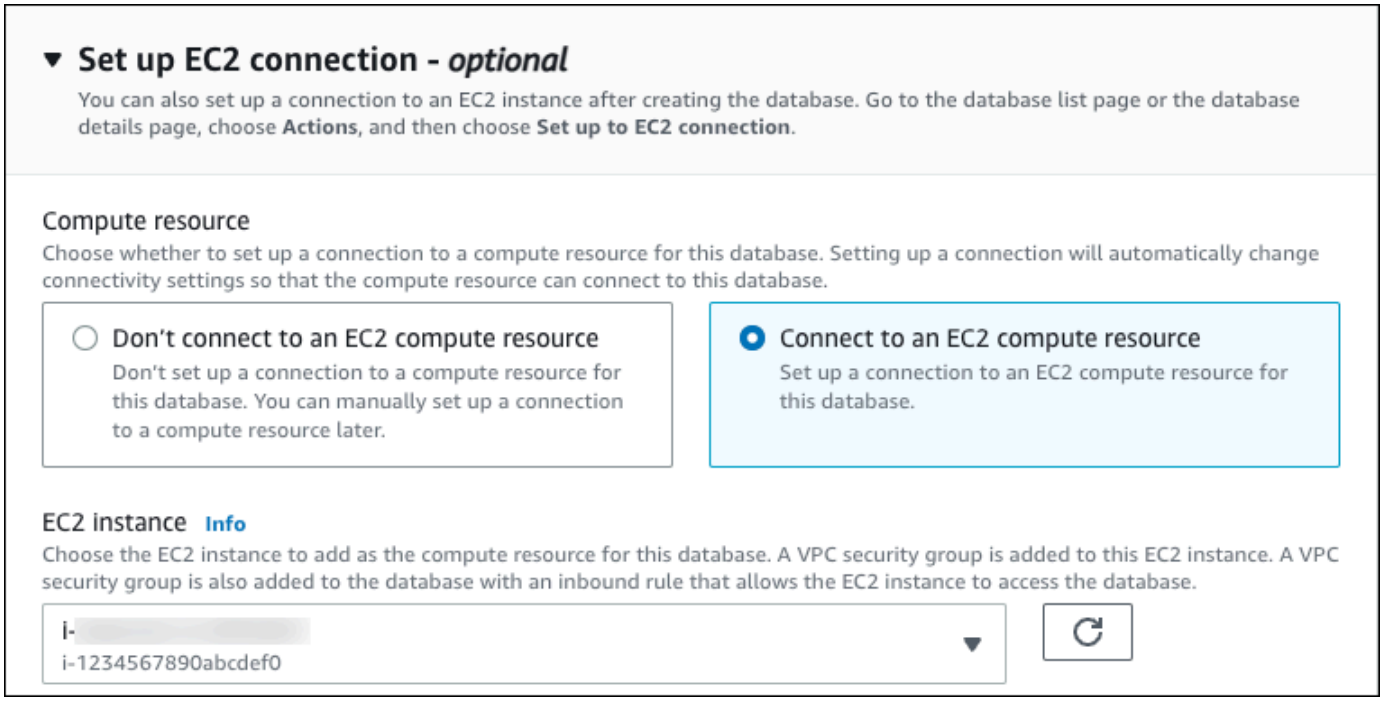

Configuration
Engine type Info
Aurora (MySQL Compatible)
Aurora (PostgreSQL Compatible)
MySQL
MariaDB
PostgreSQL
Oracle
ORACLE
Microsoft SQL Server
Microsoft SQL Server
Edition
MySQL Community
DB instance size
Production
db.r6g.xlarge
4 vCPUs
32 GiB RAM
500 GiB
Dev/Test
db.r6g.large
2 vCPUs
16 GiB RAM
100 GiB
Free tier
db.t3.micro
2 vCPUs
1 GiB RAM
20 GiB
DB instance identifier
Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.
database-test1

9. To use an automatically generated master password for the DB instance, select **Auto generate a password**.

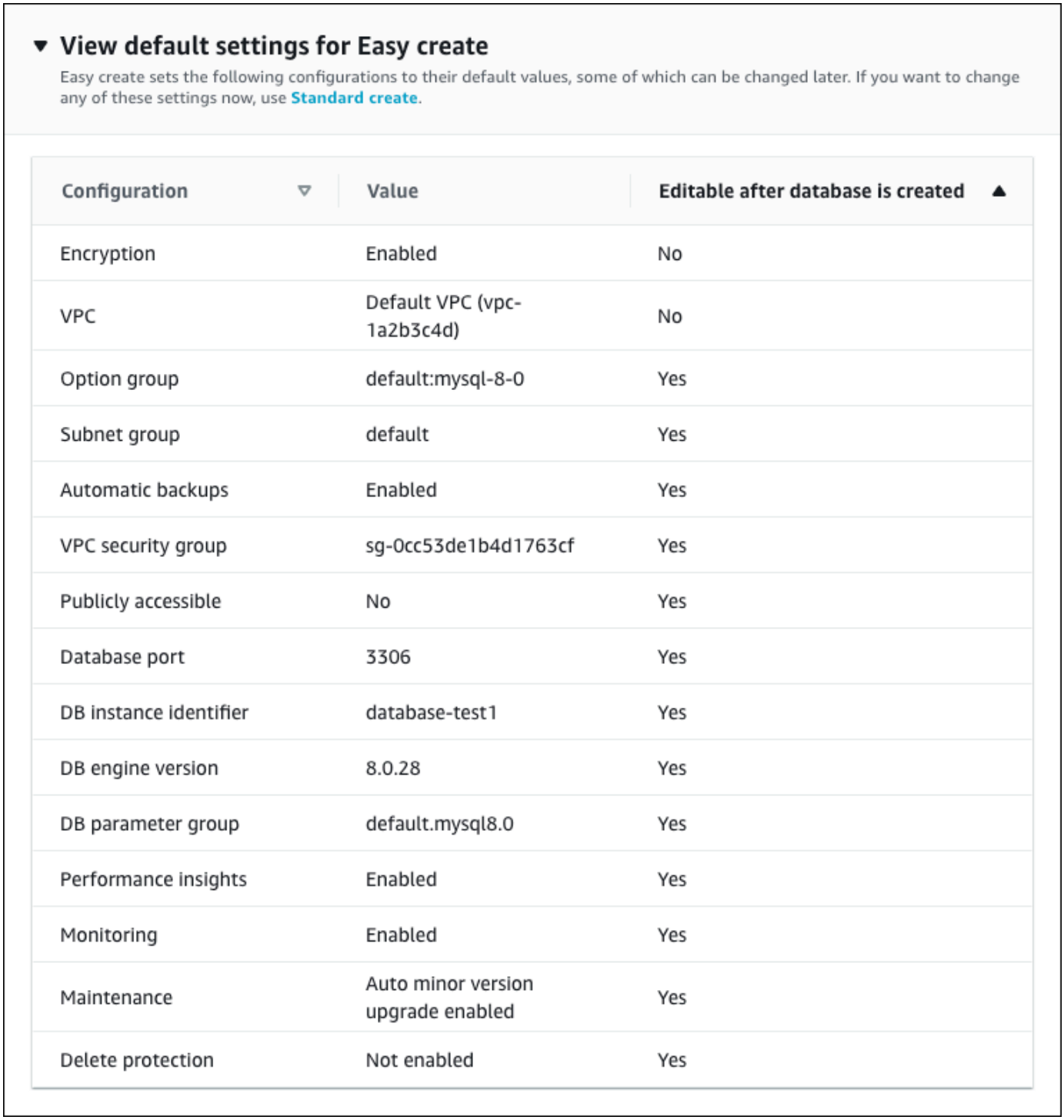
   To enter your master password, make sure **Auto generate a password** is cleared, and then enter the same password in **Master password** and **Confirm password**.

10. To set up a connection with the EC2 instance you created previously, open **Set up EC2 connection - *optional***.

    Select **Connect to an EC2 compute resource**. Choose the EC2 instance you created previously.



11. (Optional) Open **View default settings for Easy create**.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:mysql-8-0	Yes
Subnet group	default	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-0cc53de1b4d1763cf	Yes
Publicly accessible	No	Yes
Database port	3306	Yes
DB instance identifier	database-test1	Yes
DB engine version	8.0.28	Yes
DB parameter group	default.mysql8.0	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

You can examine the default settings used with **Easy create**. The **Editable after database is created** column shows which options you can change after you create the database.

- If a setting has **No** in that column, and you want a different setting, you can use **Standard create** to create the DB instance.

- If a setting has **Yes** in that column, and you want a different setting, you can either use **Standard create** to create the DB instance, or modify the DB instance after you create it to change the setting.

12. Choose **Create database**.

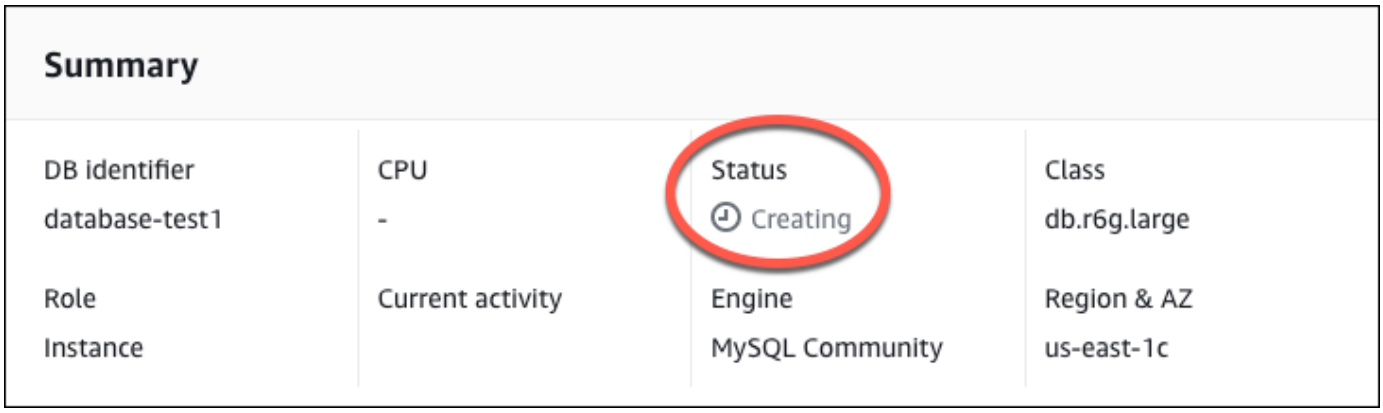
    To view the master username and password for the DB instance, choose **View credential details**.

    You can use the username and password that appears to connect to the DB instance as the master user.

    > **Important**
    >
    > You can't view the master user password again. If you don't record it, you might have to change it.
    > If you need to change the master user password after the DB instance is available, you can modify the DB instance to do so. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

13. In the **Databases** list, choose the name of the new MySQL DB instance to show its details.

    The DB instance has a status of **Creating** until it is ready to use.



    When the status changes to **Available**, you can connect to the DB instance. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new instance is available.

(Optional) Create VPC, EC2 instance, and MySQL instance using CloudFormation

Instead of using the console to create your VPC, EC2 instance, and MySQL instance, you can use CloudFormation to provision AWS resources by treating infrastructure as code. To help you organize your AWS resources into smaller and more manageable units, you can use the CloudFormation nested stack functionality. For more information, see Creating a stack on the CloudFormation console and Working with nested stacks.

Important

CloudFormation is free, but the resources that CloudFormation creates are live. You incur the standard usage fees for these resources until you terminate them. For more information, see RDS for MySQL pricing.

To create your resources using the CloudFormation console, complete the following steps:

- Download the CloudFormation template
- Configure your resources using CloudFormation

Download the CloudFormation template

A CloudFormation template is a JSON or YAML text file that contains the configuration information about the resources you want to create in the stack. This template also creates a VPC and a bastion host for you along with the RDS instance.

To download the template file, open the following link, MySQL CloudFormation template.

In the Github page, click the *Download raw file* button to save the template YAML file.

Configure your resources using CloudFormation

Note

Before starting this process, make sure you have a Key pair for an EC2 instance in your AWS account. For more information, see Amazon EC2 key pairs and Linux instances.

When you use the CloudFormation template, you must select the correct parameters to make sure your resources are created properly. Follow the steps below:

1. Sign in to the AWS Management Console and open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Choose **Create Stack**.
3. In the Specify template section, select **Upload a template file from your computer**, and then choose **Next**.
4. In the **Specify stack details** page, set the following parameters:
   a. Set **Stack name** to **MySQLTestStack**.
   b. Under **Parameters**, set **Availability Zones** by selecting three availability zones.
   c. Under **Linux Bastion Host configuration**, for **Key Name**, select a key pair to login to your EC2 instance.
   d. In **Linux Bastion Host configuration** settings, set the **Permitted IP range** to your IP address. To connect to EC2 instances in your VPC using Secure Shell (SSH), determine your public IP address using the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

      > **Warning**
      >
      > If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

   e. Under **Database General configuration**, set **Database instance class** to **db.t3.micro**.
   f. Set **Database name** to **`database-test1`**.
   g. For **Database master username**, enter a name for the master user.
   h. Set **Manage DB master user password with Secrets Manager** to `false` for this tutorial.
   i. For **Database password**, set a password of your choice. Remember this password for further steps in the tutorial.
   j. Under **Database Storage configuration**, set **Database storage type** to **gp2**.
   k. Under **Database Monitoring configuration**, set **Enable RDS Performance Insights** to false.

l. Leave all other settings as the default values. Click **Next** to continue.

5. In the **Configure stack options** page, leave all the default options. Click **Next** to continue.
6. In the **Review stack** page, select **Submit** after checking the database and Linux bastion host options.

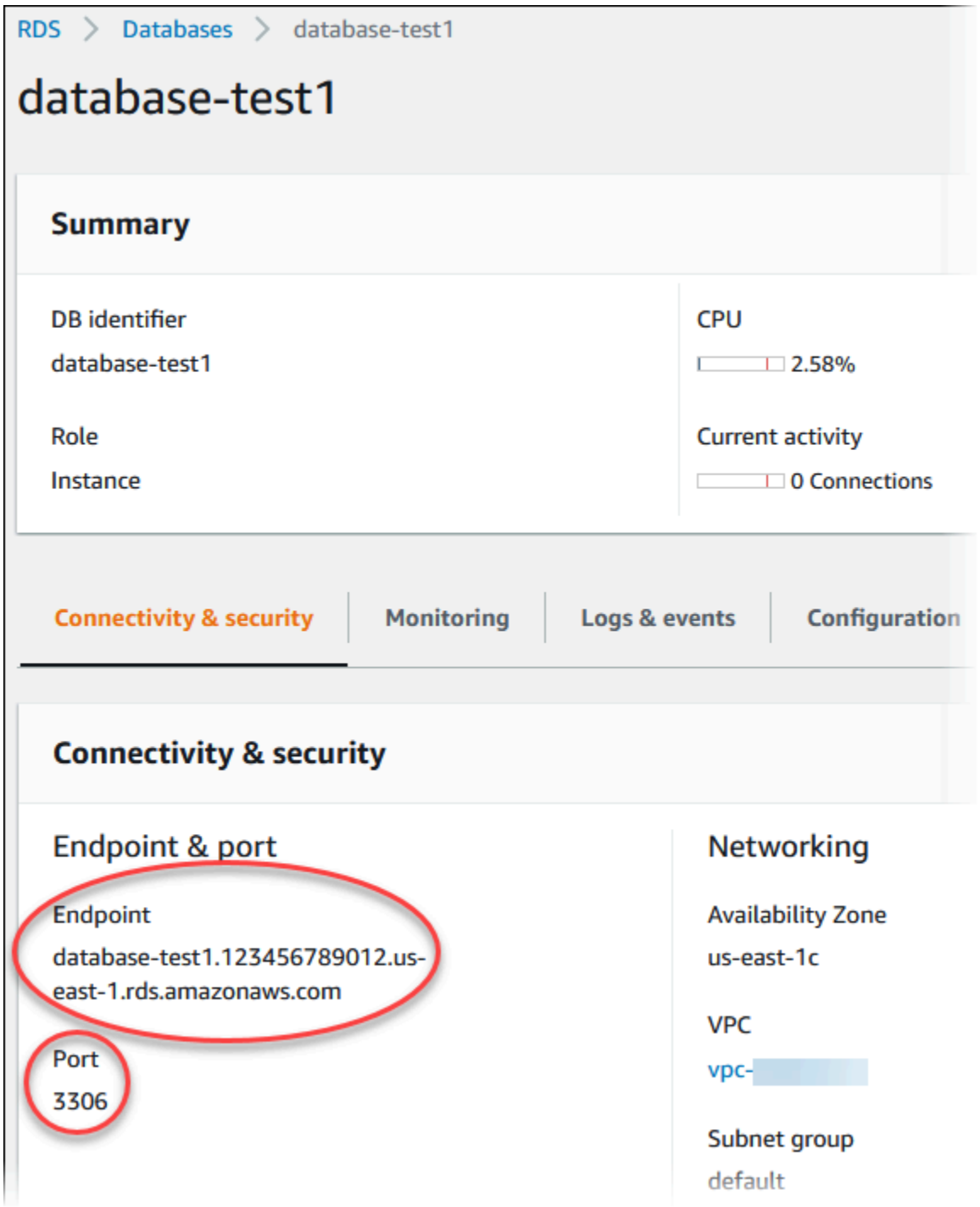
After the stack creation process completes, view the stacks with names *BastionStack* and *RDSNS* to note the information you need to connect to the database. For more information, see Viewing CloudFormation stack data and resources on the AWS Management Console.

Connect to a MySQL DB instance

You can use any standard SQL client application to connect to the DB instance. In this example, you connect to a MySQL DB instance using the mysql command-line client.

To connect to a MySQL DB instance

1. Find the endpoint (DNS name) and port number for your DB instance.
   a. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
   b. In the upper-right corner of the Amazon RDS console, choose the AWS Region for the DB instance.
   c. In the navigation pane, choose **Databases**.
   d. Choose the MySQL DB instance name to display its details.
   e. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



2. Connect to the EC2 instance that you created earlier by following the steps in Connect to your Linux instance in the *Amazon EC2 User Guide*.

   We recommend that you connect to your EC2 instance using SSH. If the SSH client utility is installed on Windows, Linux, or Mac, you can connect to the instance using the following command format:

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

For example, assume that `ec2-database-connect-key-pair.pem` is stored in `/dir1` on Linux, and the public IPv4 DNS for your EC2 instance is `ec2-12-345-678-90.compute-1.amazonaws.com`. Your SSH command would look as follows:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. Get the latest bug fixes and security updates by updating the software on your EC2 instance. To do this, use the following command.

> **Note**
>
> The `-y` option installs the updates without asking for confirmation. To examine updates before installing, omit this option.

```
sudo dnf update -y
```

4. To install the mysql command-line client from MariaDB on Amazon Linux 2023, run the following command:

```
sudo dnf install mariadb105
```

5. Connect to the MySQL DB instance. For example, enter the following command. This action lets you connect to the MySQL DB instance using the MySQL client.

   Substitute the DB instance endpoint (DNS name) for *endpoint*, and substitute the master username that you used for *admin*. Provide the master password that you used when prompted for a password.

```
mysql -h endpoint -P 3306 -u admin -p
```

   After you enter the password for the user, you should see output similar to the following.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 3082
Server version: 8.0.28 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

For more information about connecting to a MySQL DB instance, see Connecting to your MySQL DB instance. If you can't connect to your DB instance, see Can't connect to Amazon RDS DB instance.

For security, it is a best practice to use encrypted connections. Only use an unencrypted MySQL connection when the client and server are in the same VPC and the network is trusted. For information about using encrypted connections, see Connecting to your MySQL DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted).

6. Run SQL commands.

   For example, the following SQL command shows the current date and time:

```
SELECT CURRENT_TIMESTAMP;
```

Delete the EC2 instance and DB instance

After you connect to and explore the sample EC2 instance and DB instance that you created, delete them so you're no longer charged for them.

If you used CloudFormation to create resources, skip this step and go to the next step.

To delete the EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. Select the EC2 instance, and choose **Instance state, Terminate instance**.

4. Choose **Terminate** when prompted for confirmation.

For more information about deleting an EC2 instance, see Terminate your instance in the *Amazon EC2 User Guide*.

To delete the DB instance with no final DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to delete.
4. For **Actions**, choose **Delete**.
5. Clear **Create final snapshot?** and **Retain automated backups**.
6. Complete the acknowledgement and choose **Delete**.

(Optional) Delete the EC2 instance and DB instance created with CloudFormation

If you used CloudFormation to create resources, delete the CloudFormation stack after you connect to and explore the sample EC2 instance and DB instance, so you're no longer charged for them.

To delete the CloudFormation resources

1. Open the CloudFormation console.
2. On the **Stacks** page in the CloudFormationconsole, select the root stack (the stack without the name VPCStack, BastionStack or RDSNS).
3. Choose **Delete**.
4. Select **Delete stack** when prompted for confirmation.

For more information about deleting a stack in CloudFormation, see Deleting a stack on the CloudFormation console in the *AWS CloudFormation User Guide*.

(Optional) Connect your DB instance to a Lambda function

You can also connect your RDS for MySQL DB instance to a Lambda serverless compute resource. Lambda functions allow you to run code without provisioning or managing infrastructure. A

Lambda function also allows you to automatically respond to code execution requests at any scale, from a dozen events a day to hundreds of per second. For more information, see Automatically connecting a Lambda function and a DB instance.

Creating and connecting to an Oracle DB instance

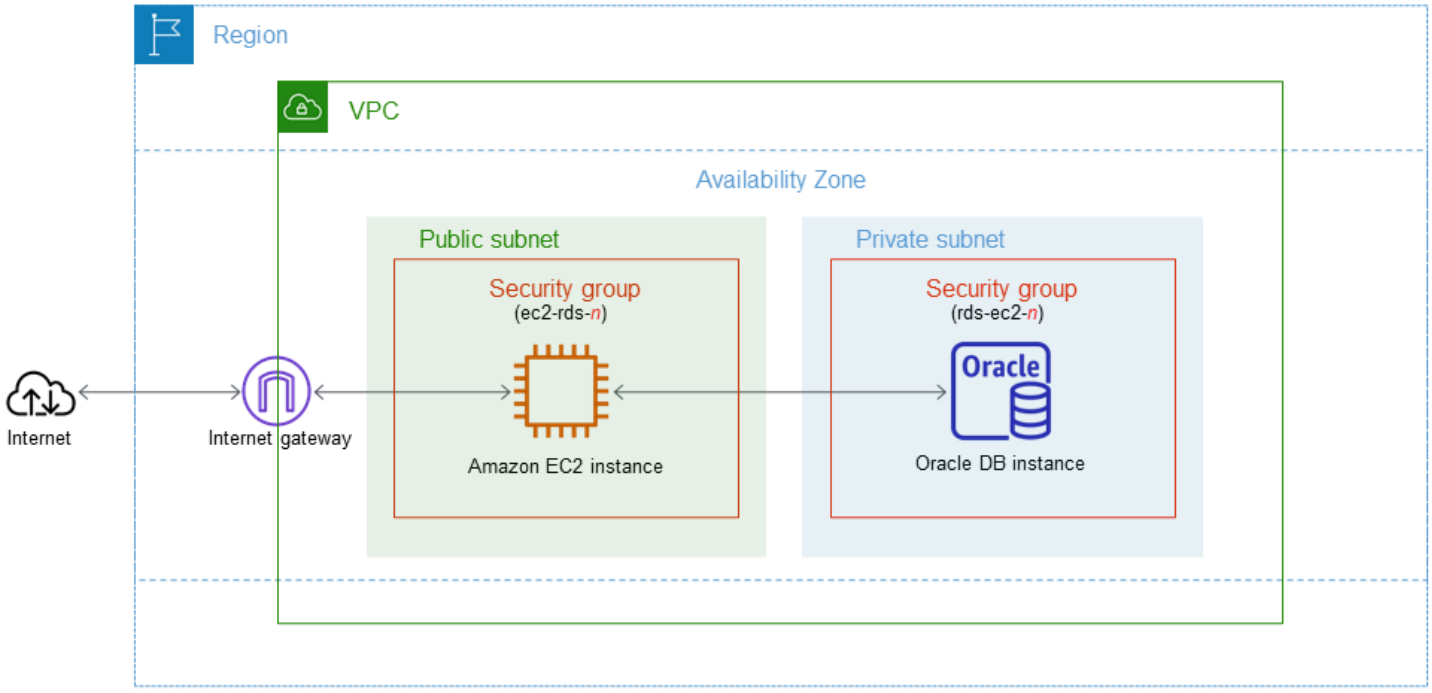
This tutorial creates an EC2 instance and an RDS for Oracle DB instance. The tutorial shows you how to access the DB instance from the EC2 instance using a standard Oracle client. As a best practice, this tutorial creates a private DB instance in a virtual private cloud (VPC). In most cases, other resources in the same VPC, such as EC2 instances, can access the DB instance, but resources outside of the VPC can't access it.

After you complete the tutorial, there is a public and private subnet in each Availability Zone in your VPC. In one Availability Zone, the EC2 instance is in the public subnet, and the DB instance is in the private subnet.

Important

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources you use. You can delete these resources after you complete the tutorial if they are no longer needed.

The following diagram shows the configuration when the tutorial is complete.



This tutorial allows you to create your resources by using one of the following methods:

1. Use the AWS Management Console - Step 2: Create an Oracle DB instance and Step 1: Create an EC2 instance
2. Use CloudFormation to create the database instance and EC2 instance - (Optional) Create VPC, EC2 instance, and Oracle DB instance using CloudFormation

The first method uses **Easy create** to create a private Oracle DB instance with the AWS Management Console. Here, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default settings for the other configuration options.

When you use **Standard create** instead, you can specify more configuration options when you create a DB instance. These options include settings for availability, security, backups, and maintenance. To create a public DB instance, you must use **Standard create**. For information, see Creating an Amazon RDS DB instance.

Topics

- Prerequisites
- Step 1: Create an EC2 instance
- Step 2: Create an Oracle DB instance
- (Optional) Create VPC, EC2 instance, and Oracle DB instance using CloudFormation
- Step 3: Connect your SQL client to an Oracle DB instance
- Step 4: Delete the EC2 instance and DB instance
- (Optional) Delete the EC2 instance and DB instance created with CloudFormation
- (Optional) Connect your DB instance to a Lambda function

Prerequisites

Before you begin, complete the steps in the following sections:

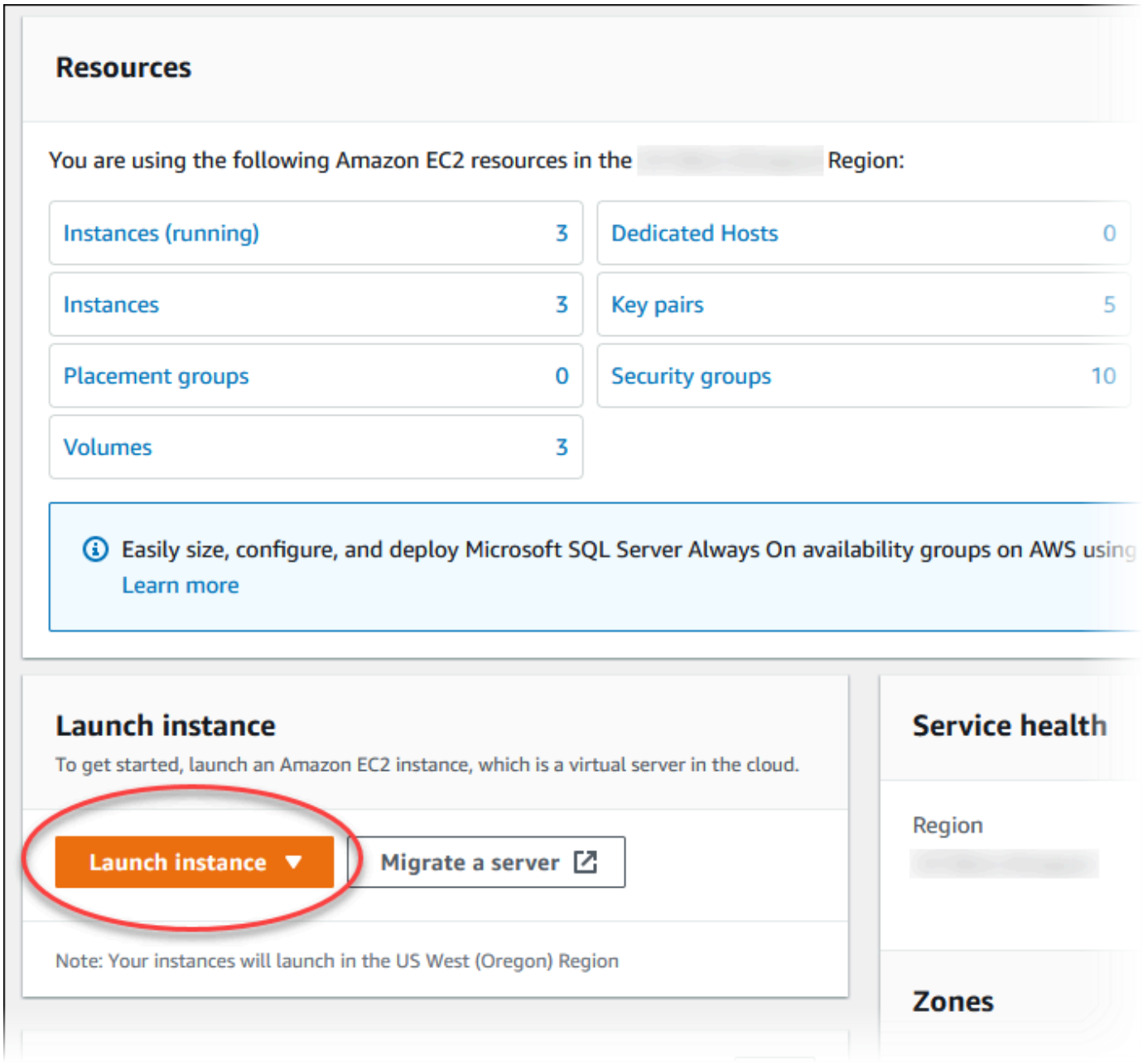
- Sign up for an AWS account
- Create a user with administrative access

Step 1: Create an EC2 instance

Create an Amazon EC2 instance that you will use to connect to your database.

To create an EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you want to create the EC2 instance.
3. Choose **EC2 Dashboard**, and then choose **Launch instance**, as shown in the following image.

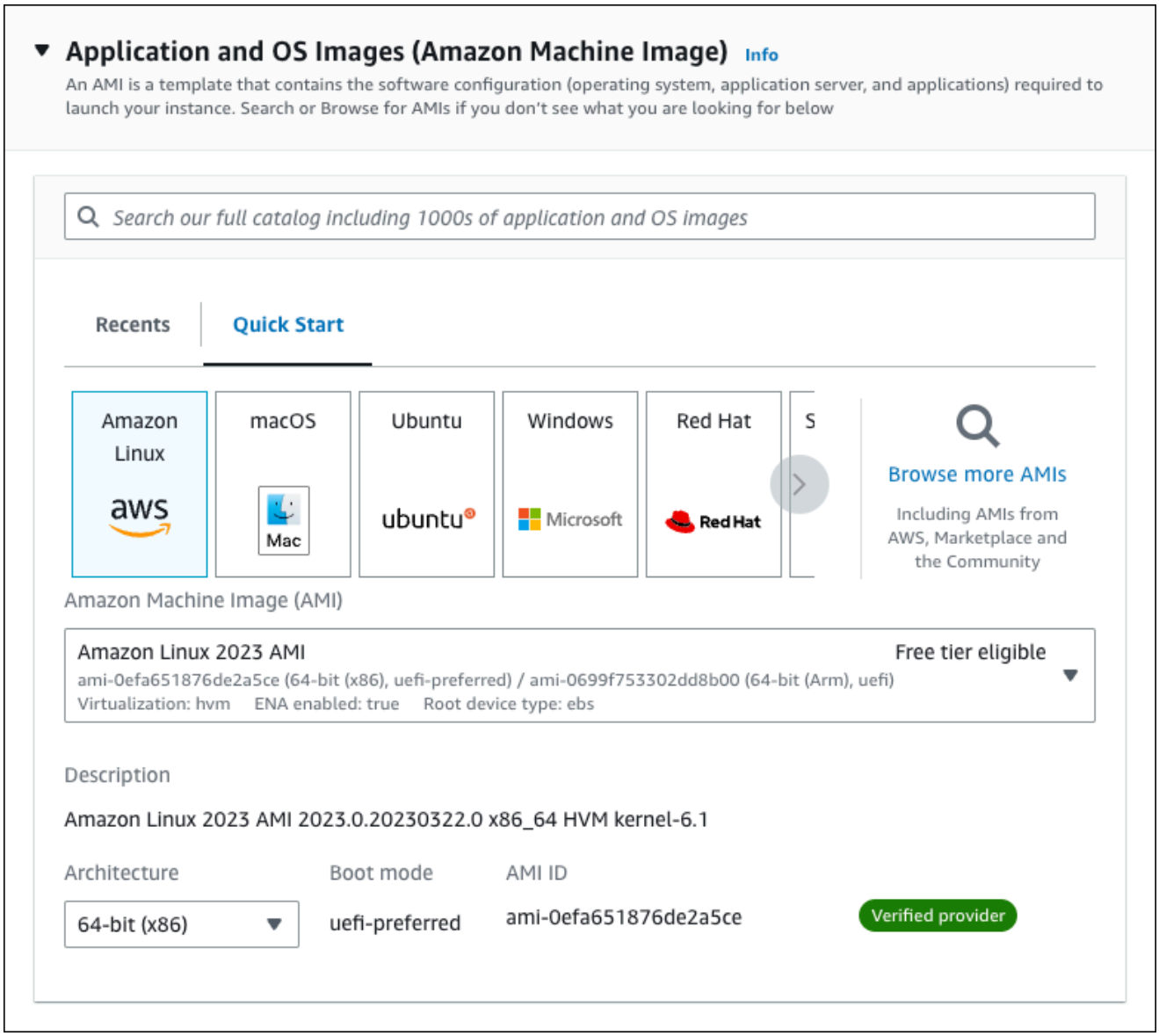


The **Launch an instance** page opens.

4. Choose the following settings on the **Launch an instance** page.

   a. Under **Name and tags**, for **Name**, enter **`ec2-database-connect`**.

   b. Under **Application and OS Images (Amazon Machine Image)**, choose **Amazon Linux**, and then choose the **Amazon Linux 2023 AMI**. Keep the default selections for the other choices.



   c. Under **Instance type**, choose **t2.micro**.

   d. Under **Key pair (login)**, choose a **Key pair name** to use an existing key pair. To create a new key pair for the Amazon EC2 instance, choose **Create new key pair** and then use the **Create key pair** window to create it.

For more information about creating a new key pair, see Create a key pair in the *Amazon EC2 User Guide*.

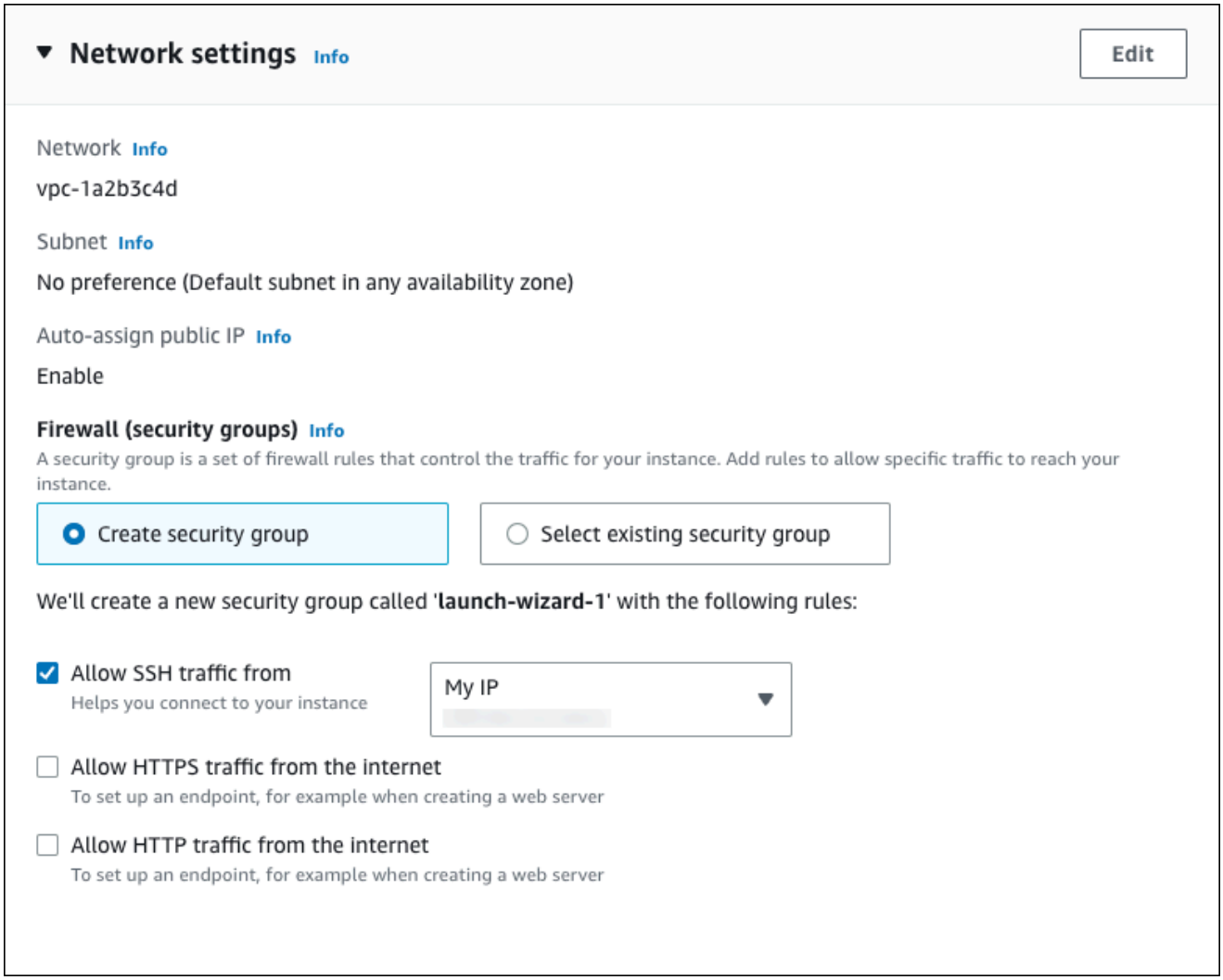
e. For **Allow SSH traffic** in **Network settings**, choose the source of SSH connections to the EC2 instance.

You can choose **My IP** if the displayed IP address is correct for SSH connections. Otherwise, you can determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH). To determine your public IP address, in a different browser window or tab, you can use the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, make sure to determine the range of IP addresses used by client computers.

> **Warning**
>
> If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

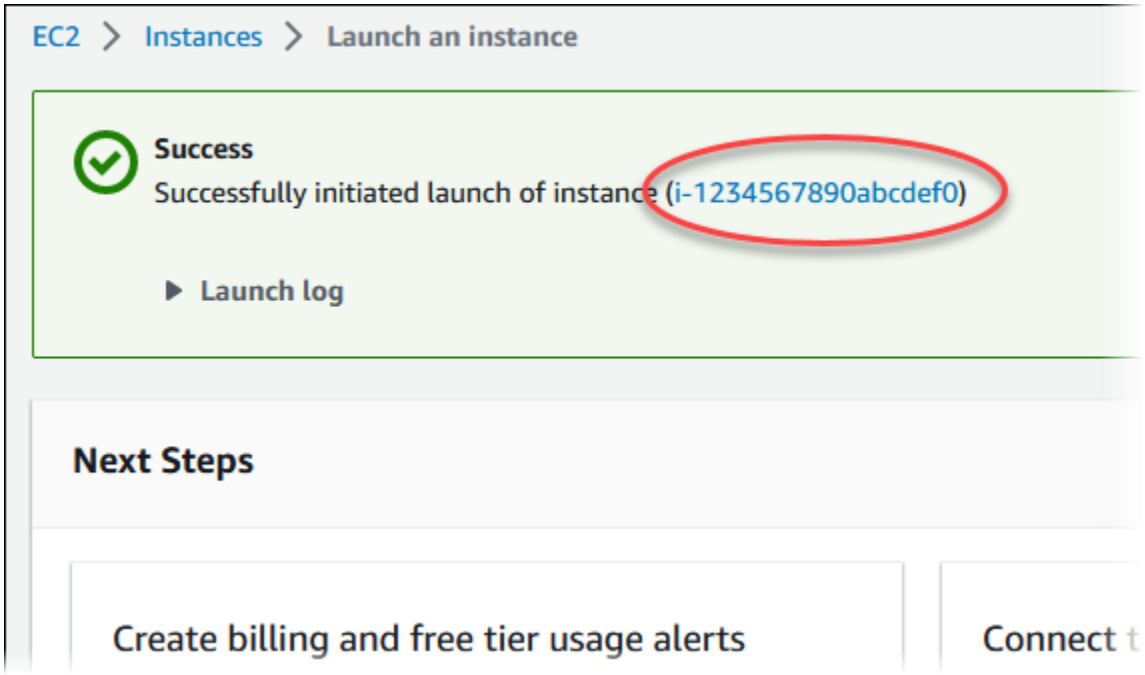
The following image shows an example of the **Network settings** section.



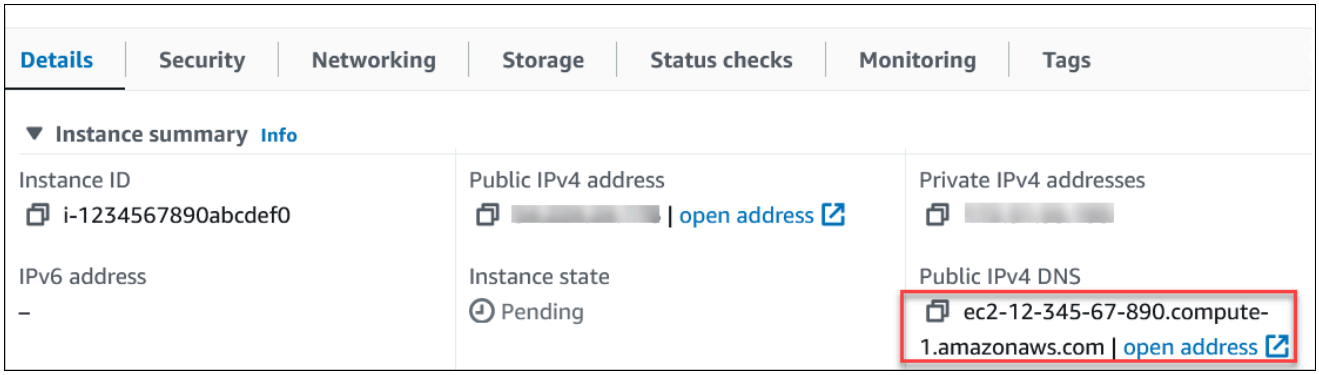
f. Leave the default values for the remaining sections.

g. Review a summary of your EC2 instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

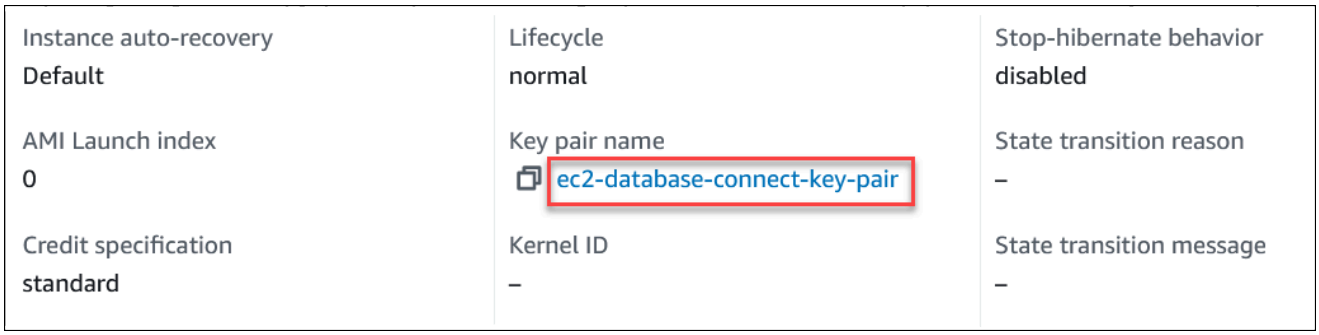
5. On the **Launch Status** page, note the identifier for your new EC2 instance, for example: `i-1234567890abcdef0`.



6. Choose the EC2 instance identifier to open the list of EC2 instances, and then select your EC2 instance.
7. In the **Details** tab, note the following values, which you need when you connect using SSH:
   a. In **Instance summary**, note the value for **Public IPv4 DNS**.



   b. In **Instance details**, note the value for **Key pair name**.



8. Wait until the **Instance state** for your EC2 instance has a status of **Running** before continuing.

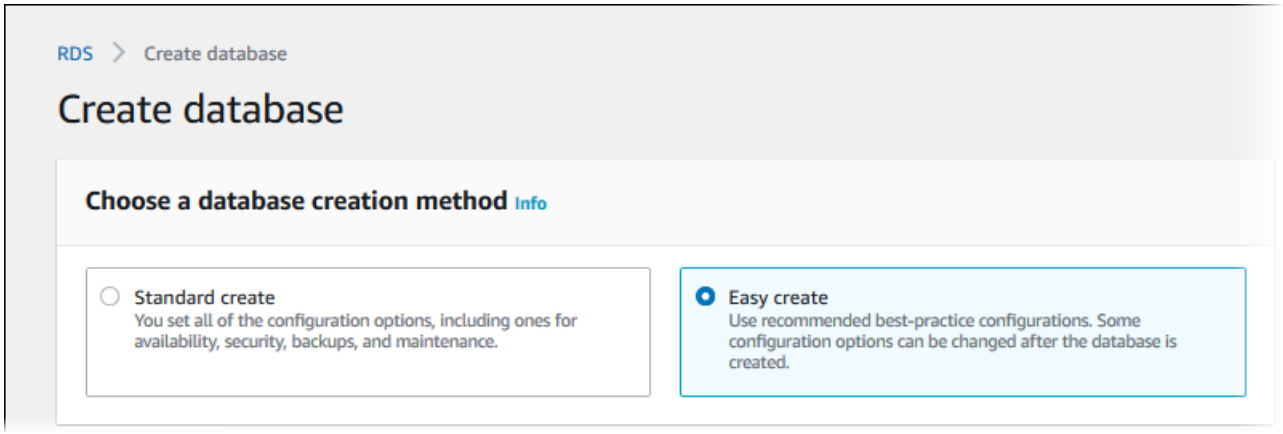
Step 2: Create an Oracle DB instance

The basic building block of Amazon RDS is the DB instance. This environment is where you run your Oracle databases.

In this example, you use **Easy create** to create a DB instance running the Oracle database engine with a db.m5.large DB instance class.

To create an Oracle DB instance with Easy create

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database** and make sure that **Easy create** is chosen.



5. In **Configuration**, choose **Oracle**.
6. For **DB instance size**, choose **Dev/Test**.
7. For **DB instance identifier**, enter **`database-test1`**.
8. For **Master username**, enter a name for the master user, or keep the default name.

   The **Create database** page should look similar to the following image.

Configuration

Engine type Info

- Aurora (MySQL Compatible)
- Aurora (PostgreSQL Compatible)
- MySQL
- MariaDB
- PostgreSQL
- Oracle
- Microsoft SQL Server

Edition

- Oracle Enterprise Edition
  Affordable and full-featured database management system supporting up to 16 vCPUs.
- Oracle Standard Edition Two
  Affordable and full-featured database management system supporting up to 16 vCPUs. Oracle Database Standard Edition Two is a replacement for Standard Edition and Standard Edition One.

DB instance size

- Production
  db.r5.large
  2 vCPUs
  16 GiB RAM
  500 GiB
- Dev/Test
  db.m5.large
  2 vCPUs
  8 GiB RAM
  100 GiB

DB instance identifier

Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

database-test1

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Master username Info

Type a login ID for the master user of your DB instance.

admin

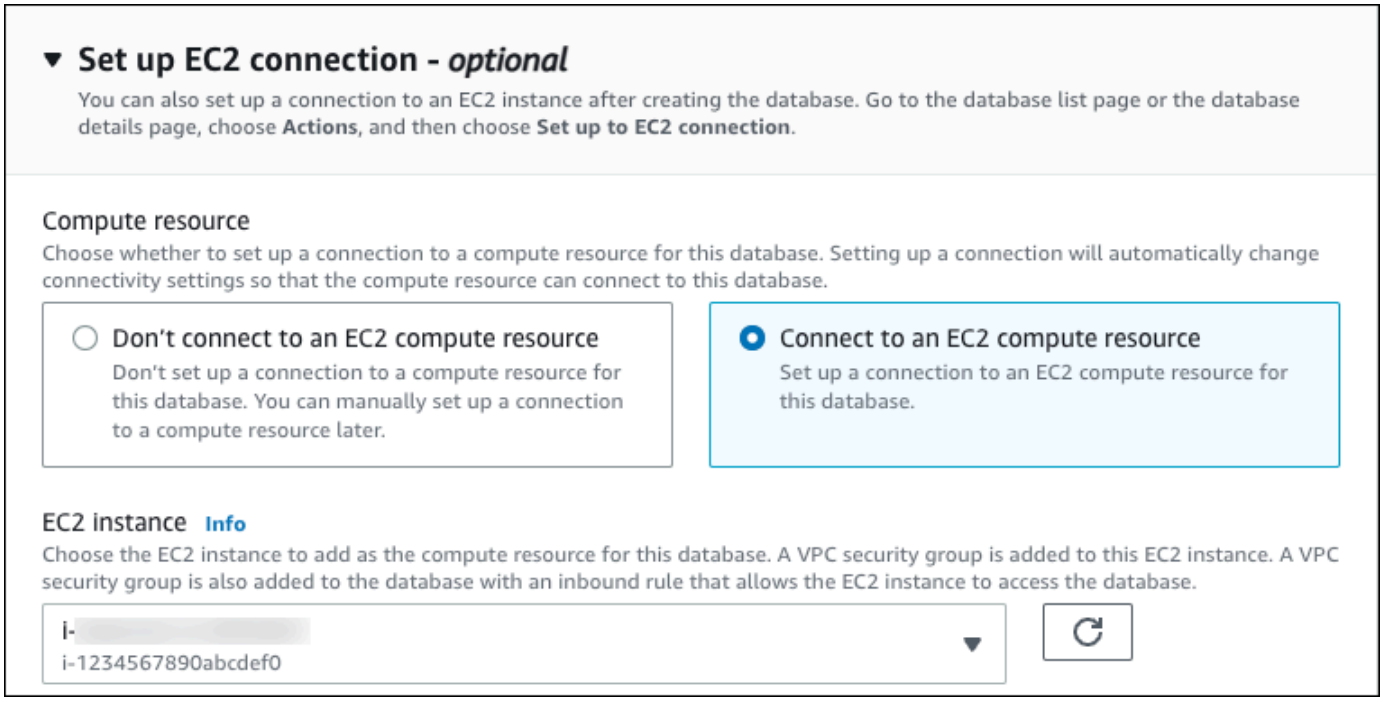
1 to 16 alphanumeric characters. First character must be a letter.

9. To use an automatically generated master password for the DB instance, select **Auto generate a password**.

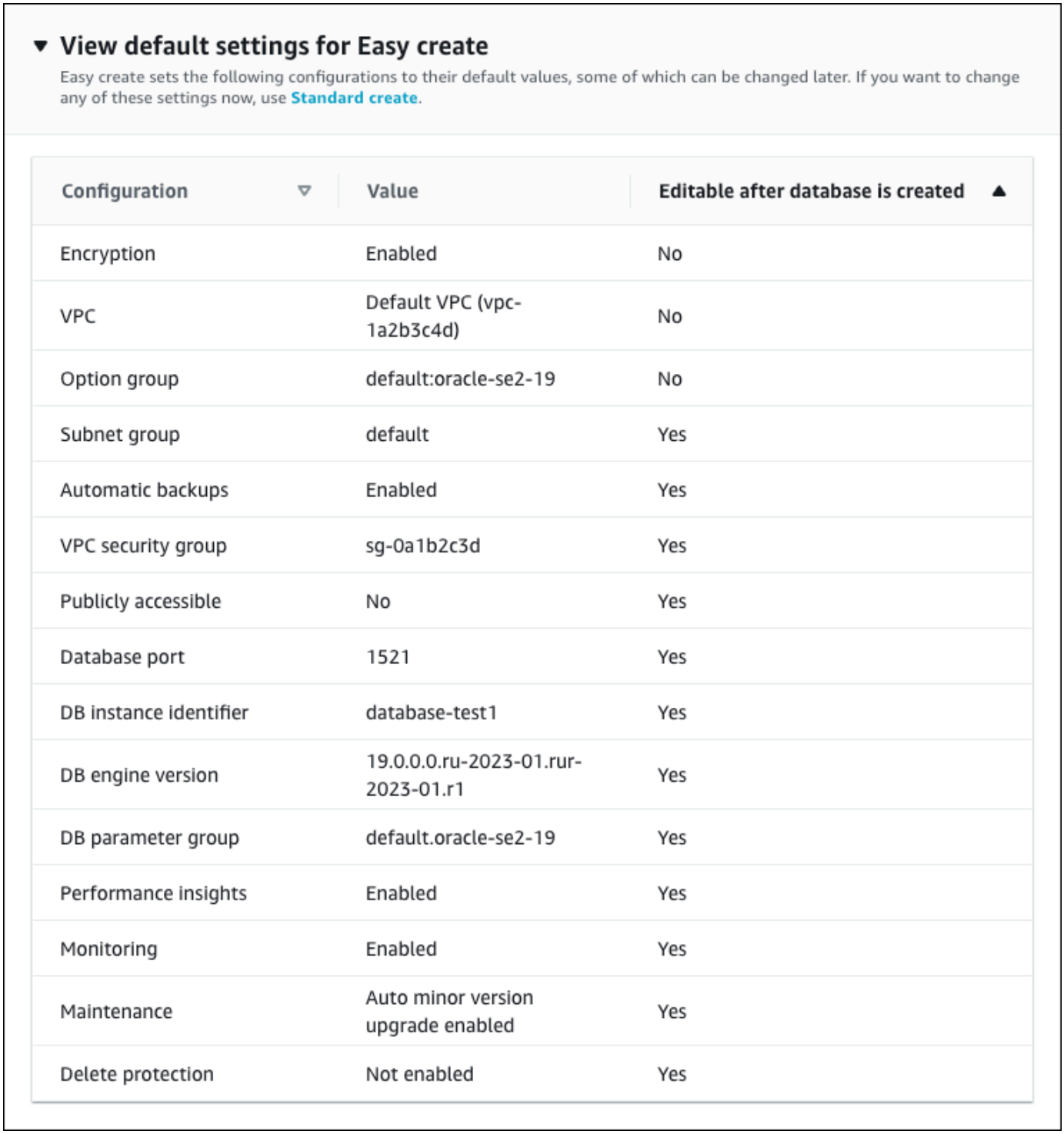
   To enter your master password, make sure **Auto generate a password** is cleared, and then enter the same password in **Master password** and **Confirm password**.

10. To set up a connection with the EC2 instance you created previously, open **Set up EC2 connection - *optional***.

    Select **Connect to an EC2 compute resource**. Choose the EC2 instance you created previously.



11. Open **View default settings for Easy create**.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:oracle-se2-19	No
Subnet group	default	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-0a1b2c3d	Yes
Publicly accessible	No	Yes
Database port	1521	Yes
DB instance identifier	database-test1	Yes
DB engine version	19.0.0.0.ru-2023-01.rur-2023-01.r1	Yes
DB parameter group	default.oracle-se2-19	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

You can examine the default settings used with **Easy create**. The **Editable after database is created** column shows which options you can change after you create the database.

- If a setting has **No** in that column, and you want a different setting, you can use **Standard create** to create the DB instance.

- If a setting has **Yes** in that column, and you want a different setting, you can either use **Standard create** to create the DB instance, or modify the DB instance after you create it to change the setting.

12. Choose **Create database**.

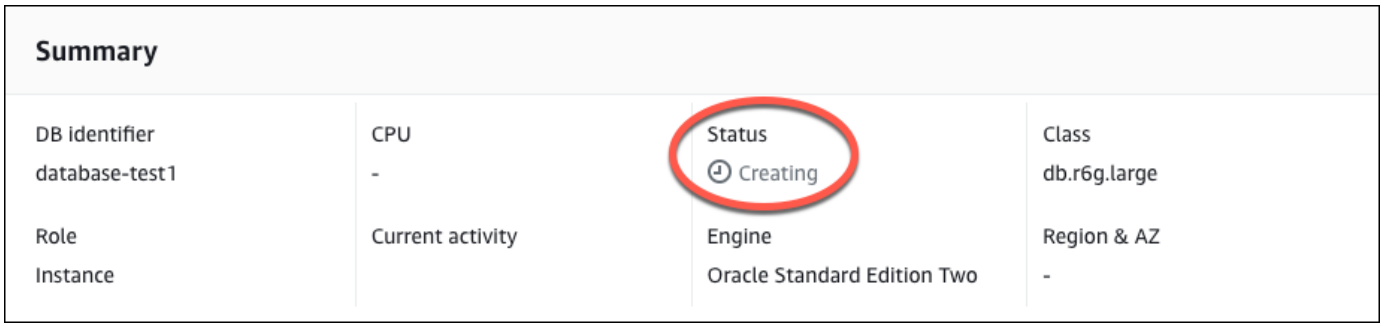
    To view the master username and password for the DB instance, choose **View credential details**.

    You can use the username and password that appears to connect to the DB instance as the master user.

    > **Important**
    >
    > You can't view the master user password again. If you don't record it, you might have to change it.
    > If you need to change the master user password after the DB instance is available, you can modify the DB instance to do so. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

13. In the **Databases** list, choose the name of the new Oracle DB instance to show its details.

    The DB instance has a status of **Creating** until it is ready to use.



    When the status changes to **Available**, you can connect to the DB instance. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new instance is available. While the DB instance is being created, you can move on to the next step and create an EC2 instance.

(Optional) Create VPC, EC2 instance, and Oracle DB instance using CloudFormation

Instead of using the console to create your VPC, EC2 instance, and Oracle DB instance, you can use CloudFormation to provision AWS resources by treating infrastructure as code. To help you organize your AWS resources into smaller and more manageable units, you can use the CloudFormation nested stack functionality. For more information, see Creating a stack on the CloudFormation console and Working with nested stacks.

Important

CloudFormation is free, but the resources that CloudFormation creates are live. You incur the standard usage fees for these resources until you terminate them. For more information, see RDS for Oracle pricing.

To create your resources using the CloudFormation console, complete the following steps:

- Step 1: Download the CloudFormation template
- Step 2: Configure your resources using CloudFormation

Download the CloudFormation template

A CloudFormation template is a JSON or YAML text file that contains the configuration information about the resources you want to create in the stack. This template also creates a VPC and a bastion host for you along with the RDS instance.

To download the template file, open the following link, Oracle CloudFormation template.

In the Github page, click the *Download raw file* button to save the template YAML file.

Configure your resources using CloudFormation

Note

Before starting this process, make sure you have a Key pair for an EC2 instance in your AWS account. For more information, see Amazon EC2 key pairs and Linux instances.

When you use the CloudFormation template, you must select the correct parameters to make sure your resources are created properly. Follow the steps below:

1. Sign in to the AWS Management Console and open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Choose **Create Stack**.
3. In the Specify template section, select **Upload a template file from your computer**, and then choose **Next**.
4. In the **Specify stack details** page, set the following parameters:
   a. Set **Stack name** to **OracleTestStack**.
   b. Under **Parameters**, set **Availability Zones** by selecting three availability zones.
   c. Under **Linux Bastion Host configuration**, for **Key Name**, select a key pair to login to your EC2 instance.
   d. In **Linux Bastion Host configuration** settings, set the **Permitted IP range** to your IP address. To connect to EC2 instances in your VPC using Secure Shell (SSH), determine your public IP address using the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

      > **Warning**
      >
      > If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

   e. Under **Database General configuration**, set **Database instance class** to **db.t3.micro**.
   f. Set **Database name** to **`database-test1`**.
   g. For **Database master username**, enter a name for the master user.
   h. Set **Manage DB master user password with Secrets Manager** to `false` for this tutorial.
   i. For **Database password**, set a password of your choice. Remember this password for further steps in the tutorial.
   j. Under **Database Storage configuration**, set **Database storage type** to **gp2**.
   k. Under **Database Monitoring configuration**, set **Enable RDS Performance Insights** to false.

l. Leave all other settings as the default values. Click **Next** to continue.

5. In the **Configure stack options** page, leave all the default options. Click **Next** to continue.
6. In the **Review stack** page, select **Submit** after checking the database and Linux bastion host options.

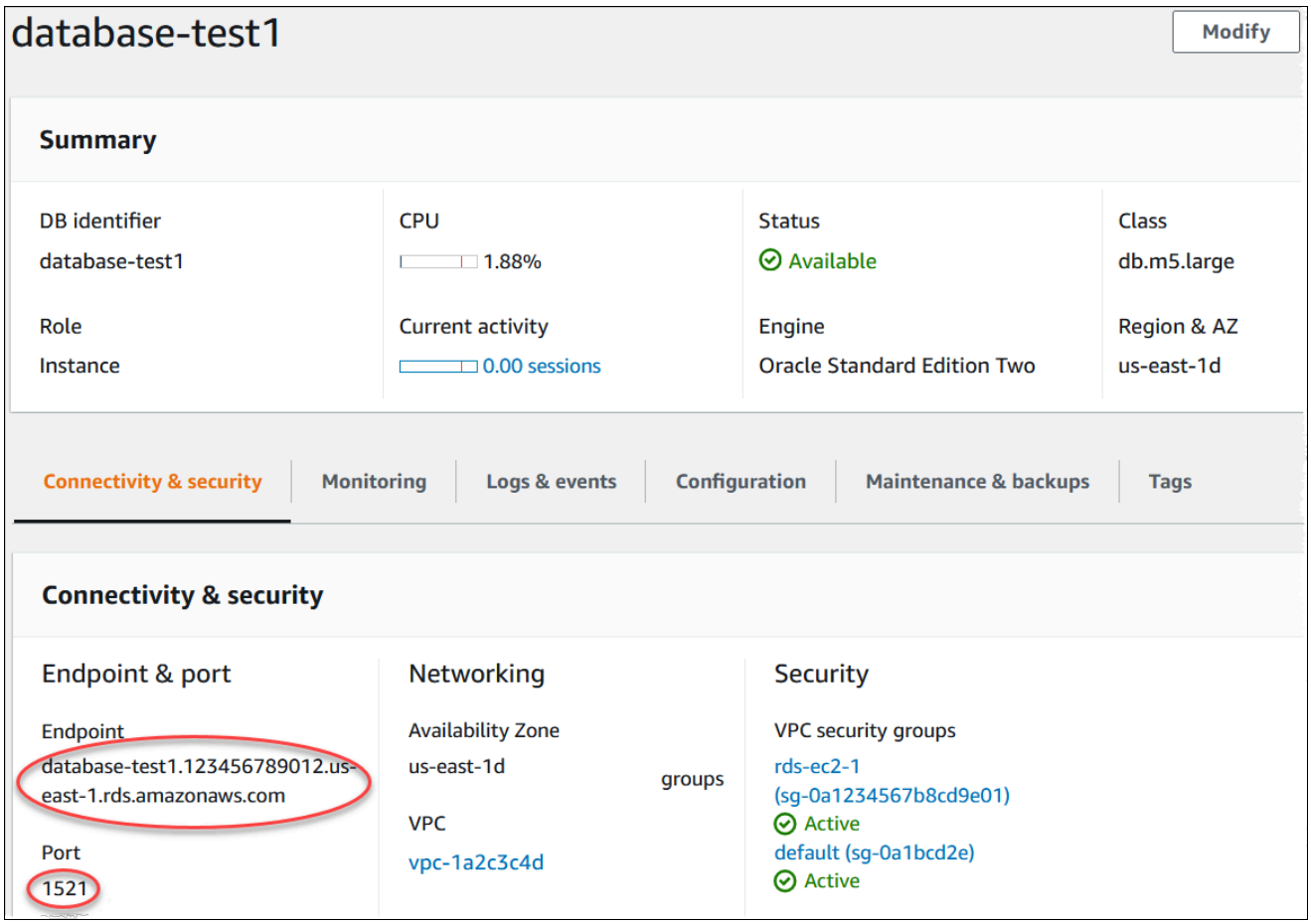
After the stack creation process completes, view the stacks with names *BastionStack* and *RDSNS* to note the information you need to connect to the database. For more information, see Viewing CloudFormation stack data and resources on the AWS Management Console.

Step 3: Connect your SQL client to an Oracle DB instance

You can use any standard SQL client application to connect to your DB instance. In this example, you connect to an Oracle DB instance using the Oracle command-line client.

To connect to an Oracle DB instance

1. Find the endpoint (DNS name) and port number for your DB instance.
   a. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
   b. In the upper-right corner of the Amazon RDS console, choose the AWS Region for the DB instance.
   c. In the navigation pane, choose **Databases**.
   d. Choose the Oracle DB instance name to display its details.
   e. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



2. Connect to the EC2 instance that you created earlier by following the steps in Connect to your Linux instance in the *Amazon EC2 User Guide*.

   We recommend that you connect to your EC2 instance using SSH. If the SSH client utility is installed on Windows, Linux, or Mac, you can connect to the instance using the following command format:

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   For example, assume that `ec2-database-connect-key-pair.pem` is stored in `/dir1` on Linux, and the public IPv4 DNS for your EC2 instance is `ec2-12-345-678-90.compute-1.amazonaws.com`. Your SSH command would look as follows:

   ```
   ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
   user@ec2-12-345-678-90.compute-1.amazonaws.com
   ```

3. Get the latest bug fixes and security updates by updating the software on your EC2 instance. To do so, use the following command.

   > **Note**
   >
   > The `-y` option installs the updates without asking for confirmation. To examine updates before installing, omit this option.

   ```
   sudo dnf update -y
   ```

4. In a web browser, go to https://www.oracle.com/database/technologies/instant-client/linux-x86-64-downloads.html.
5. For the latest database version that appears on the web page, copy the .rpm links (not the .zip links) for the Instant Client Basic Package and SQL*Plus Package. For example, the following links are for Oracle Database version 21.9:

   - https://download.oracle.com/otn_software/linux/instantclient/219000/oracle-instantclient-basic-21.9.0.0.0-1.el8.x86_64.rpm
   - https://download.oracle.com/otn_software/linux/instantclient/219000/oracle-instantclient-sqlplus-21.9.0.0.0-1.el8.x86_64.rpm

6. In your SSH session, run the `wget` command to the download the .rpm files from the links that you obtained in the previous step. The following example downloads the .rpm files for Oracle Database version 21.9:

   ```
   wget https://download.oracle.com/otn_software/linux/instantclient/219000/oracle-instantclient-basic-21.9.0.0.0-1.el8.x86_64.rpm
   wget https://download.oracle.com/otn_software/linux/instantclient/219000/oracle-instantclient-sqlplus-21.9.0.0.0-1.el8.x86_64.rpm
   ```

7. Install the packages by running the `dnf` command as follows:

   ```
   sudo dnf install oracle-instantclient-*.rpm
   ```

8. Start SQL*Plus and connect to the Oracle DB instance. For example, enter the following command.

Substitute the DB instance endpoint (DNS name) for *oracle-db-instance-endpoint* and substitute the master user name that you used for *admin*. When you use **Easy create** for Oracle, the database name is DATABASE. Provide the master password that you used when prompted for a password.

```
sqlplus admin@oracle-db-instance-endpoint:1521/DATABASE
```

After you enter the password for the user, you should see output similar to the following.

```
SQL*Plus: Release 21.0.0.0.0 - Production on Wed Mar 1 16:41:28 2023
Version 21.9.0.0.0

Copyright (c) 1982, 2022, Oracle.  All rights reserved.

Enter password:
Last Successful login time: Wed Mar 01 2023 16:30:52 +00:00

Connected to:
Oracle Database 19c Standard Edition 2 Release 19.0.0.0.0 - Production
Version 19.18.0.0.0

SQL>
```

For more information about connecting to an RDS for Oracle DB instance, see Connecting to your Oracle DB instance. If you can't connect to your DB instance, see Can't connect to Amazon RDS DB instance.

For security, it is a best practice to use encrypted connections. Only use an unencrypted Oracle connection when the client and server are in the same VPC and the network is trusted. For information about using encrypted connections, see Securing Oracle DB instance connections.

9. Run SQL commands.

   For example, the following SQL command shows the current date:

```
SELECT SYSDATE FROM DUAL;
```

Step 4: Delete the EC2 instance and DB instance

After you connect to and explore the sample EC2 instance and DB instance that you created, delete them so you're no longer charged for them.

If you used CloudFormation to create resources, skip this step and go to the next step.

To delete the EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. Select the EC2 instance, and choose **Instance state, Terminate instance**.
4. Choose **Terminate** when prompted for confirmation.

For more information about deleting an EC2 instance, see Terminate your instance in the *Amazon EC2 User Guide*.

To delete the DB instance with no final DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to delete.
4. For **Actions**, choose **Delete**.
5. Clear **Create final snapshot?** and **Retain automated backups**.
6. Complete the acknowledgement and choose **Delete**.

(Optional) Delete the EC2 instance and DB instance created with CloudFormation

If you used CloudFormation to create resources, delete the CloudFormation stack after you connect to and explore the sample EC2 instance and DB instance, so you're no longer charged for them.

To delete the CloudFormation resources

1. Open the CloudFormation console.

2. On the **Stacks** page in the CloudFormationconsole, select the root stack (the stack without the name VPCStack, BastionStack or RDSNS).
3. Choose **Delete**.
4. Select **Delete stack** when prompted for confirmation.

For more information about deleting a stack in CloudFormation, see Deleting a stack on the CloudFormation console in the *AWS CloudFormation User Guide*.

(Optional) Connect your DB instance to a Lambda function

You can also connect your RDS for Oracle DB instance to a Lambda serverless compute resource. Lambda functions allow you to run code without provisioning or managing infrastructure. A Lambda function also allows you to automatically respond to code execution requests at any scale, from a dozen events a day to hundreds of per second. For more information, see Automatically connecting a Lambda function and a DB instance.

Creating and connecting to a PostgreSQL DB instance

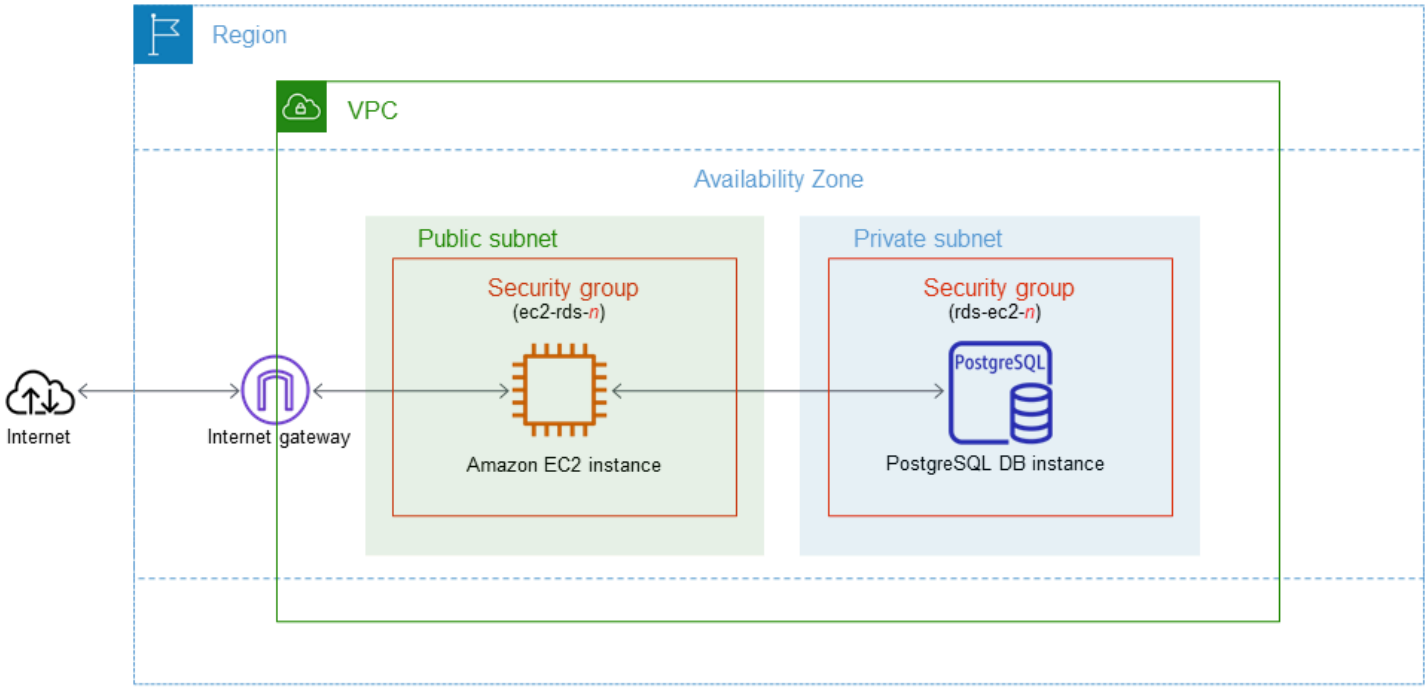
This tutorial creates an EC2 instance and an RDS for PostgreSQL DB instance. The tutorial shows you how to access the DB instance from the EC2 instance using a standard PostgreSQL client. As a best practice, this tutorial creates a private DB instance in a virtual private cloud (VPC). In most cases, other resources in the same VPC, such as EC2 instances, can access the DB instance, but resources outside of the VPC can't access it.

After you complete the tutorial, there is a public and private subnet in each Availability Zone in your VPC. In one Availability Zone, the EC2 instance is in the public subnet, and the DB instance is in the private subnet.

Important

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources you use. You can delete these resources after you complete the tutorial if they are no longer needed.

The following diagram shows the configuration when the tutorial is complete.



This tutorial allows you to create your resources by using one of the following methods:

1. Use the AWS Management Console - Create an EC2 instance and Create a PostgreSQL DB instance
2. Use CloudFormation to create the database instance and EC2 instance - (Optional) Create VPC, EC2 instance, and PostgreSQL instance using CloudFormation

The first method uses **Easy create** to create a private PostgreSQL DB instance with the AWS Management Console. Here, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default settings for the other configuration options.

When you use **Standard create** instead, you can specify more configuration options when you create a DB instance. These options include settings for availability, security, backups, and maintenance. To create a public DB instance, you must use **Standard create**. For information, see Creating an Amazon RDS DB instance.

Topics

- Prerequisites
- Create an EC2 instance
- Create a PostgreSQL DB instance
- (Optional) Create VPC, EC2 instance, and PostgreSQL instance using CloudFormation
- Connect to a PostgreSQL DB instance
- Delete the EC2 instance and DB instance
- (Optional) Delete the EC2 instance and DB instance created with CloudFormation
- (Optional) Connect your DB instance to a Lambda function

Prerequisites

Before you begin, complete the steps in the following sections:

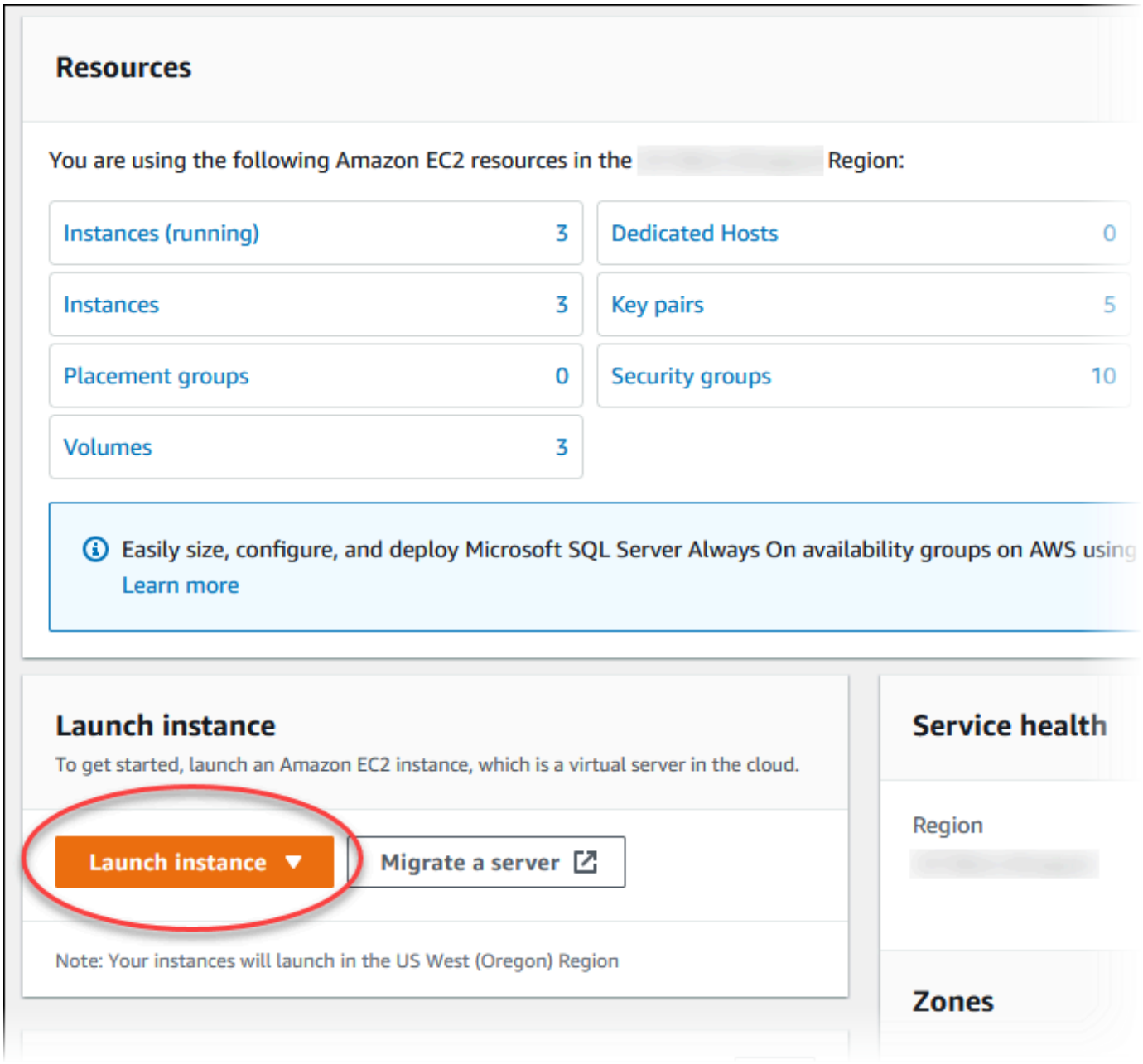
- Sign up for an AWS account
- Create a user with administrative access

Create an EC2 instance

Create an Amazon EC2 instance that you will use to connect to your database.

To create an EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you want to create the EC2 instance.
3. Choose **EC2 Dashboard**, and then choose **Launch instance**, as shown in the following image.

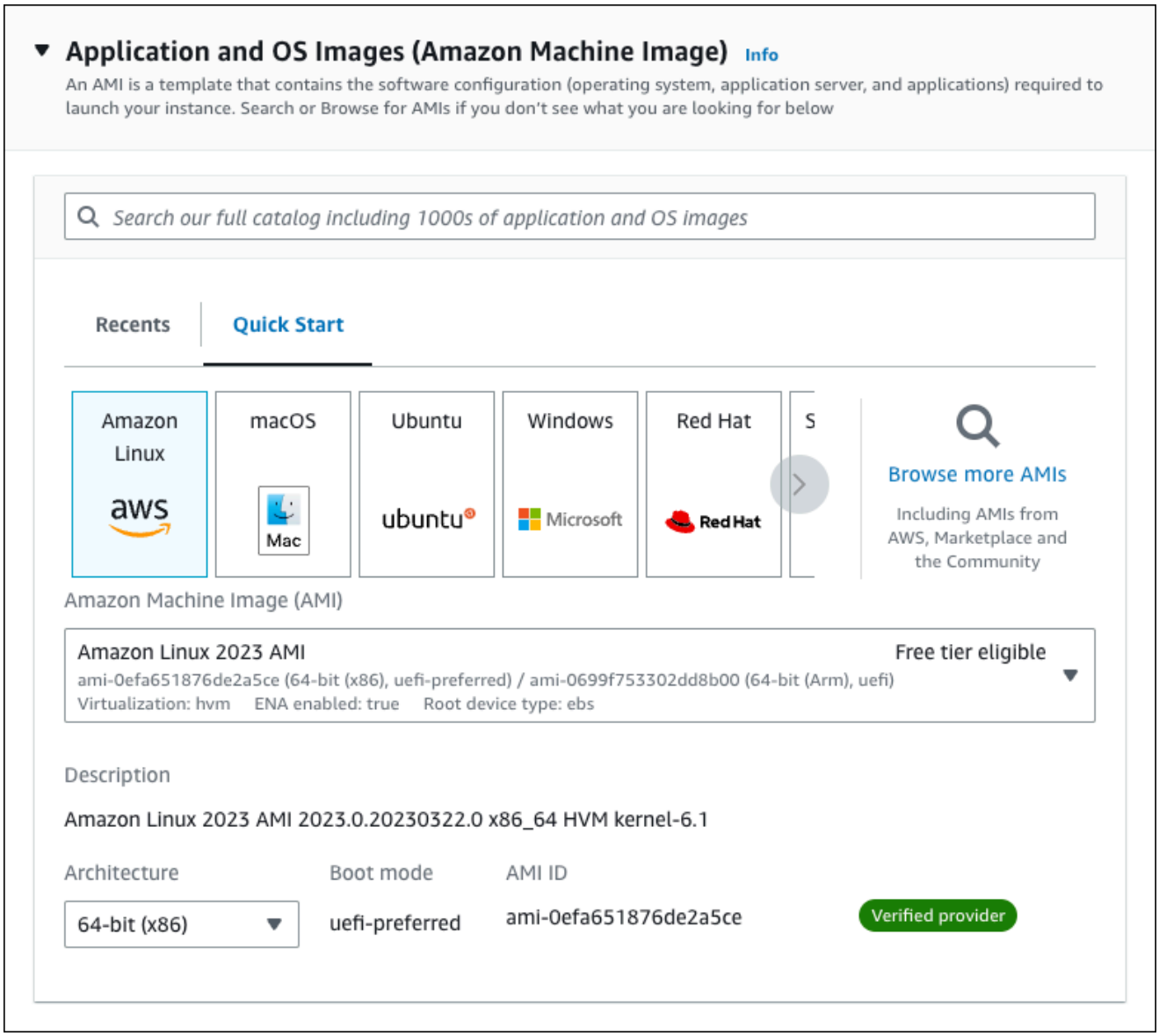


The **Launch an instance** page opens.

4. Choose the following settings on the **Launch an instance** page.

   a. Under **Name and tags**, for **Name**, enter **`ec2-database-connect`**.

   b. Under **Application and OS Images (Amazon Machine Image)**, choose **Amazon Linux**, and then choose the **Amazon Linux 2023 AMI**. Keep the default selections for the other choices.



   c. Under **Instance type**, choose **t2.micro**.

   d. Under **Key pair (login)**, choose a **Key pair name** to use an existing key pair. To create a new key pair for the Amazon EC2 instance, choose **Create new key pair** and then use the **Create key pair** window to create it.

For more information about creating a new key pair, see Create a key pair in the *Amazon EC2 User Guide*.

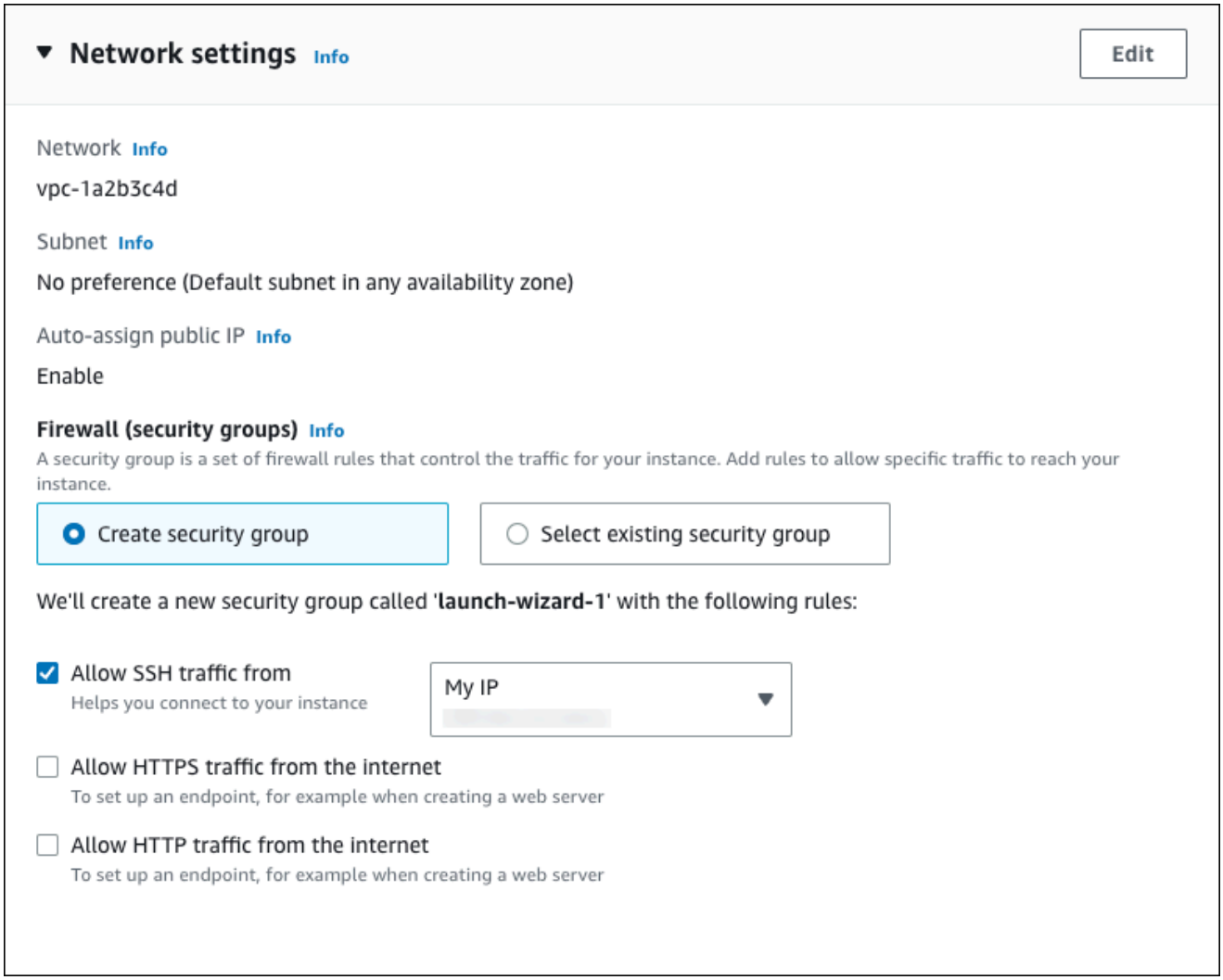
e. For **Allow SSH traffic** in **Network settings**, choose the source of SSH connections to the EC2 instance.

You can choose **My IP** if the displayed IP address is correct for SSH connections. Otherwise, you can determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH). To determine your public IP address, in a different browser window or tab, you can use the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, make sure to determine the range of IP addresses used by client computers.

> **Warning**
>
> If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

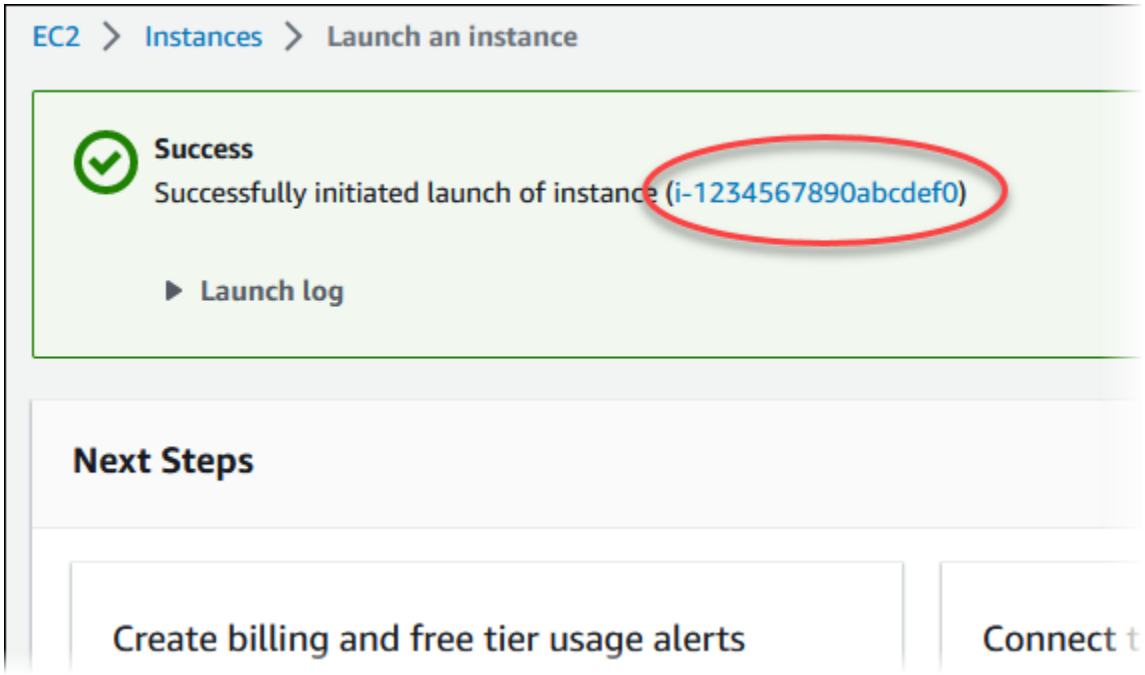
The following image shows an example of the **Network settings** section.



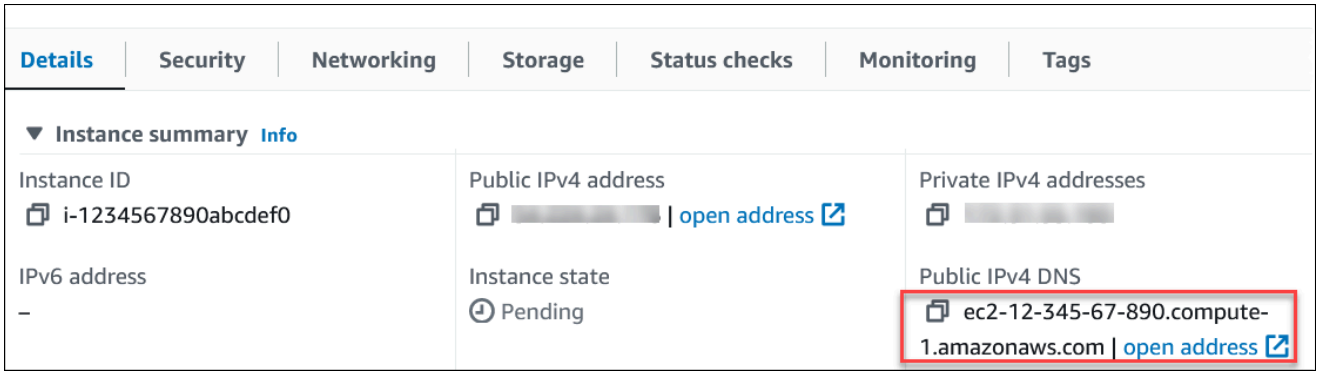
f. Leave the default values for the remaining sections.

g. Review a summary of your EC2 instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

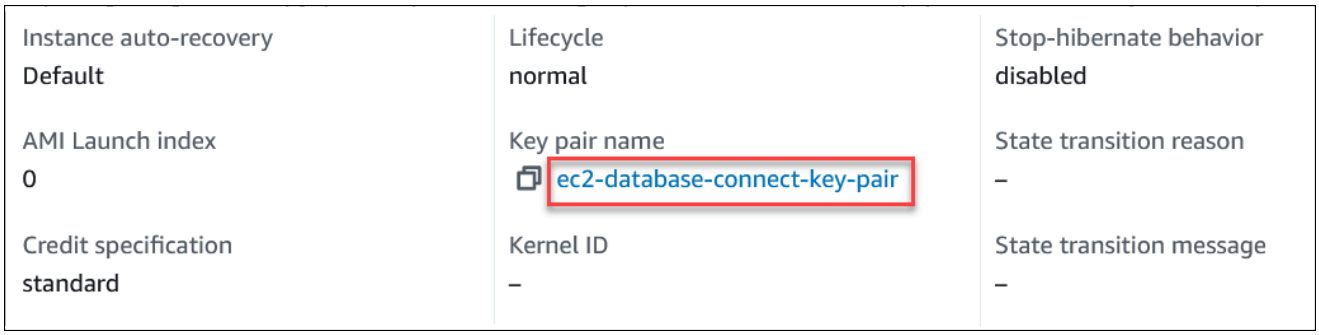
5. On the **Launch Status** page, note the identifier for your new EC2 instance, for example: `i-1234567890abcdef0`.



6. Choose the EC2 instance identifier to open the list of EC2 instances, and then select your EC2 instance.
7. In the **Details** tab, note the following values, which you need when you connect using SSH:
   a. In **Instance summary**, note the value for **Public IPv4 DNS**.



   b. In **Instance details**, note the value for **Key pair name**.



8. Wait until the **Instance state** for your EC2 instance has a status of **Running** before continuing.

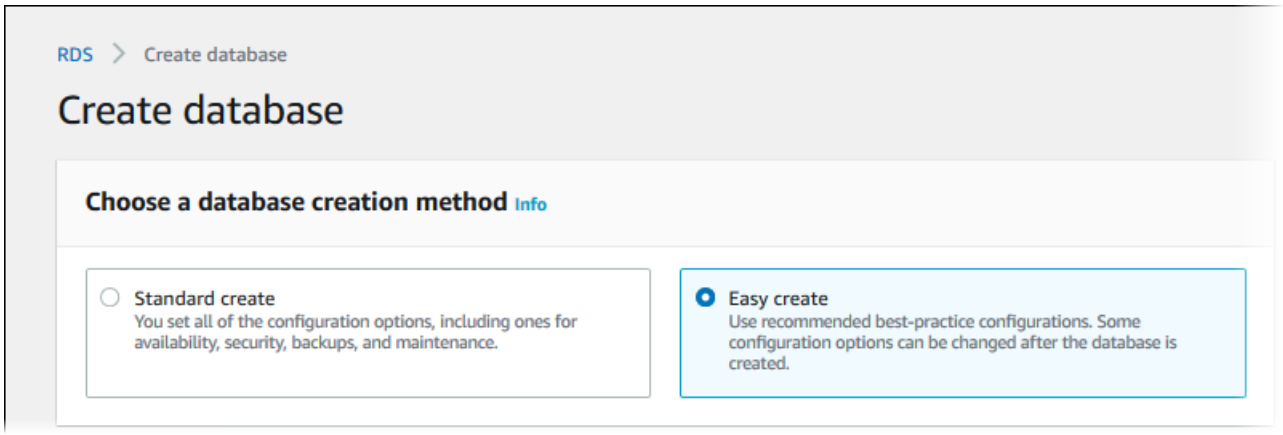
Create a PostgreSQL DB instance

The basic building block of Amazon RDS is the DB instance. This environment is where you run your PostgreSQL databases.

In this example, you use **Easy create** to create a DB instance running the PostgreSQL database engine with a db.t3.micro DB instance class.

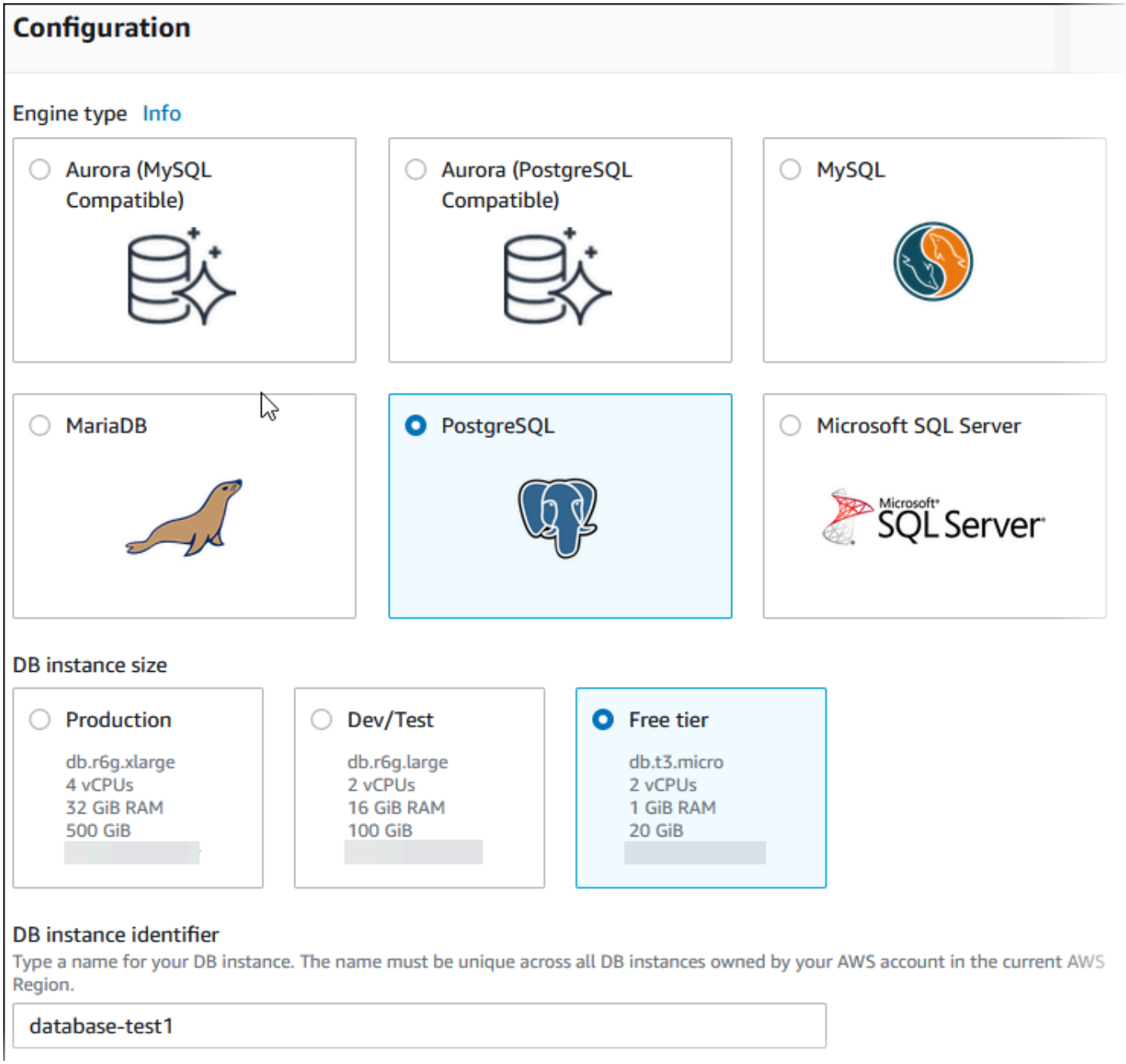
To create a PostgreSQL DB instance with Easy create

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database** and make sure that **Easy create** is chosen.



5. In **Configuration**, choose **PostgreSQL**.
6. For **DB instance size**, choose **Free tier**. **Free tier** appears for free plan accounts. **Sandbox** appears for paid plan accounts.
7. For **DB instance identifier**, enter **`database-test1`**.
8. For **Master username**, enter a name for the master user, or keep the default name (**`postgres`**).

   The **Create database** page should look similar to the following image. For free plan accounts, **Free tier** appears. For paid plan accounts, **Sandbox** appears.

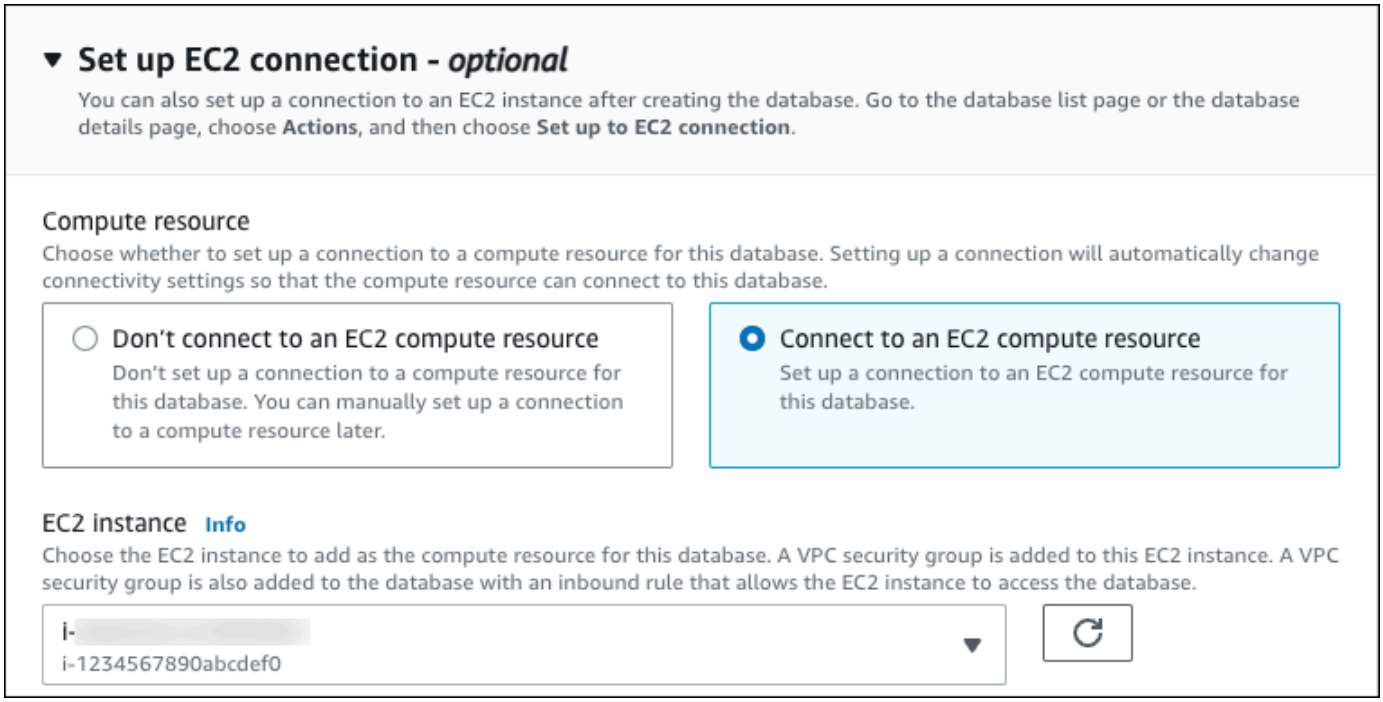


9. To use an automatically generated master password for the DB instance, select **Auto generate a password**.

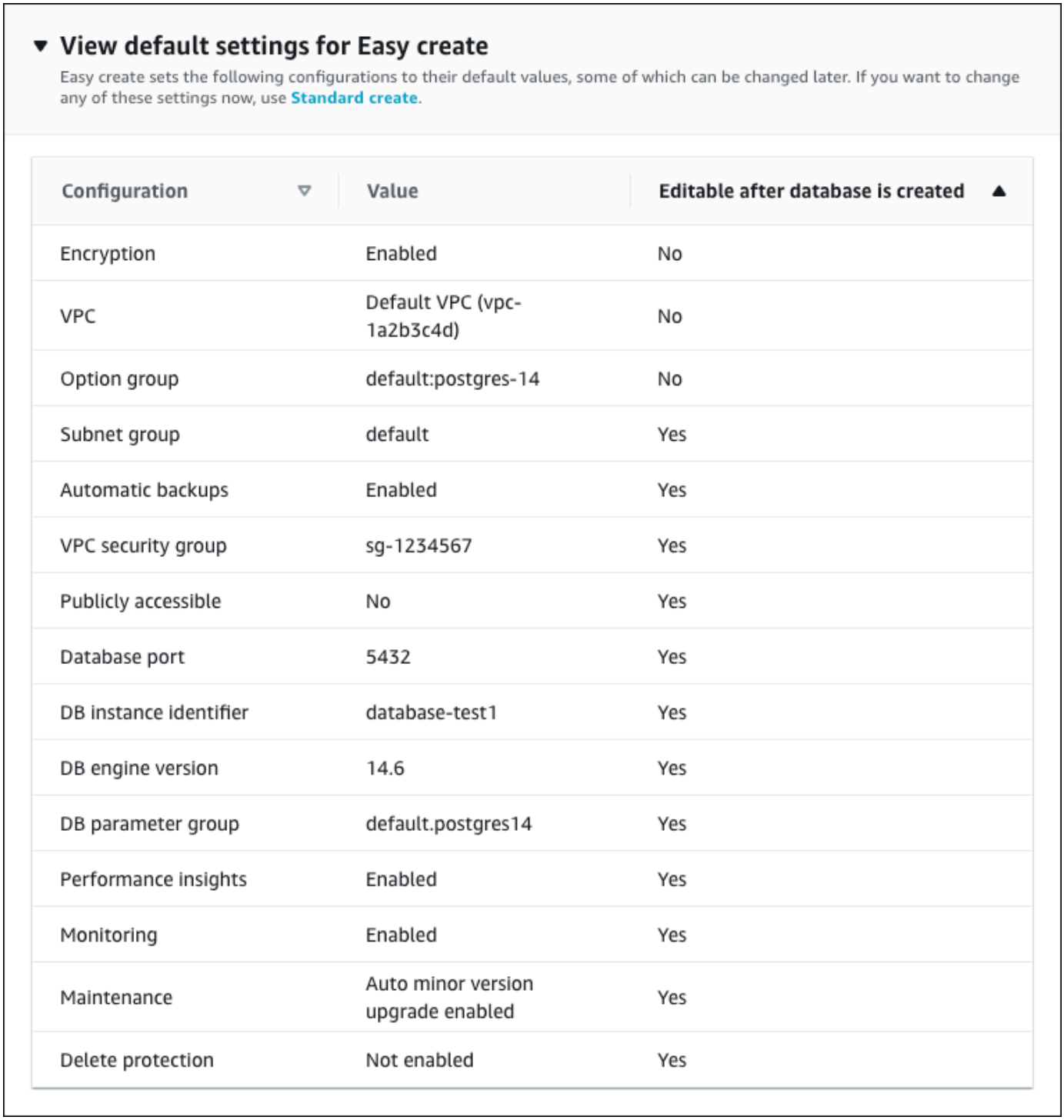
   To enter your master password, make sure **Auto generate a password** is cleared, and then enter the same password in **Master password** and **Confirm password**.

10. To set up a connection with the EC2 instance you created previously, open **Set up EC2 connection - *optional***.

    Select **Connect to an EC2 compute resource**. Choose the EC2 instance you created previously.



11. Open **View default settings for Easy create**.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:postgres-14	No
Subnet group	default	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	5432	Yes
DB instance identifier	database-test1	Yes
DB engine version	14.6	Yes
DB parameter group	default.postgres14	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

You can examine the default settings used with **Easy create**. The **Editable after database is created** column shows which options you can change after you create the database.

- If a setting has **No** in that column, and you want a different setting, you can use **Standard create** to create the DB instance.

- If a setting has **Yes** in that column, and you want a different setting, you can either use **Standard create** to create the DB instance, or modify the DB instance after you create it to change the setting.

12. Choose **Create database**.

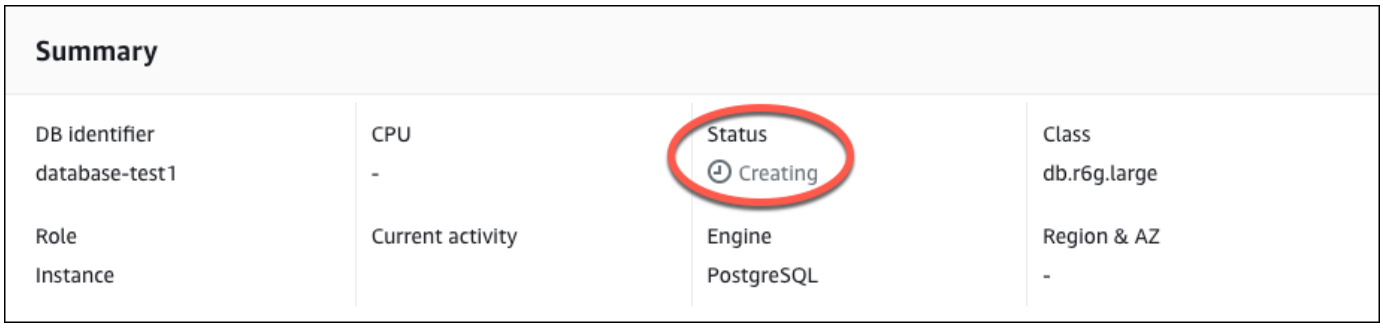
    To view the master username and password for the DB instance, choose **View credential details**.

    You can use the username and password that appears to connect to the DB instance as the master user.

    > **Important**
    >
    > You can't view the master user password again. If you don't record it, you might have to change it.
    > If you need to change the master user password after the DB instance is available, you can modify the DB instance to do so. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

13. In the **Databases** list, choose the name of the new PostgreSQL DB instance to show its details.

    The DB instance has a status of **Creating** until it is ready to use.



    When the status changes to **Available**, you can connect to the DB instance. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new instance is available.

(Optional) Create VPC, EC2 instance, and PostgreSQL instance using CloudFormation

Instead of using the console to create your VPC, EC2 instance, and PostgreSQL instance, you can use CloudFormation to provision AWS resources by treating infrastructure as code. To help you organize your AWS resources into smaller and more manageable units, you can use the CloudFormation nested stack functionality. For more information, see Creating a stack on the CloudFormation console and Working with nested stacks.

Important

CloudFormation is free, but the resources that CloudFormation creates are live. You incur the standard usage fees for these resources until you terminate them. For more information, see RDS for PostgreSQL pricing.

To create your resources using the CloudFormation console, complete the following steps:

- Download the CloudFormation template
- Configure your resources using CloudFormation

Download the CloudFormation template

A CloudFormation template is a JSON or YAML text file that contains the configuration information about the resources you want to create in the stack. This template also creates a VPC and a bastion host for you along with the RDS instance.

To download the template file, open the following link, PostgreSQL CloudFormation template.

In the Github page, click the *Download raw file* button to save the template YAML file.

Configure your resources using CloudFormation

Note

Before starting this process, make sure you have a Key pair for an EC2 instance in your AWS account. For more information, see Amazon EC2 key pairs and Linux instances.

When you use the CloudFormation template, you must select the correct parameters to make sure your resources are created properly. Follow the steps below:

1. Sign in to the AWS Management Console and open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Choose **Create Stack**.
3. In the Specify template section, select **Upload a template file from your computer**, and then choose **Next**.
4. In the **Specify stack details** page, set the following parameters:
   a. Set **Stack name** to **PostgreSQLTestStack**.
   b. Under **Parameters**, set **Availability Zones** by selecting three availability zones.
   c. Under **Linux Bastion Host configuration**, for **Key Name**, select a key pair to login to your EC2 instance.
   d. In **Linux Bastion Host configuration** settings, set the **Permitted IP range** to your IP address. To connect to EC2 instances in your VPC using Secure Shell (SSH), determine your public IP address using the service at https://checkip.amazonaws.com. An example of an IP address is 192.0.2.1/32.

      > **Warning**
      >
      > If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public EC2 instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your EC2 instances using SSH.

   e. Under **Database General configuration**, set **Database instance class** to **db.t3.micro**.
   f. Set **Database name** to **`database-test1`**.
   g. For **Database master username**, enter a name for the master user.
   h. Set **Manage DB master user password with Secrets Manager** to `false` for this tutorial.
   i. For **Database password**, set a password of your choice. Remember this password for further steps in the tutorial.
   j. Under **Database Storage configuration**, set **Database storage type** to **gp2**.
   k. Under **Database Monitoring configuration**, set **Enable RDS Performance Insights** to false.

l. Leave all other settings as the default values. Click **Next** to continue.

5. In the **Configure stack options** page, leave all the default options. Click **Next** to continue.
6. In the **Review stack** page, select **Submit** after checking the database and Linux bastion host options.

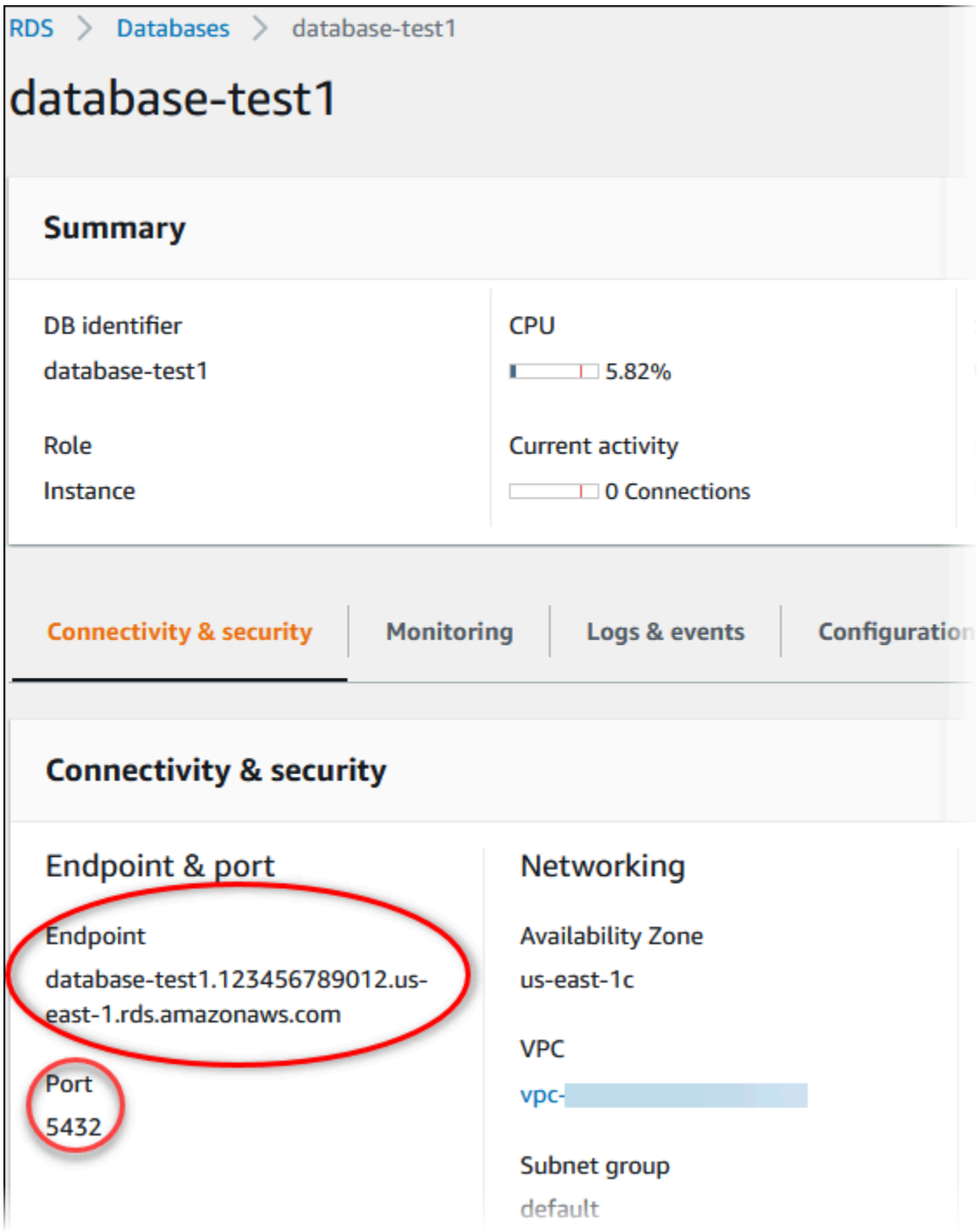
After the stack creation process completes, view the stacks with names *BastionStack* and *RDSNS* to note the information you need to connect to the database. For more information, see Viewing CloudFormation stack data and resources on the AWS Management Console.

Connect to a PostgreSQL DB instance

You can connect to the DB instance using pgadmin or psql. This example explains how to connect to a PostgreSQL DB instance using the psql command-line client.

To connect to a PostgreSQL DB instance using psql

1. Find the endpoint (DNS name) and port number for your DB instance.
   a. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
   b. In the upper-right corner of the Amazon RDS console, choose the AWS Region for the DB instance.
   c. In the navigation pane, choose **Databases**.
   d. Choose the PostgreSQL DB instance name to display its details.
   e. On the **Connectivity & security** tab, copy the endpoint. Also note the port number. You need both the endpoint and the port number to connect to the DB instance.



2. Connect to the EC2 instance that you created earlier by following the steps in Connect to your Linux instance in the *Amazon EC2 User Guide*.

   We recommend that you connect to your EC2 instance using SSH. If the SSH client utility is installed on Windows, Linux, or Mac, you can connect to the instance using the following command format:

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

For example, assume that `ec2-database-connect-key-pair.pem` is stored in `/dir1` on Linux, and the public IPv4 DNS for your EC2 instance is `ec2-12-345-678-90.compute-1.amazonaws.com`. Your SSH command would look as follows:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. Get the latest bug fixes and security updates by updating the software on your EC2 instance. To do this, use the following command.

> **Note**
>
> The `-y` option installs the updates without asking for confirmation. To examine updates before installing, omit this option.

```
sudo dnf update -y
```

4. To install the psql command-line client from PostgreSQL on Amazon Linux 2023, run the following command:

```
sudo dnf install postgresql15
```

5. Connect to the PostgreSQL DB instance. For example, enter the following command at a command prompt on a client computer. This action lets you connect to the PostgreSQL DB instance using the psql client.

   Substitute the DB instance endpoint (DNS name) for *`endpoint`*, substitute the database name `--dbname` that you want to connect to for *`postgres`*, and substitute the master username that you used for *`postgres`*. Provide the master password that you used when prompted for a password.

```
psql --host=endpoint --port=5432 --dbname=postgres --username=postgres
```

After you enter the password for the user, you should see output similar to the following:

```
psql (14.3, server 14.6)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

For more information on connecting to a PostgreSQL DB instance, see Connecting to a DB instance running the PostgreSQL database engine. If you can't connect to your DB instance, see Troubleshooting connections to your RDS for PostgreSQL instance.

For security, it is a best practice to use encrypted connections. Only use an unencrypted PostgreSQL connection when the client and server are in the same VPC and the network is trusted. For information about using encrypted connections, see Connecting to a PostgreSQL DB instance over SSL.

6. Run SQL commands.

   For example, the following SQL command shows the current date and time:

   ```
   SELECT CURRENT_TIMESTAMP;
   ```

Delete the EC2 instance and DB instance

After you connect to and explore the sample EC2 instance and DB instance that you created, delete them so you're no longer charged for them.

If you used CloudFormation to create resources, skip this step and go to the next step.

To delete the EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane, choose **Instances**.
3. Select the EC2 instance, and choose **Instance state, Terminate instance**.
4. Choose **Terminate** when prompted for confirmation.

For more information about deleting an EC2 instance, see Terminate your instance in the *Amazon EC2 User Guide*.

To delete a DB instance with no final DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to delete.
4. For **Actions**, choose **Delete**.
5. Clear **Create final snapshot?** and **Retain automated backups**.
6. Complete the acknowledgement and choose **Delete**.

(Optional) Delete the EC2 instance and DB instance created with CloudFormation

If you used CloudFormation to create resources, delete the CloudFormation stack after you connect to and explore the sample EC2 instance and DB instance, so you're no longer charged for them.

To delete the CloudFormation resources

1. Open the CloudFormation console.
2. On the **Stacks** page in the CloudFormationconsole, select the root stack (the stack without the name VPCStack, BastionStack or RDSNS).
3. Choose **Delete**.
4. Select **Delete stack** when prompted for confirmation.

For more information about deleting a stack in CloudFormation, see Deleting a stack on the CloudFormation console in the *AWS CloudFormation User Guide*.

(Optional) Connect your DB instance to a Lambda function

You can also connect your RDS for PostgreSQL DB instance to a Lambda serverless compute resource. Lambda functions allow you to run code without provisioning or managing infrastructure. A Lambda function also allows you to automatically respond to code execution requests at

any scale, from a dozen events a day to hundreds of per second. For more information, see Automatically connecting a Lambda function and a DB instance.

Tutorial: Create a web server and an Amazon RDS DB instance

This tutorial shows you how to install an Apache web server with PHP and create a MariaDB, MySQL, or PostgreSQL database. The web server runs on an Amazon EC2 instance using Amazon Linux 2023, and you can choose between a MySQL or PostgreSQL DB instance. Both the Amazon EC2 instance and the DB instance run in a virtual private cloud (VPC) based on the Amazon VPC service.

Important

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources you use. You can delete these resources after you complete the tutorial if they are no longer needed.

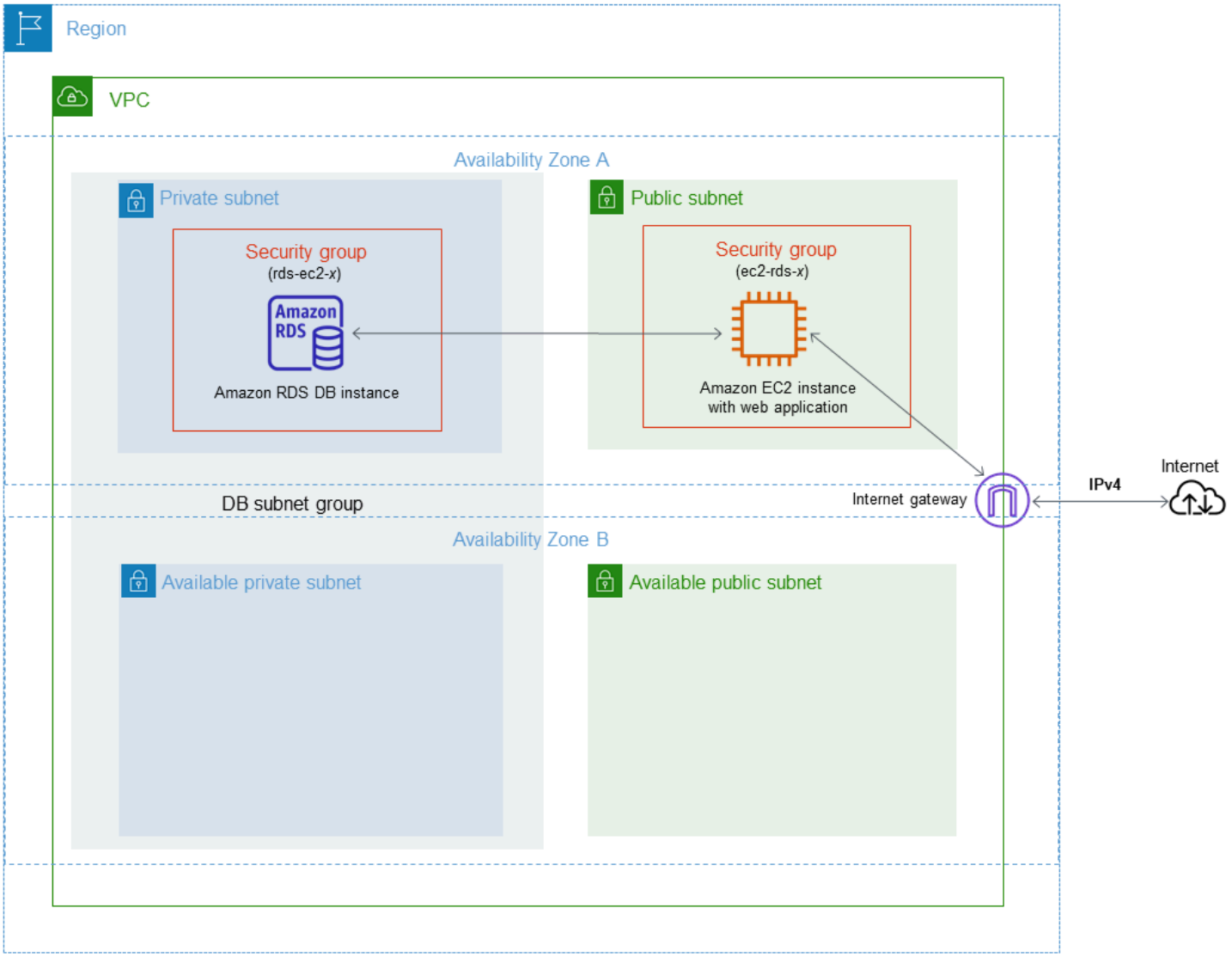
Note

This tutorial works with Amazon Linux 2023 and might not work for other versions of Linux.

In the tutorial that follows, you create an EC2 instance that uses the default VPC, subnets, and security group for your AWS account. This tutorial shows you how to create the DB instance and automatically set up connectivity with the EC2 instance that you created. The tutorial then shows you how to install the web server on the EC2 instance. You connect your web server to your DB instance in the VPC using the DB instance endpoint.

1. Launch an EC2 instance to connect with your DB instance
2. Create an Amazon RDS DB instance
3. Install a web server on your EC2 instance

The following diagram shows the configuration when the tutorial is complete.



Note

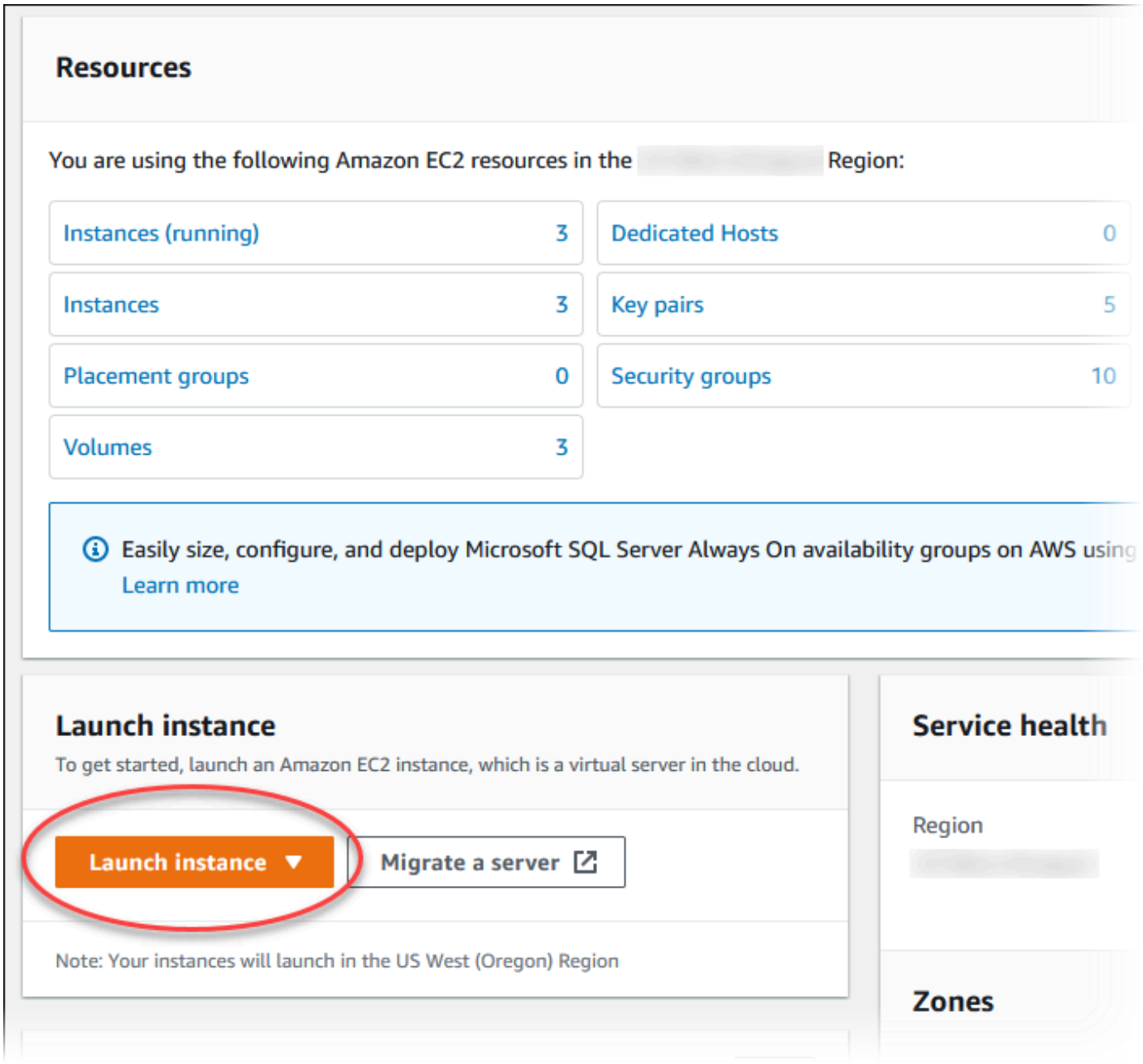
After you complete the tutorial, there is a public and private subnet in each Availability Zone in your VPC. This tutorial uses the default VPC for your AWS account and automatically sets up connectivity between your EC2 instance and DB instance. If you would rather configure a new VPC for this scenario instead, complete the tasks in Tutorial: Create a VPC for use with a DB instance (IPv4 only).

Launch an EC2 instance to connect with your DB instance

Create an Amazon EC2 instance in the public subnet of your VPC.

To launch an EC2 instance

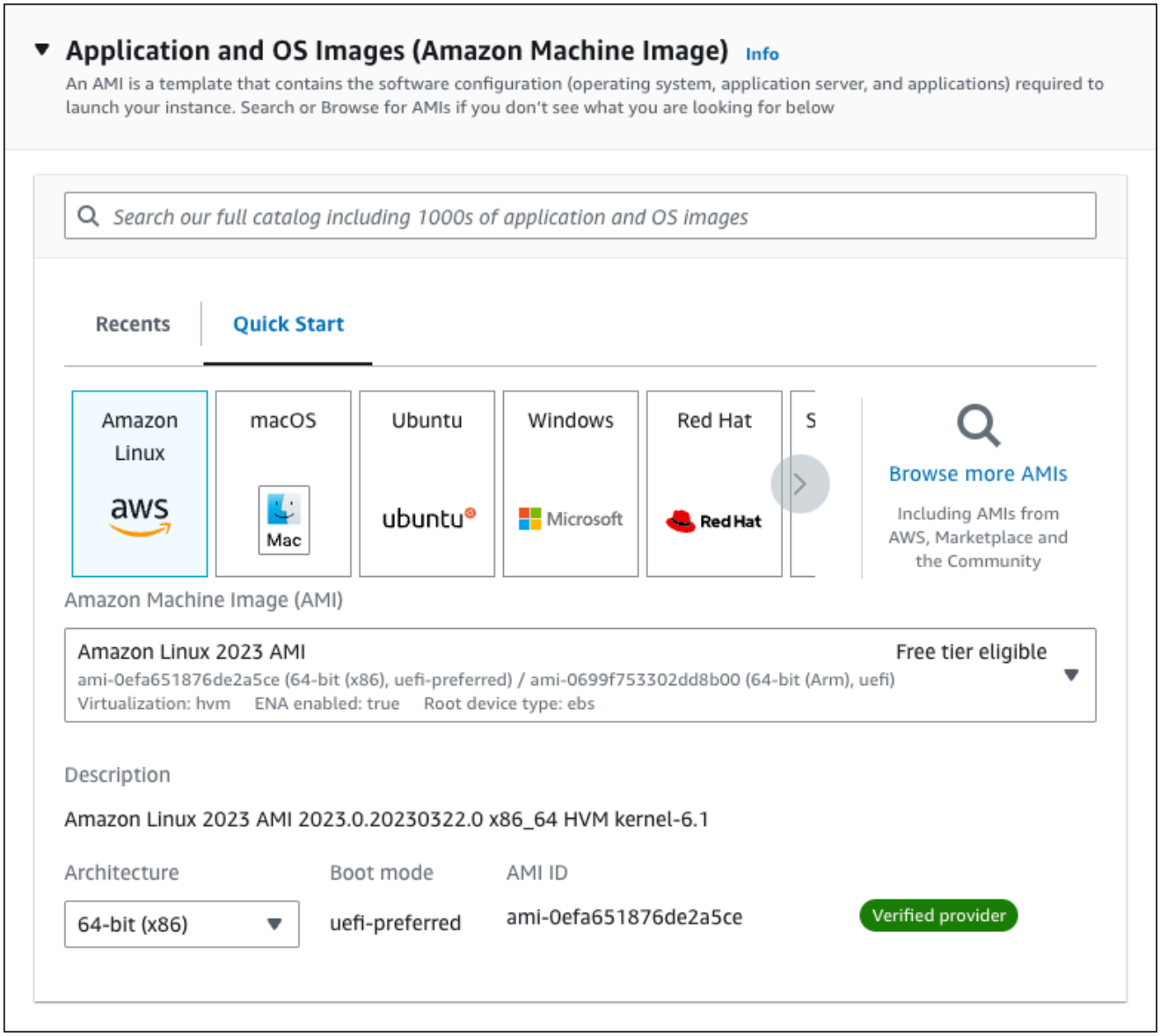
1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region where you want to create the EC2 instance.
3. Choose **EC2 Dashboard**, and then choose **Launch instance**, as shown following.



4. Choose the following settings in the **Launch an instance** page.

a. Under **Name and tags**, for **Name**, enter **`tutorial-ec2-instance-web-server`**.

b. Under **Application and OS Images (Amazon Machine Image)**, choose **Amazon Linux**, and then choose the **Amazon Linux 2023 AMI**. Keep the defaults for the other choices.



c. Under **Instance type**, choose **t2.micro**.

d. Under **Key pair (login)**, choose a **Key pair name** to use an existing key pair. To create a new key pair for the Amazon EC2 instance, choose **Create new key pair** and then use the **Create key pair** window to create it.

   For more information about creating a new key pair, see Create a key pair in the *Amazon EC2 User Guide*.

e. Under **Network settings**, set these values and keep the other values as their defaults:

- For **Allow SSH traffic from**, choose the source of SSH connections to the EC2 instance.

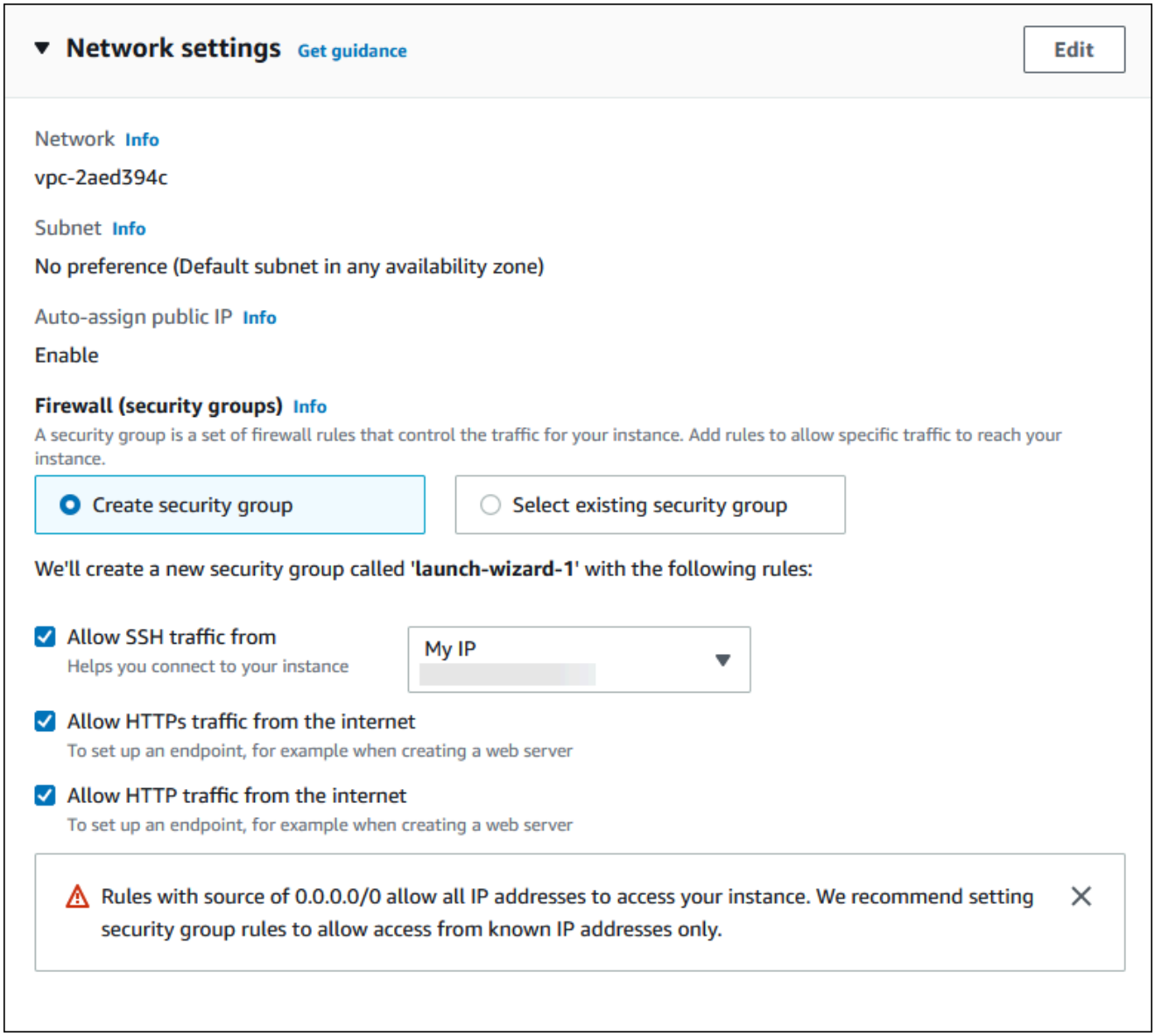
  You can choose **My IP** if the displayed IP address is correct for SSH connections.

  Otherwise, you can determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH). To determine your public IP address, in a different browser window or tab, you can use the service at https://checkip.amazonaws.com. An example of an IP address is `203.0.113.25/32`.

  In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, make sure to determine the range of IP addresses used by client computers.

  > **Warning**
  >
  > If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your instances using SSH.

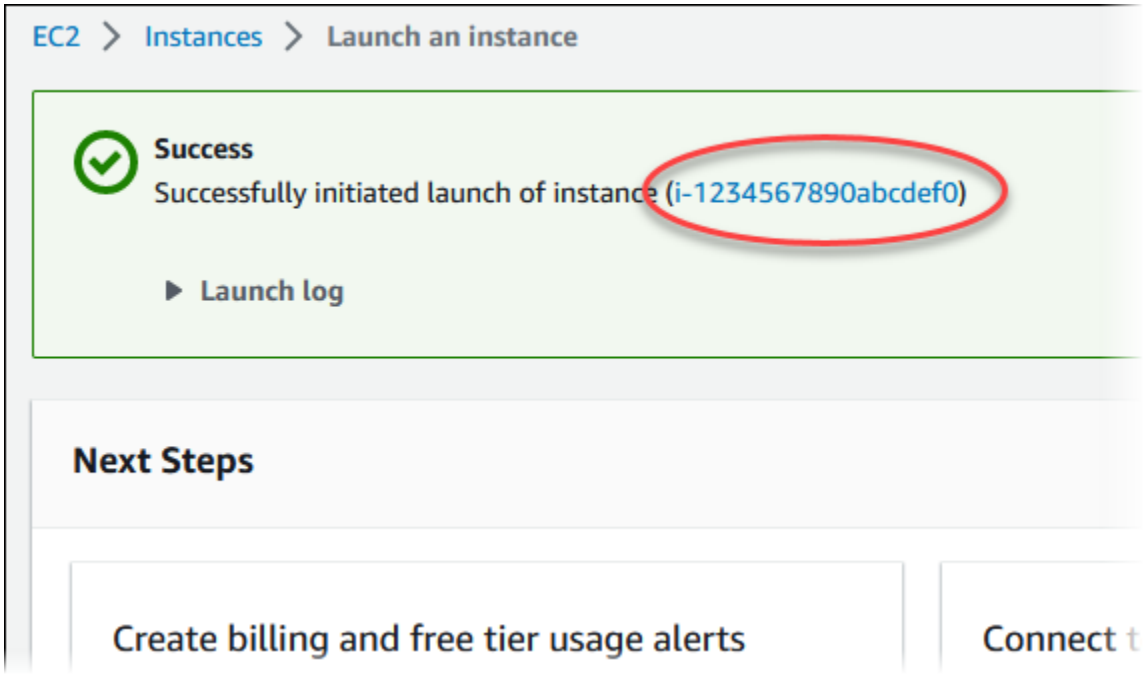
- Turn on **Allow HTTPs traffic from the internet**.
- Turn on **Allow HTTP traffic from the internet**.



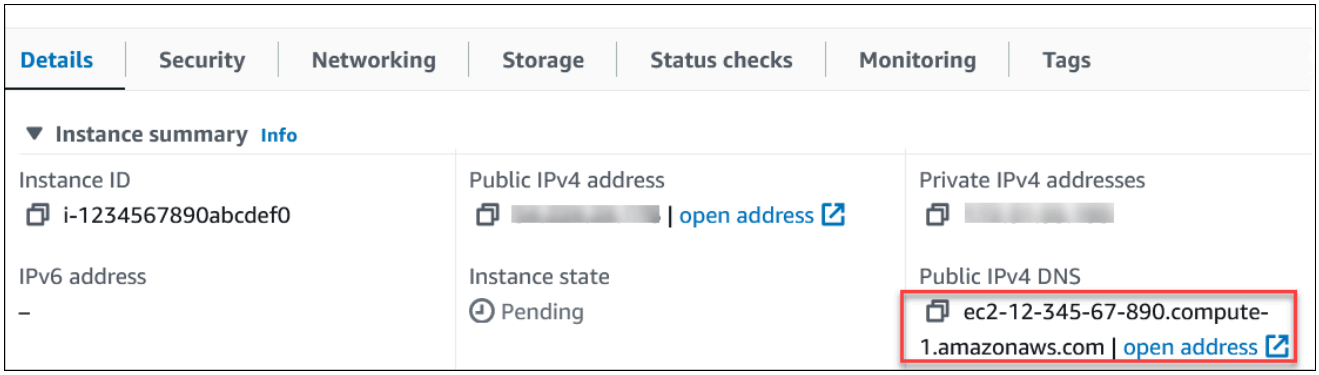
f. Leave the default values for the remaining sections.

g. Review a summary of your instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.

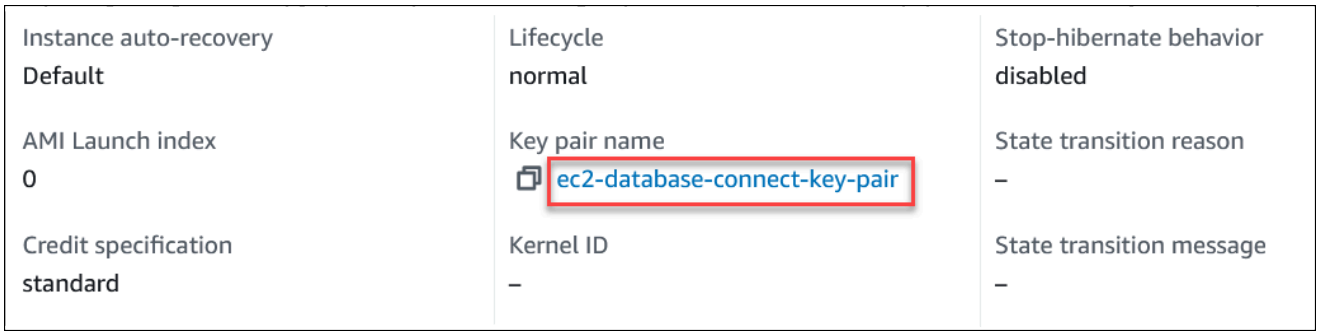
5. On the **Launch Status** page, note the identifier for your new EC2 instance, for example: `i-1234567890abcdef0`.



6. Choose the EC2 instance identifier to open the list of EC2 instances, and then select your EC2 instance.
7. In the **Details** tab, note the following values, which you need when you connect using SSH:
   a. In **Instance summary**, note the value for **Public IPv4 DNS**.



   b. In **Instance details**, note the value for **Key pair name**.



8. Wait until **Instance state** for your instance is **Running** before continuing.
9. Complete Create an Amazon RDS DB instance.

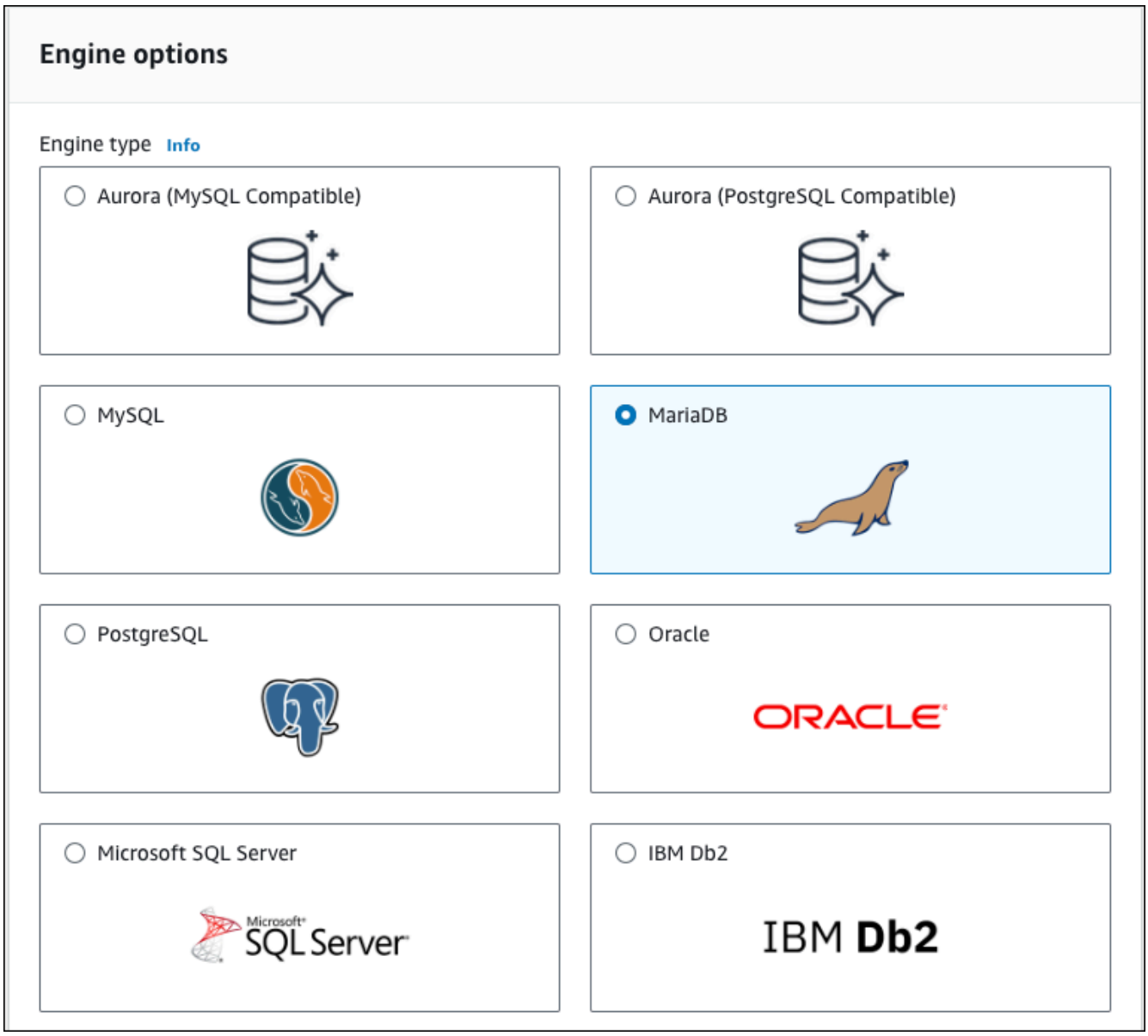
Create an Amazon RDS DB instance

Create an RDS for MariaDB, RDS for MySQL, or RDS for PostgreSQL DB instance that maintains the data used by a web application.

RDS for MariaDB

To create a MariaDB instance

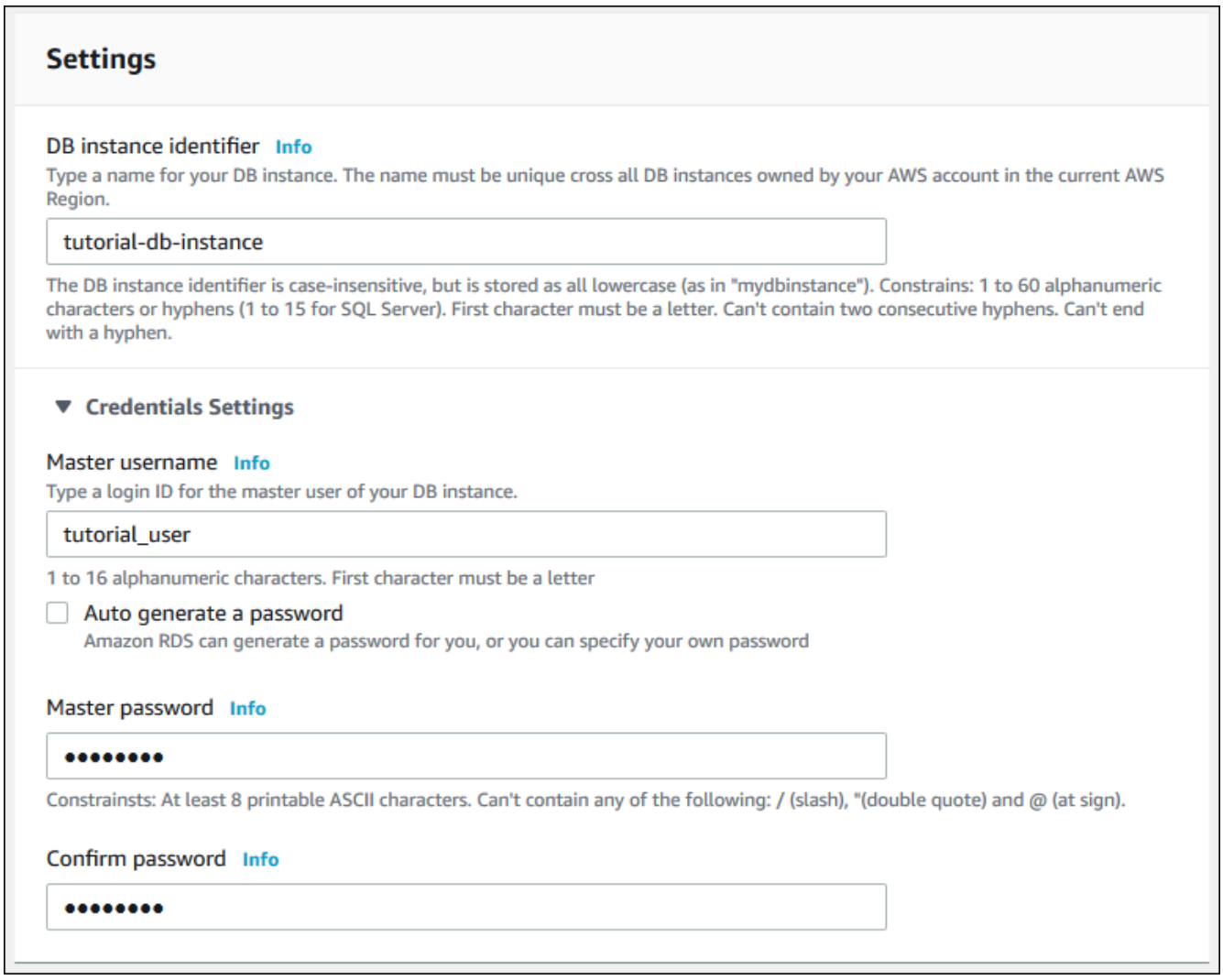
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the AWS Management Console, check the AWS Region. It should be the same as the one where you created your EC2 instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.
5. On the **Create database** page, choose **Standard create**.
6. For **Engine options**, choose **MariaDB**.



7. For **Templates**, choose **Free tier** or **Sandbox**. **Free tier** appears for free plan accounts. **Sandbox** appears for paid plan accounts.

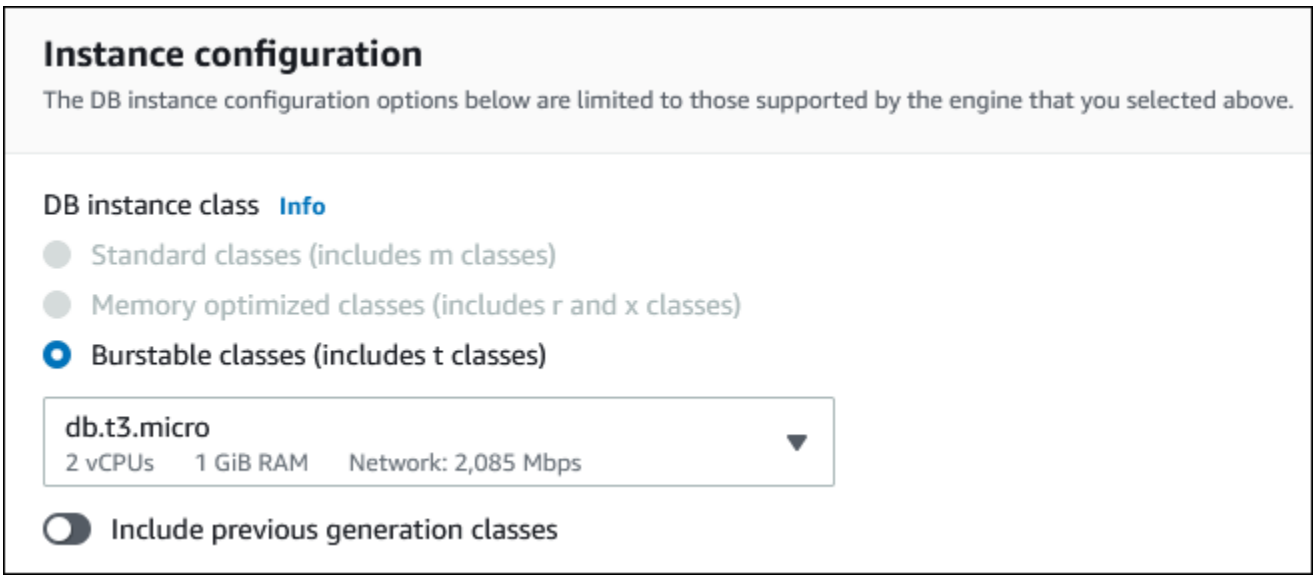


8. In the **Availability and durability** section, keep the defaults.
9. In the **Settings** section, set these values:
   - **DB instance identifier** – Type **`tutorial-db-instance`**.
   - **Master username** – Type **`tutorial_user`**.
   - **Auto generate a password** – Leave the option turned off.
   - **Master password** – Type a password.
   - **Confirm password** – Retype the password.

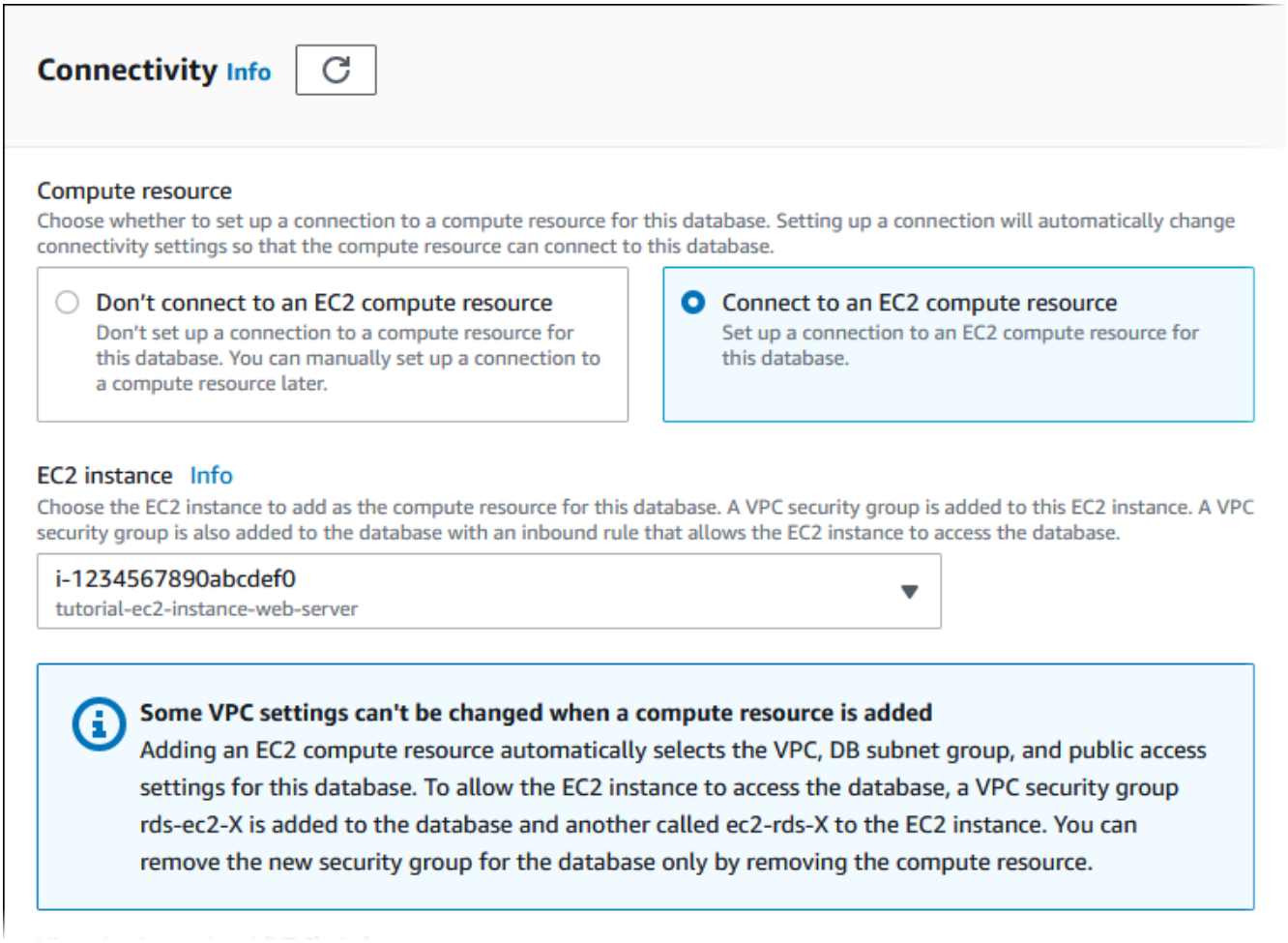


10. In the **Instance configuration** section, set these values:

    - **Burstable classes (includes t classes)**
    - **db.t3.micro**

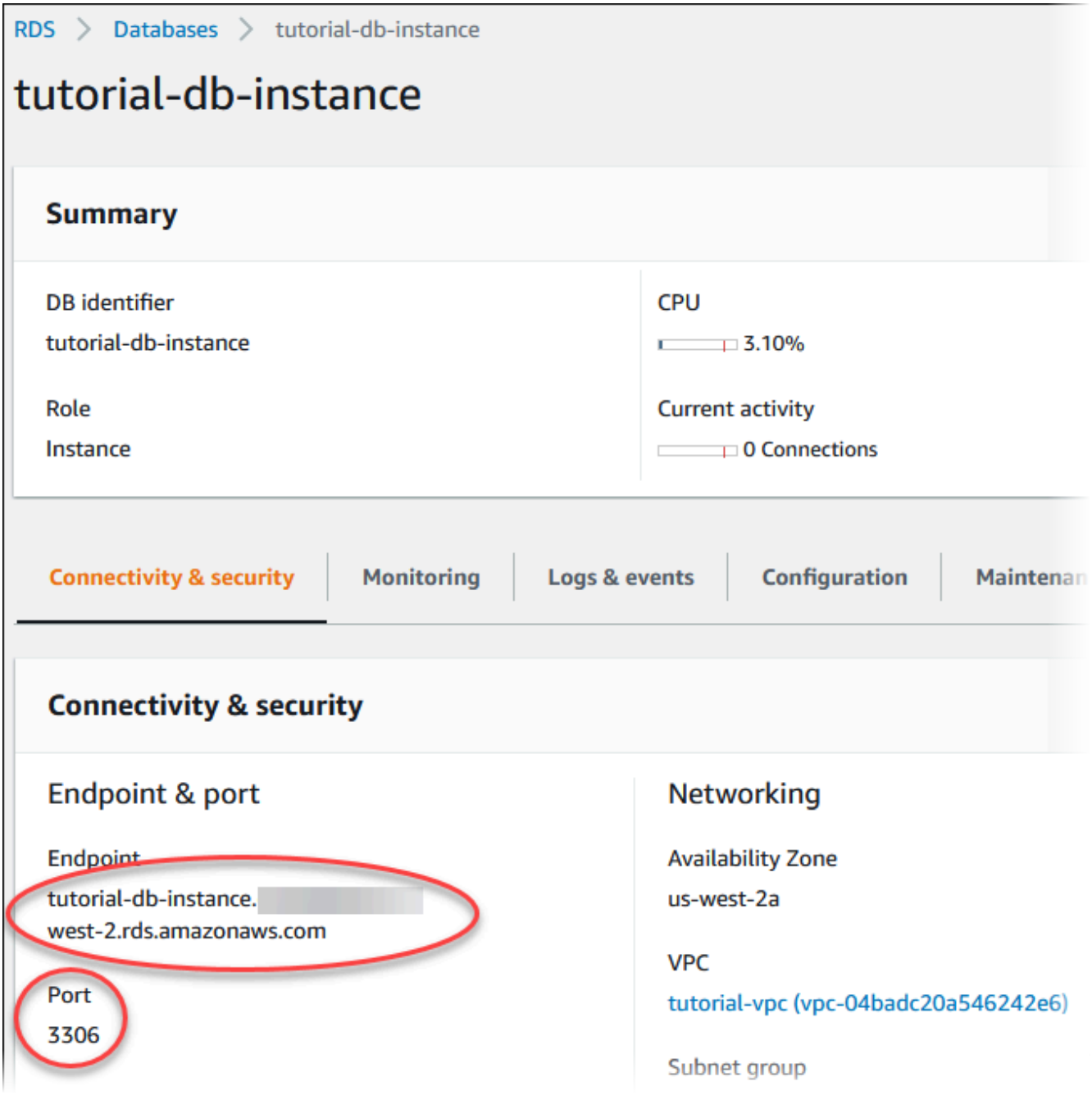


11. In the **Storage** section, keep the defaults.
12. In the **Connectivity** section, set these values and keep the other values as their defaults:
    - For **Compute resource**, choose **Connect to an EC2 compute resource**.
    - For **EC2 instance**, choose the EC2 instance you created previously, such as **tutorial-ec2-instance-web-server**.



13. In the **Database authentication** section, make sure **Password authentication** is selected.
14. Open the **Additional configuration** section, and enter **`sample`** for **Initial database name**. Keep the default settings for the other options.
15. To create your MariaDB instance, choose **Create database**.

    Your new DB instance appears in the **Databases** list with the status **Creating**.
16. Wait for the **Status** of your new DB instance to show as **Available**. Then choose the DB instance name to show its details.
17. In the **Connectivity & security** section, view the **Endpoint** and **Port** of the DB instance.



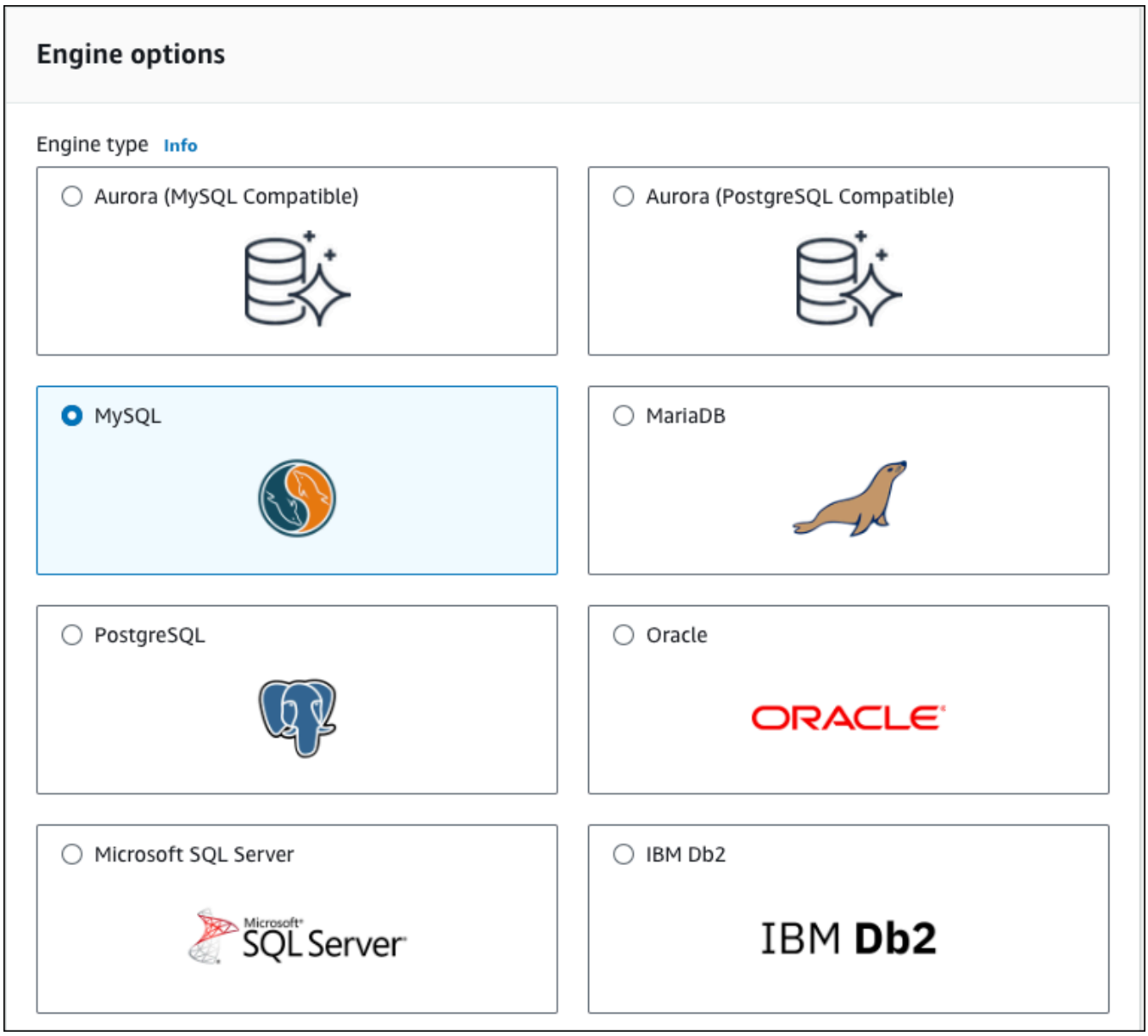
Note the endpoint and port for your DB instance. You use this information to connect your web server to your DB instance.

18. Complete Install a web server on your EC2 instance.

RDS for MySQL

To create a MySQL DB instance

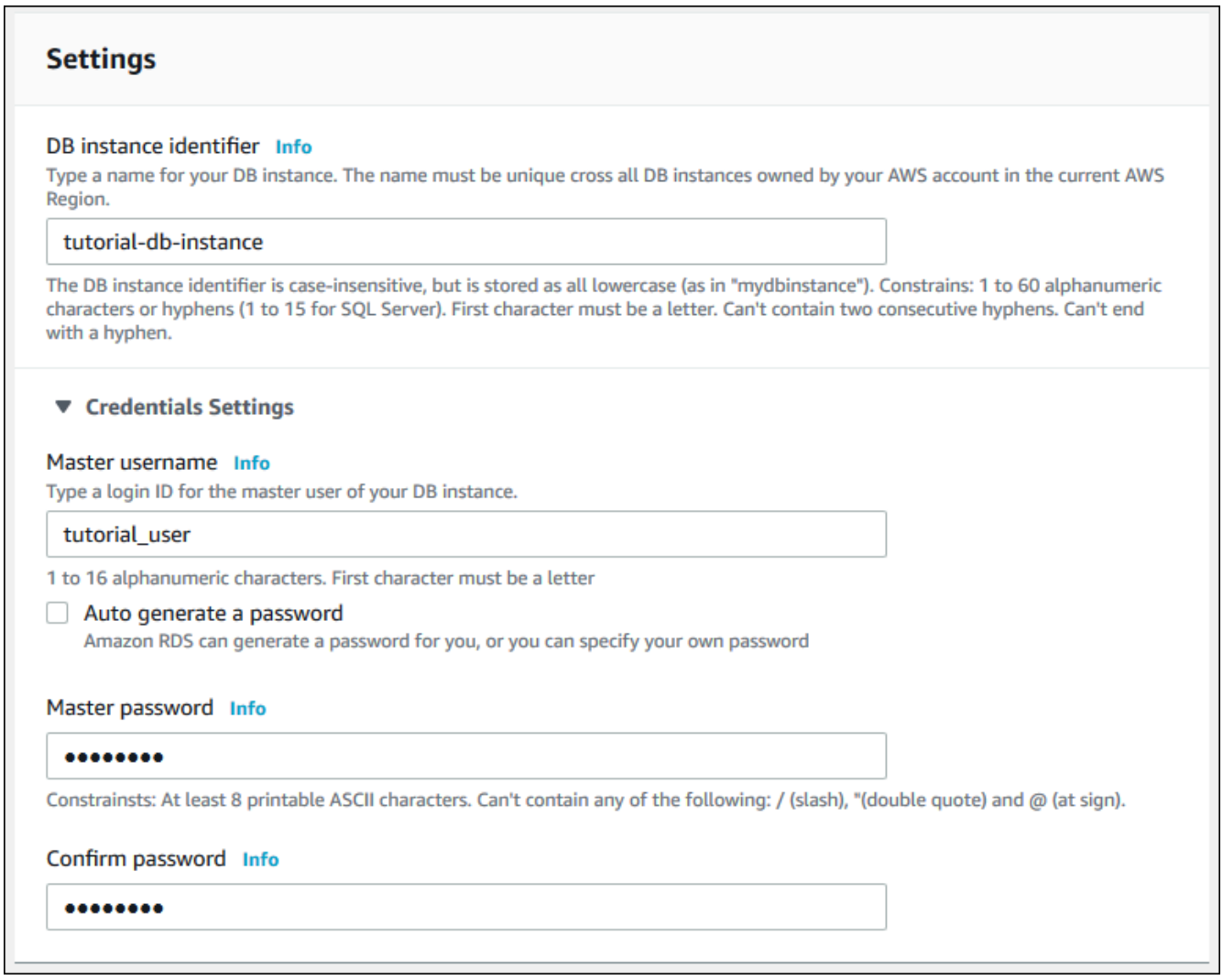
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the AWS Management Console, check the AWS Region. It should be the same as the one where you created your EC2 instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.
5. On the **Create database** page, choose **Standard create**.
6. For **Engine options**, choose **MySQL**.



7. For **Templates**, choose **Free tier** or **Sandbox**. **Free tier** appears for free plan accounts. **Sandbox** appears for paid plan accounts.

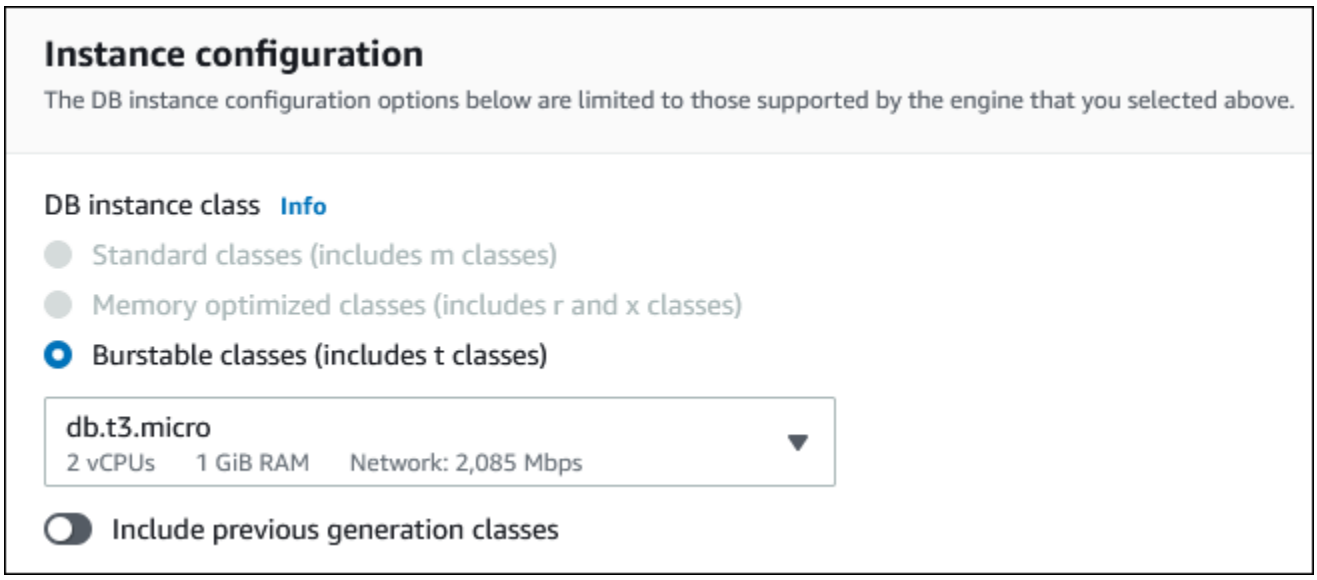


8. In the **Availability and durability** section, keep the defaults.
9. In the **Settings** section, set these values:
   - **DB instance identifier** – Type **`tutorial-db-instance`**.
   - **Master username** – Type **`tutorial_user`**.
   - **Auto generate a password** – Leave the option turned off.
   - **Master password** – Type a password.
   - **Confirm password** – Retype the password.

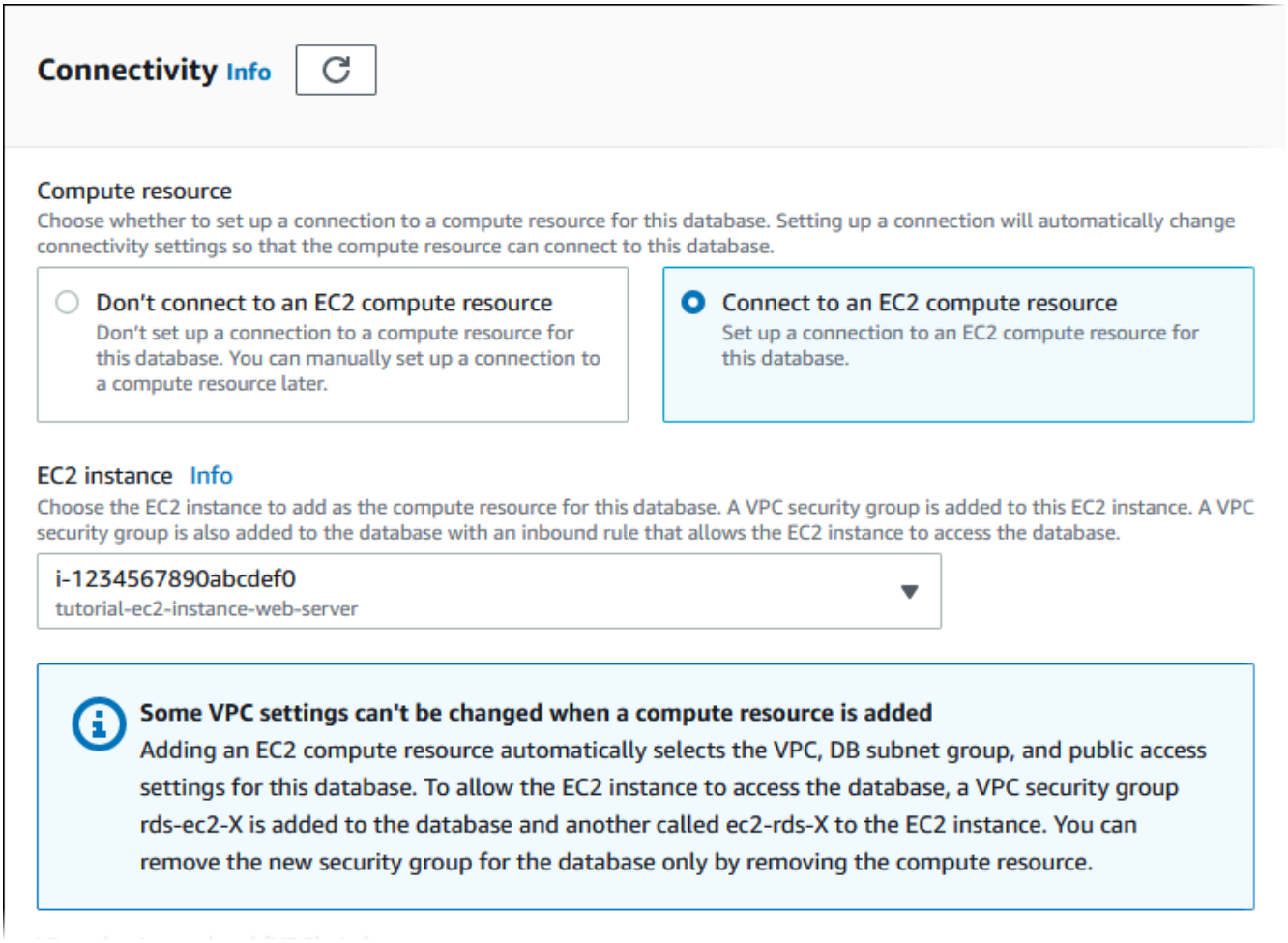


10. In the **Instance configuration** section, set these values:

    - **Burstable classes (includes t classes)**
    - **db.t3.micro**

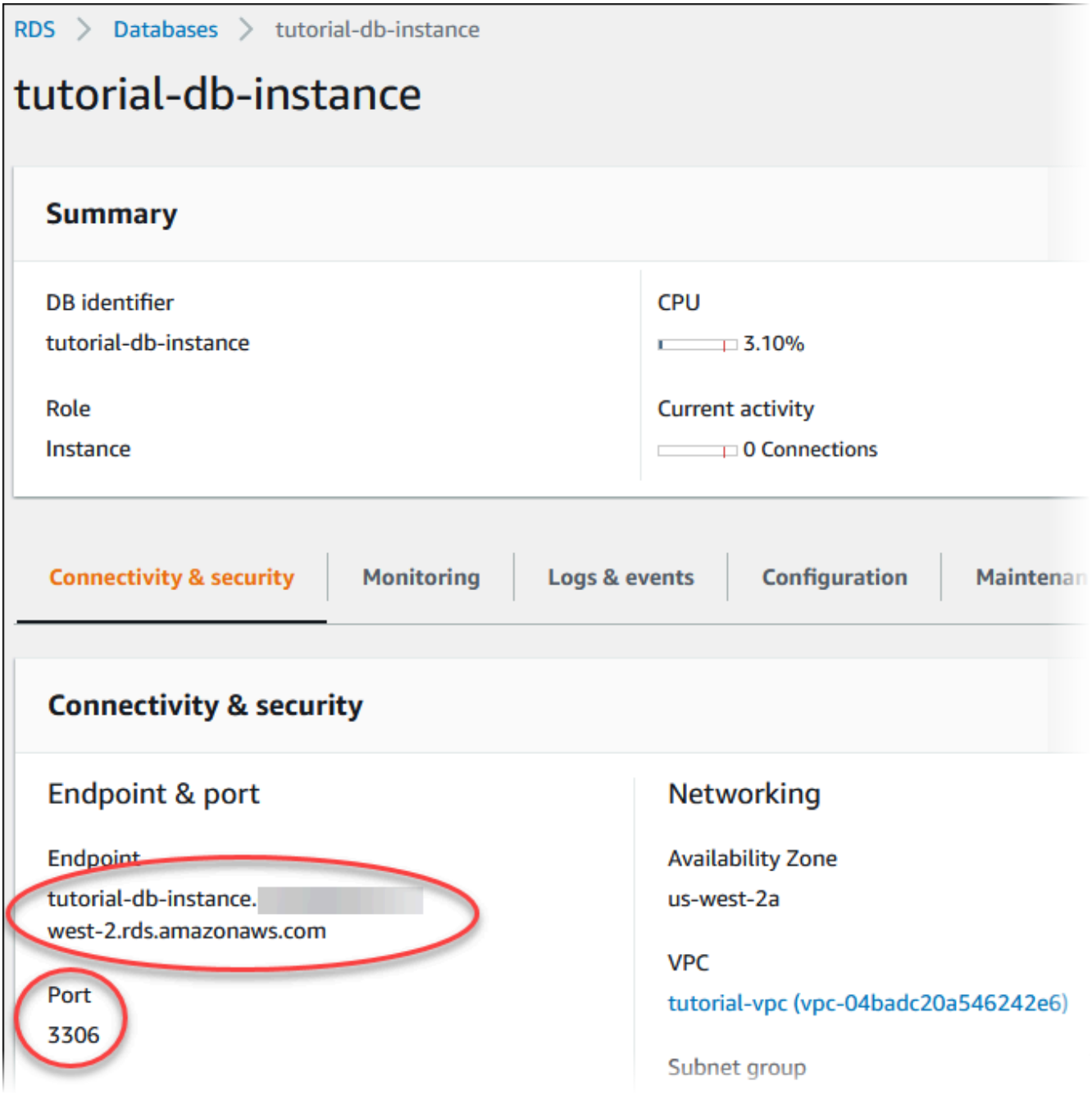


11. In the **Storage** section, keep the defaults.
12. In the **Connectivity** section, set these values and keep the other values as their defaults:
    - For **Compute resource**, choose **Connect to an EC2 compute resource**.
    - For **EC2 instance**, choose the EC2 instance you created previously, such as **tutorial-ec2-instance-web-server**.



13. In the **Database authentication** section, make sure **Password authentication** is selected.
14. Open the **Additional configuration** section, and enter **`sample`** for **Initial database name**. Keep the default settings for the other options.
15. To create your MySQL DB instance, choose **Create database**.

    Your new DB instance appears in the **Databases** list with the status **Creating**.
16. Wait for the **Status** of your new DB instance to show as **Available**. Then choose the DB instance name to show its details.
17. In the **Connectivity & security** section, view the **Endpoint** and **Port** of the DB instance.



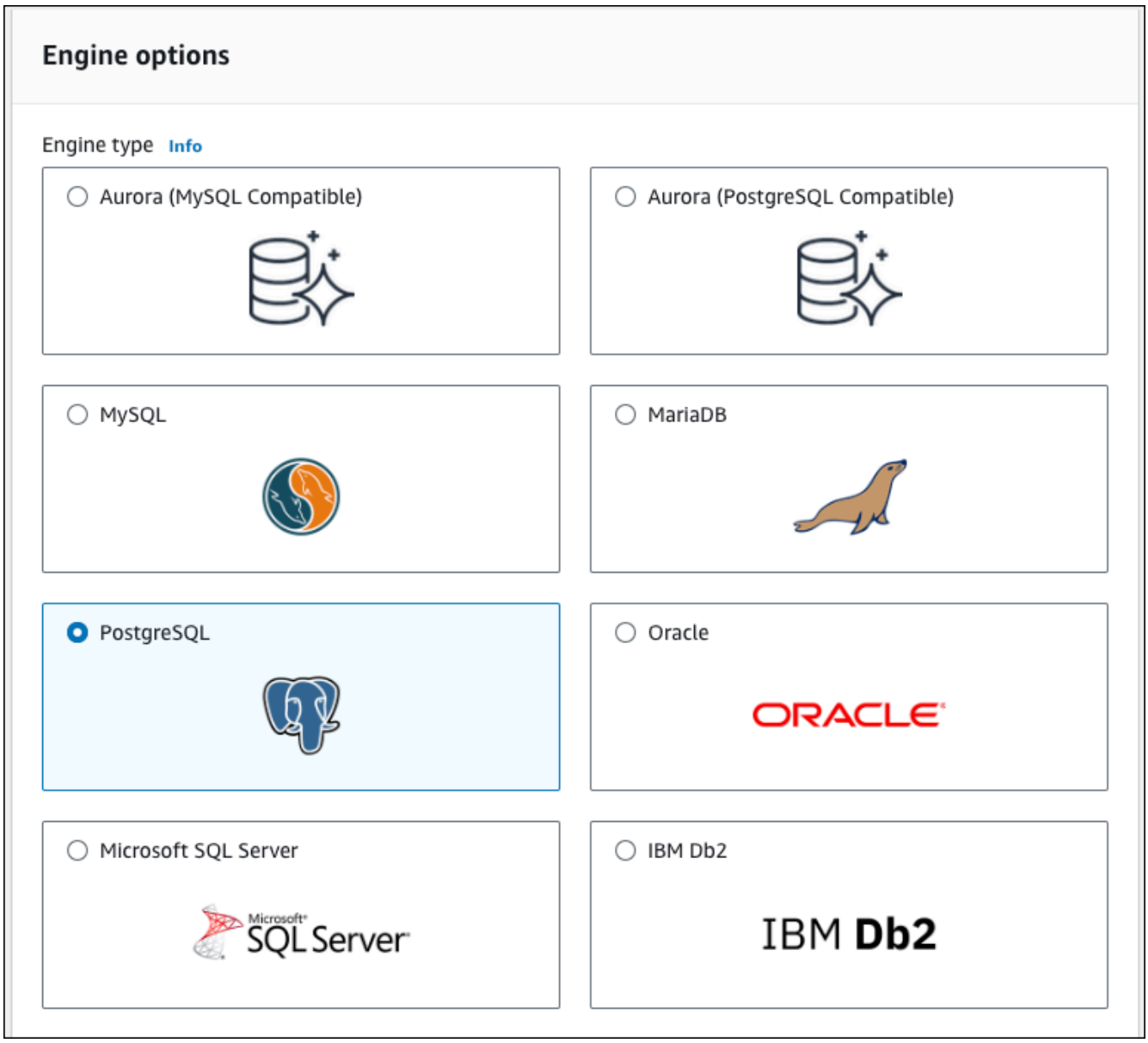
Note the endpoint and port for your DB instance. You use this information to connect your web server to your DB instance.

18. Complete Install a web server on your EC2 instance.

RDS for PostgreSQL

To create a PostgreSQL DB instance

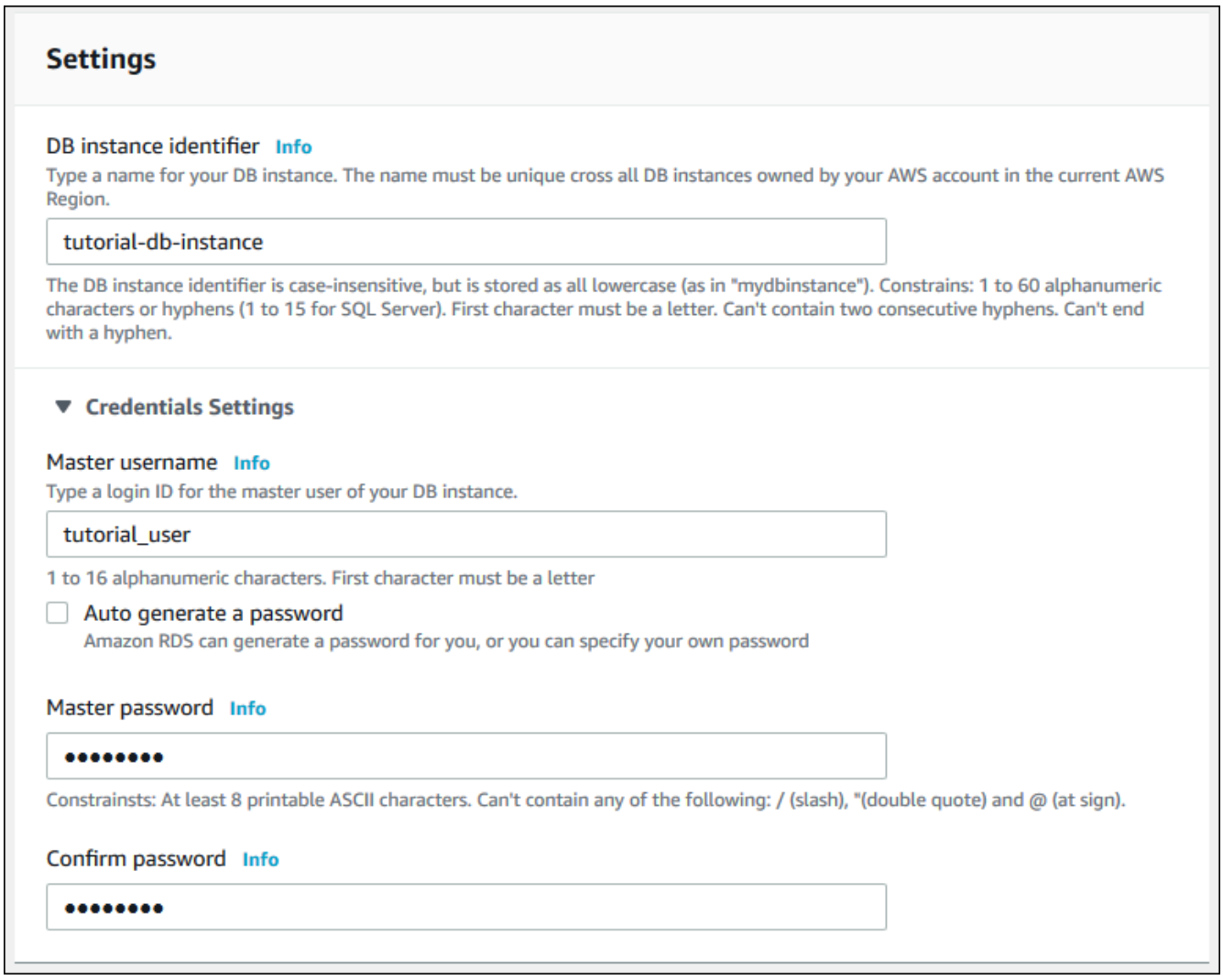
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the AWS Management Console, check the AWS Region. It should be the same as the one where you created your EC2 instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.
5. On the **Create database** page, choose **Standard create**.
6. For **Engine options**, choose **PostgreSQL**.



7. For **Templates**, choose **Free tier** or **Sandbox**. **Free tier** appears for free plan accounts. **Sandbox** appears for paid plan accounts.

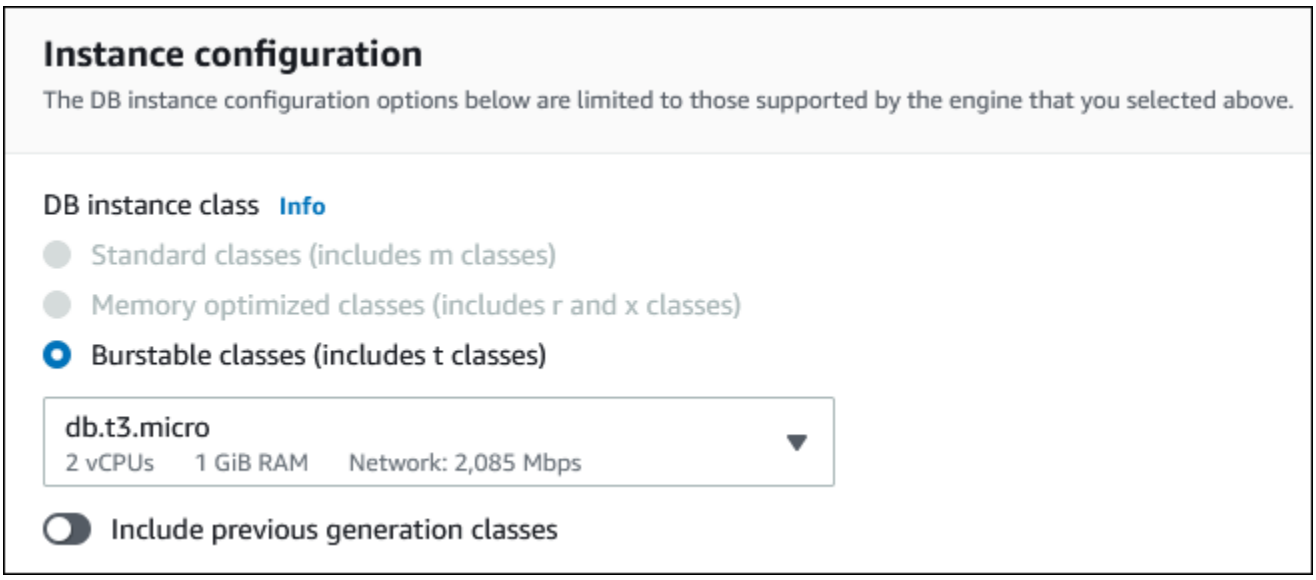


8. In the **Availability and durability** section, keep the defaults.
9. In the **Settings** section, set these values:
   - **DB instance identifier** – Type **`tutorial-db-instance`**.
   - **Master username** – Type **`tutorial_user`**.
   - **Auto generate a password** – Leave the option turned off.
   - **Master password** – Type a password.
   - **Confirm password** – Retype the password.

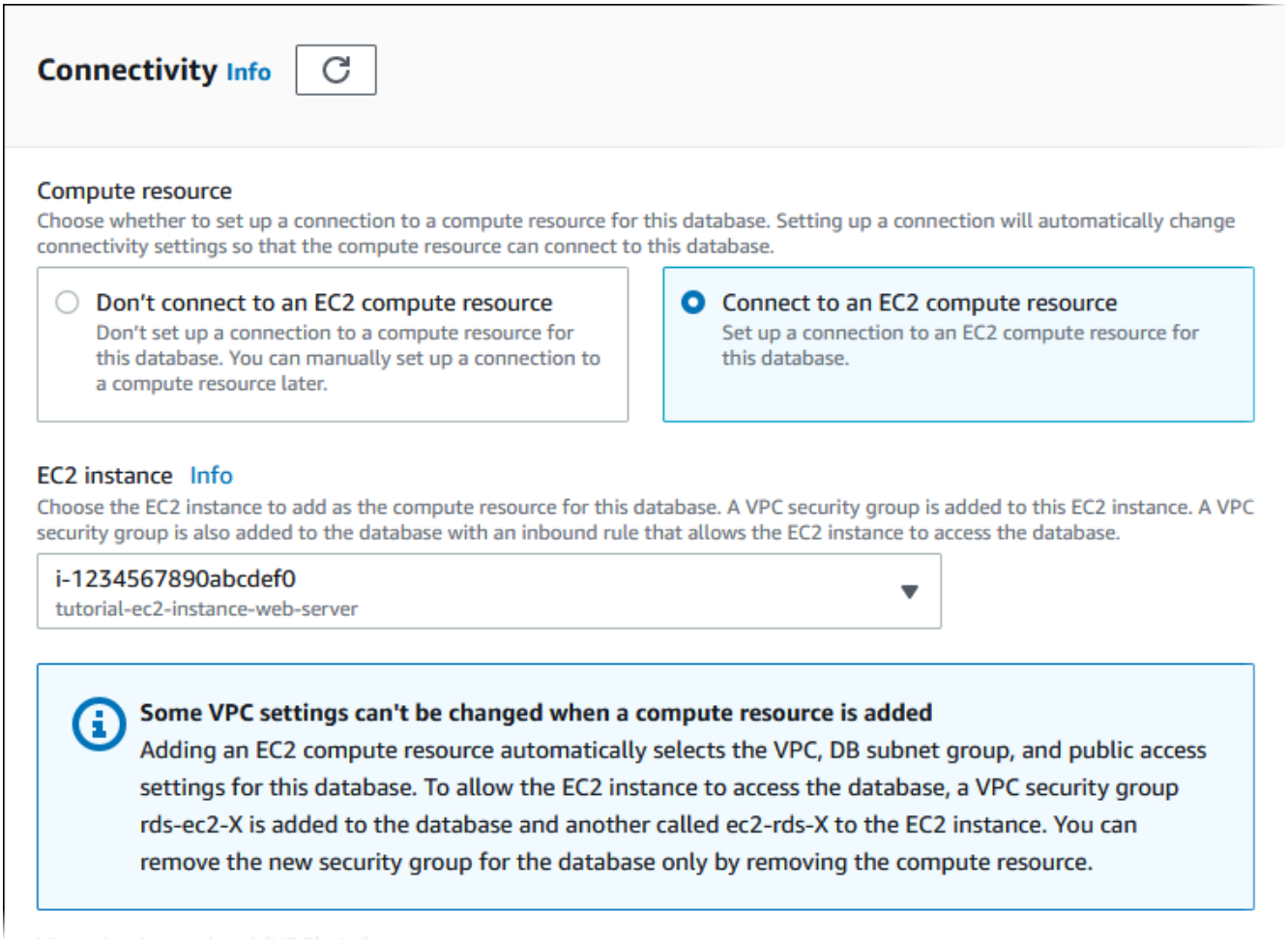


10. In the **Instance configuration** section, set these values:

    - **Burstable classes (includes t classes)**
    - **db.t3.micro**

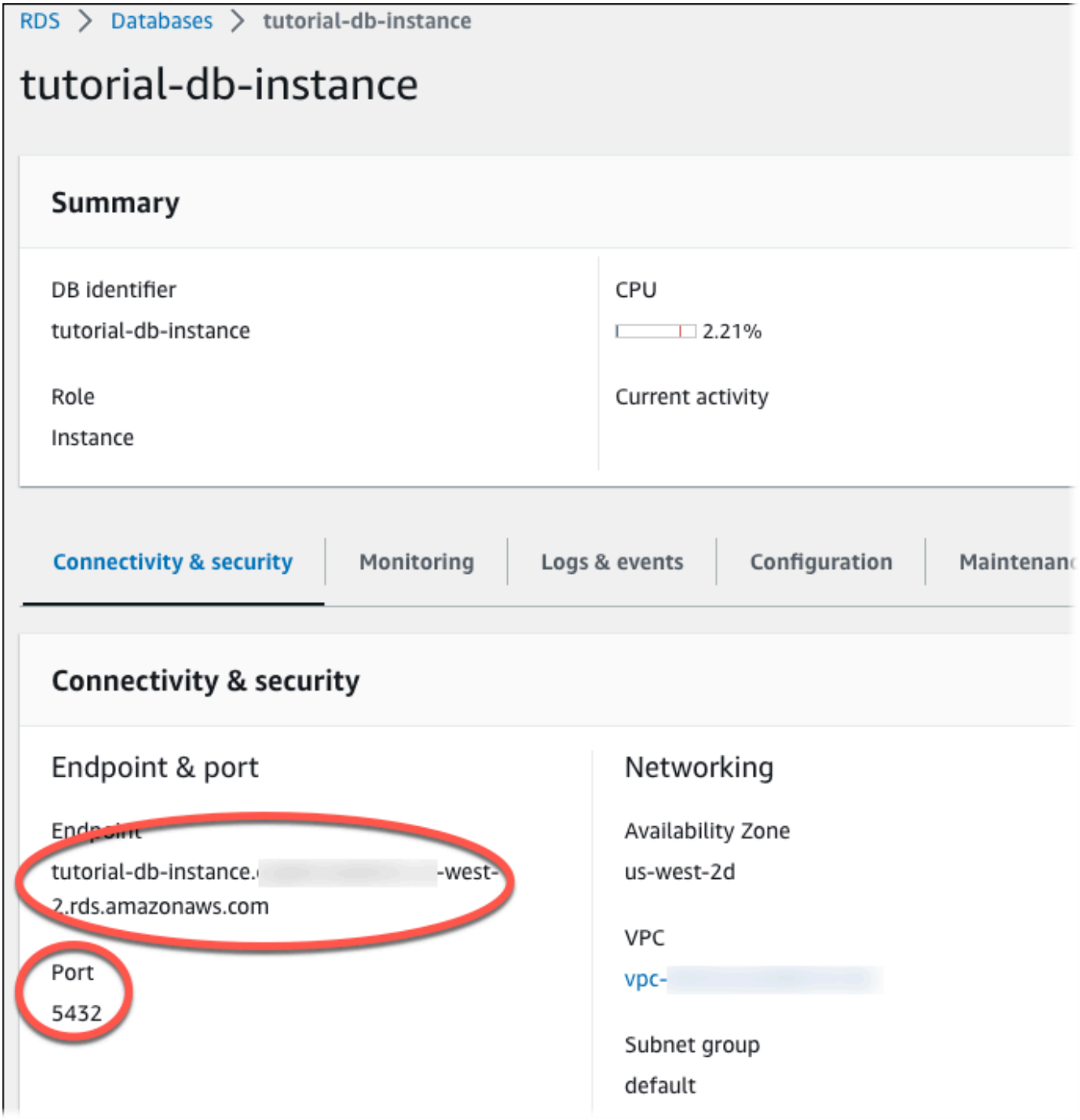


11. In the **Storage** section, keep the defaults.
12. In the **Connectivity** section, set these values and keep the other values as their defaults:
    - For **Compute resource**, choose **Connect to an EC2 compute resource**.
    - For **EC2 instance**, choose the EC2 instance you created previously, such as **tutorial-ec2-instance-web-server**.



13. In the **Database authentication** section, make sure **Password authentication** is selected.
14. Open the **Additional configuration** section, and enter **`sample`** for **Initial database name**. Keep the default settings for the other options.
15. To create your PostgreSQL DB instance, choose **Create database**.

    Your new DB instance appears in the **Databases** list with the status **Creating**.
16. Wait for the **Status** of your new DB instance to show as **Available**. Then choose the DB instance name to show its details.
17. In the **Connectivity & security** section, view the **Endpoint** and **Port** of the DB instance.



Note the endpoint and port for your DB instance. You use this information to connect your web server to your DB instance.

18. Complete Install a web server on your EC2 instance.

Install a web server on your EC2 instance

Install a web server on the EC2 instance you created in Launch an EC2 instance to connect with your DB instance. The web server connects to the Amazon RDS DB instance that you created in Create an Amazon RDS DB instance.

Install an Apache web server with PHP and MariaDB

Connect to your EC2 instance and install the web server.

To connect to your EC2 instance and install the Apache web server with PHP

1. Connect to the EC2 instance that you created earlier by following the steps in Connect to your Linux instance in the *Amazon EC2 User Guide*.

   We recommend that you connect to your EC2 instance using SSH. If the SSH client utility is installed on Windows, Linux, or Mac, you can connect to the instance using the following command format:

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   For example, assume that `ec2-database-connect-key-pair.pem` is stored in `/dir1` on Linux, and the public IPv4 DNS for your EC2 instance is `ec2-12-345-678-90.compute-1.amazonaws.com`. Your SSH command would look as follows:

   ```
   ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
   user@ec2-12-345-678-90.compute-1.amazonaws.com
   ```

2. Get the latest bug fixes and security updates by updating the software on your EC2 instance. To do this, use the following command.

   > **Note**
   >
   > The `-y` option installs the updates without asking for confirmation. To examine updates before installing, omit this option.

   ```
   sudo dnf update -y
   ```

3. After the updates complete, install the Apache web server, PHP, and MariaDB or PostgreSQL software using the following commands. This command installs multiple software packages and related dependencies at the same time.

   MariaDB & MySQL

   ```
   sudo dnf install -y httpd php php-mysqli mariadb105
   ```

   PostgreSQL

   ```
   sudo dnf install -y httpd php php-pgsql postgresql15
   ```

   If you receive an error, your instance probably wasn't launched with an Amazon Linux 2023 AMI. You might be using the Amazon Linux 2 AMI instead. You can view your version of Amazon Linux using the following command.

   ```
   cat /etc/system-release
   ```

   For more information, see Updating instance software.

4. Start the web server with the command shown following.

   ```
   sudo systemctl start httpd
   ```

   You can test that your web server is properly installed and started. To do this, enter the public Domain Name System (DNS) name of your EC2 instance in the address bar of a web browser, for example: `http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`. If your web server is running, then you see the Apache test page.

   If you don't see the Apache test page, check your inbound rules for the VPC security group that you created in Tutorial: Create a VPC for use with a DB instance (IPv4 only). Make sure that your inbound rules include one allowing HTTP (port 80) access for the IP address to connect to the web server.

   Note

   The Apache test page appears only when there is no content in the document root directory, `/var/www/html`. After you add content to the document root directory,

your content appears at the public DNS address of your EC2 instance. Before this point, it appears on the Apache test page.

5. Configure the web server to start with each system boot using the `systemctl` command.

```
sudo systemctl enable httpd
```

To allow `ec2-user` to manage files in the default root directory for your Apache web server, modify the ownership and permissions of the `/var/www` directory. There are many ways to accomplish this task. In this tutorial, you add `ec2-user` to the `apache` group, to give the `apache` group ownership of the `/var/www` directory and assign write permissions to the group.

To set file permissions for the Apache web server

1. Add the `ec2-user` user to the `apache` group.

```
sudo usermod -a -G apache ec2-user
```

2. Log out to refresh your permissions and include the new `apache` group.

```
exit
```

3. Log back in again and verify that the `apache` group exists with the `groups` command.

```
groups
```

Your output looks similar to the following:

```
ec2-user adm wheel apache systemd-journal
```

4. Change the group ownership of the `/var/www` directory and its contents to the `apache` group.

```
sudo chown -R ec2-user:apache /var/www
```

5. Change the directory permissions of `/var/www` and its subdirectories to add group write permissions and set the group ID on subdirectories created in the future.

```
sudo chmod 2775 /var/www
```

```
find /var/www -type d -exec sudo chmod 2775 {} \;
```

6. Recursively change the permissions for files in the `/var/www` directory and its subdirectories to add group write permissions.

```
find /var/www -type f -exec sudo chmod 0664 {} \;
```

Now, `ec2-user` (and any future members of the `apache` group) can add, delete, and edit files in the Apache document root. This makes it possible for you to add content, such as a static website or a PHP application.

> **Note**
>
> A web server running the HTTP protocol provides no transport security for the data that it sends or receives. When you connect to an HTTP server using a web browser, much information is visible to eavesdroppers anywhere along the network pathway. This information includes the URLs that you visit, the content of web pages that you receive, and the contents (including passwords) of any HTML forms.
> The best practice for securing your web server is to install support for HTTPS (HTTP Secure). This protocol protects your data with SSL/TLS encryption. For more information, see Tutorial: Configure SSL/TLS with the Amazon Linux AMI in the *Amazon EC2 User Guide*.

Connect your Apache web server to your DB instance

Next, you add content to your Apache web server that connects to your Amazon RDS DB instance.

To add content to the Apache web server that connects to your DB instance

1. While still connected to your EC2 instance, change the directory to `/var/www` and create a new subdirectory named `inc`.

```
cd /var/www
mkdir inc
cd inc
```

2. Create a new file in the `inc` directory named `dbinfo.inc`, and then edit the file by calling nano (or the editor of your choice).

```
>dbinfo.inc
nano dbinfo.inc
```

3. Add the following contents to the `dbinfo.inc` file. Here, *db_instance_endpoint* is your DB instance endpoint, without the port, for your DB instance.

> **Note**
>
> We recommend placing the user name and password information in a folder that isn't part of the document root for your web server. Doing this reduces the possibility of your security information being exposed.
> Make sure to change `master password` to a suitable password in your application.

```
<?php

define('DB_SERVER', 'db_instance_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');
?>
```

4. Save and close the `dbinfo.inc` file. If you are using nano, save and close the file by using Ctrl +S and Ctrl+X.
5. Change the directory to `/var/www/html`.

```
cd /var/www/html
```

6. Create a new file in the `html` directory named `SamplePage.php`, and then edit the file by calling nano (or the editor of your choice).

```
>SamplePage.php
nano SamplePage.php
```

7. Add the following contents to the `SamplePage.php` file:

   MariaDB & MySQL

```
<?php include "../inc/dbinfo.inc"; ?>
```

```
<html>
<body>
<h1>Sample page</h1>
<?php

  /* Connect to MySQL and select the database. */
  $connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

  if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
 mysqli_connect_error();

  $database = mysqli_select_db($connection, DB_DATABASE);

  /* Ensure that the EMPLOYEES table exists. */
  VerifyEmployeesTable($connection, DB_DATABASE);

  /* If input fields are populated, add a row to the EMPLOYEES table. */
  $employee_name = htmlentities($_POST['NAME']);
  $employee_address = htmlentities($_POST['ADDRESS']);

  if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
  }
?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
```

```
</form>

<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php

  mysqli_free_result($result);
  mysqli_close($connection);

?>

</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = mysqli_real_escape_string($connection, $name);
   $a = mysqli_real_escape_string($connection, $address);
```

```
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</
p>");
  }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = mysqli_real_escape_string($connection, $tableName);
  $d = mysqli_real_escape_string($connection, $dbName);

  $checktable = mysqli_query($connection,
      "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t'
 AND TABLE_SCHEMA = '$d'");

  if(mysqli_num_rows($checktable) > 0) return true;

  return false;
}
?>
```

PostgreSQL

```
<?php include "../inc/dbinfo.inc"; ?>

<html>
<body>
```

```
<h1>Sample page</h1>
<?php

/* Connect to PostgreSQL and select the database. */
$constring = "host=" . DB_SERVER . " dbname=" . DB_DATABASE . " user=" .
 DB_USERNAME . " password=" . DB_PASSWORD ;
$connection = pg_connect($constring);

if (!$connection){
 echo "Failed to connect to PostgreSQL";
 exit;
}

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);

/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
  AddEmployee($connection, $employee_name, $employee_address);
}

?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
    <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
    <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
    <input type="submit" value="Add Data" />
      </td>
    </tr>
```

```
  </table>
</form>
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = pg_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = pg_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>
</table>

<!-- Clean up. -->
<?php

  pg_free_result($result);
  pg_close($connection);
?>
</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = pg_escape_string($name);
   $a = pg_escape_string($address);
   echo "Forming Query";
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";
```

```
   if(!pg_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID serial PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!pg_query($connection, $query)) echo("<p>Error creating table.</p>");
  }
}
/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = strtolower(pg_escape_string($tableName)); //table name is case sensitive
  $d = pg_escape_string($dbName); //schema is 'public' instead of 'sample' db
 name so not using that

  $query = "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME =
 '$t';";
  $checktable = pg_query($connection, $query);

  if (pg_num_rows($checktable) >0) return true;
  return false;

}
?>
```

8. Save and close the `SamplePage.php` file.
9. Verify that your web server successfully connects to your DB instance by opening a web browser and browsing to `http://`*`EC2 instance endpoint`*`/SamplePage.php`, for example: `http://ec2-12-345-67-890.us-west-2.compute.amazonaws.com/SamplePage.php`.

You can use `SamplePage.php` to add data to your DB instance. The data that you add is then displayed on the page. To verify that the data was inserted into the table, install MySQL client on the Amazon EC2 instance. Then connect to the DB instance and query the table.

For information about installing the MySQL client and connecting to a DB instance, see Connecting to your MySQL DB instance.

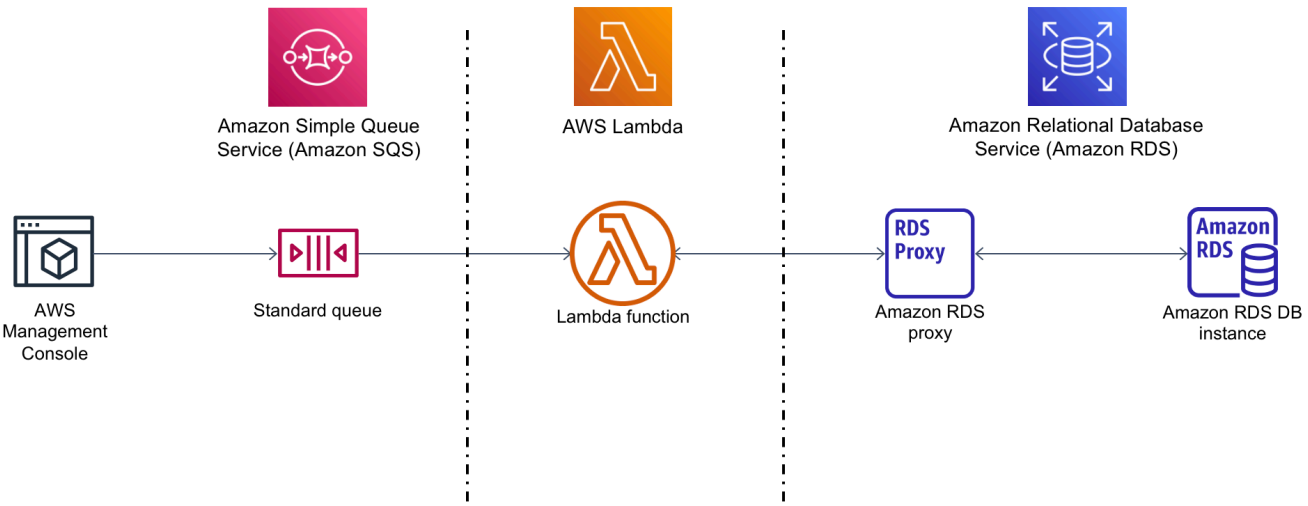
To make sure that your DB instance is as secure as possible, verify that sources outside of the VPC can't connect to your DB instance.

After you have finished testing your web server and your database, you should delete your DB instance and your Amazon EC2 instance.

- To delete a DB instance, follow the instructions in Deleting a DB instance. You don't need to create a final snapshot.
- To terminate an Amazon EC2 instance, follow the instruction in Terminate your instance in the *Amazon EC2 User Guide*.

Tutorial: Using a Lambda function to access an Amazon RDS database

In this tutorial, you use a Lambda function to write data to an Amazon Relational Database Service (Amazon RDS) database through RDS Proxy. Your Lambda function reads records from an Amazon Simple Queue Service (Amazon SQS) queue and writes a new item to a table in your database whenever a message is added. In this example, you use the AWS Management Console to manually add messages to your queue. The following diagram shows the AWS resources you use to complete the tutorial.



With Amazon RDS, you can run a managed relational database in the cloud using common database products like Microsoft SQL Server, MariaDB, MySQL, Oracle Database, and PostgreSQL. By using Lambda to access your database, you can read and write data in response to events, such as a new customer registering with your website. Your function, database instance, and proxy scale automatically to meet periods of high demand.

To complete this tutorial, you carry out the following tasks:

1. Launch an RDS for MySQL database instance and a proxy in your AWS account's default VPC.
2. Create and test a Lambda function that creates a new table in your database and writes data to it.

3. Create an Amazon SQS queue and configure it to invoke your Lambda function whenever a new message is added.
4. Test the complete setup by adding messages to your queue using the AWS Management Console and monitoring the results using CloudWatch Logs.

By completing these steps, you learn:

- How to use Amazon RDS to create a database instance and a proxy, and connect a Lambda function to the proxy.
- How to use Lambda to perform create and read operations on an Amazon RDS database.
- How to use Amazon SQS to invoke a Lambda function.

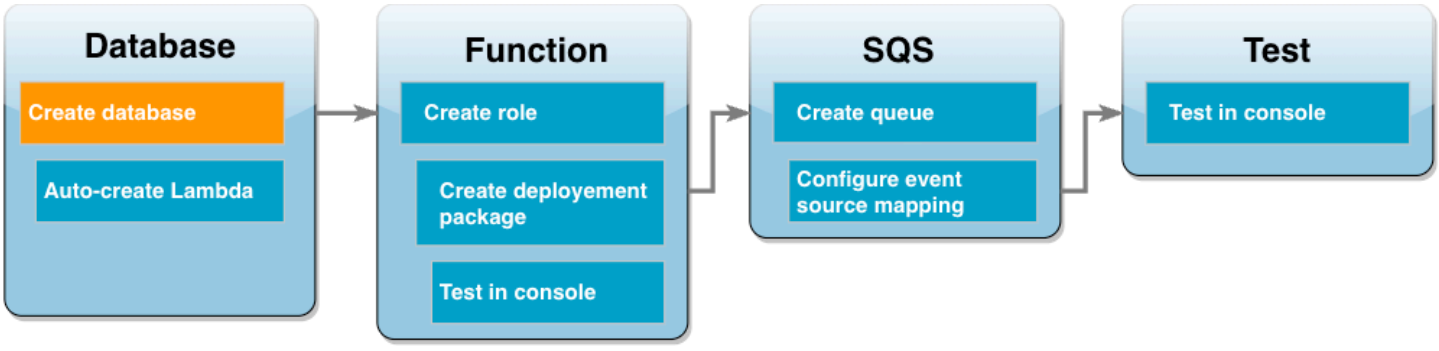
You can complete this tutorial using the AWS Management Console or the AWS Command Line Interface (AWS CLI).

Prerequisites

Before you begin, complete the steps in the following sections:

- Sign up for an AWS account
- Create a user with administrative access

Create an Amazon RDS DB instance



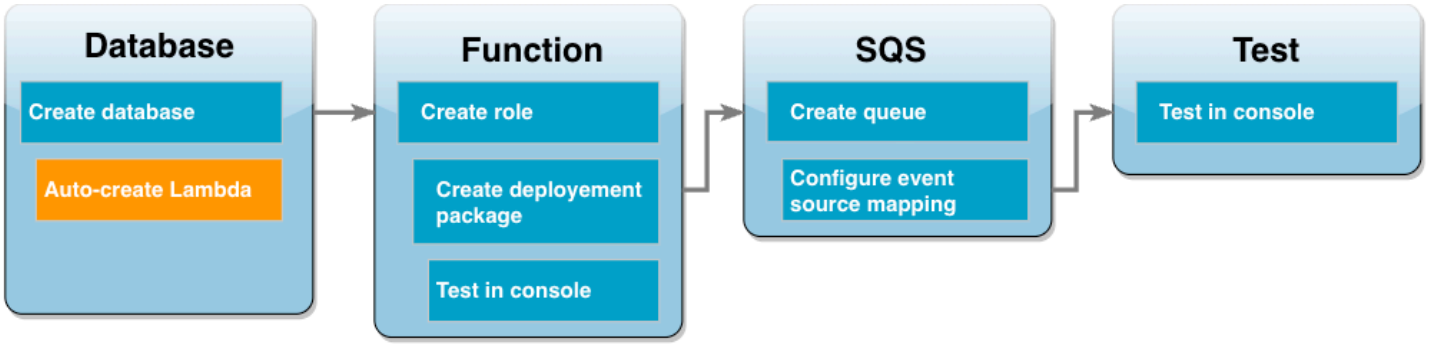
An Amazon RDS DB instance is an isolated database environment running in the AWS Cloud. An instance can contain one or more user-created databases. Unless you specify otherwise, Amazon RDS creates new database instances in the default VPC included in your AWS account. For more information about Amazon VPC, see the Amazon Virtual Private Cloud User Guide.

In this tutorial, you create a new instance in your AWS account's default VPC and create a database named `ExampleDB` in that instance. You can create your DB instance and database using either the AWS Management Console or the AWS CLI.

To create a database instance

1. Open the Amazon RDS console and choose **Create database**.
2. Leave the **Standard create** option selected, then in **Engine options**, choose **MySQL**.
3. In **Templates**, choose **Free tier** or **Sandbox**. **Free tier** appears for free tier accounts. **Sandbox** appears for paid plan accounts.
4. In **Settings**, for **DB instance identifier**, enter **`MySQLForLambda`**.
5. Set your username and password by doing the following:
   a. In **Credentials settings**, leave **Master username** set to `admin`.
   b. For **Master password**, enter and confirm a password to access your database.
6. Specify the database name by doing the following:
   - Leave all the remaining default options selected and scroll down to the **Additional configuration** section.
   - Expand this section and enter **`ExampleDB`** as the **Initial database name**.
7. Leave all the remaining default options selected and choose **Create database**.

Create Lambda function and proxy



You can use the RDS console to create a Lambda function and a proxy in the same VPC as the database.

> **Note**
>
> You can only create these associated resources when your database has completed creation and is in **Available** status.

To create an associated function and proxy

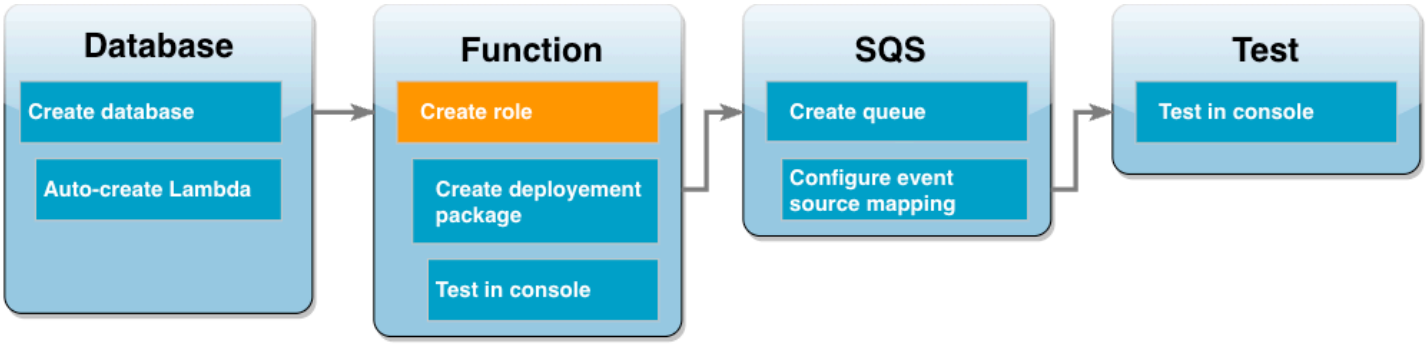
1. From the **Databases** page, check if your database is in the **Available** status. If so, proceed to the next step. Else, wait till your database is available.
2. Select your database and choose **Set up Lambda connection** from **Actions**.
3. In the **Set up Lambda connection** page, choose **Create new function**.

   Set the **New Lambda function name** to **`LambdaFunctionWithRDS`**.
4. In the **RDS Proxy** section, select the **Connect using RDS Proxy** option. Further choose **Create new proxy**.

   - For **Database credentials**, choose **Database username and password**.
   - For **Username**, specify `admin`.
   - For **Password**, enter the password you created for your database instance.
5. Select **Set up** to complete the proxy and Lambda function creation.

The wizard completes the set up and provides a link to the Lambda console to review your new function. Note the proxy endpoint before switching to the Lambda console.

Create a function execution role



Before you create your Lambda function, you create an execution role to give your function the necessary permissions. For this tutorial, Lambda needs permission to manage the network

connection to the VPC containing your database instance and to poll messages from an Amazon SQS queue.

To give your Lambda function the permissions it needs, this tutorial uses IAM managed policies. These are policies that grant permissions for many common use cases and are available in your AWS account. For more information about using managed policies, see Policy best practices.

To create the Lambda execution role

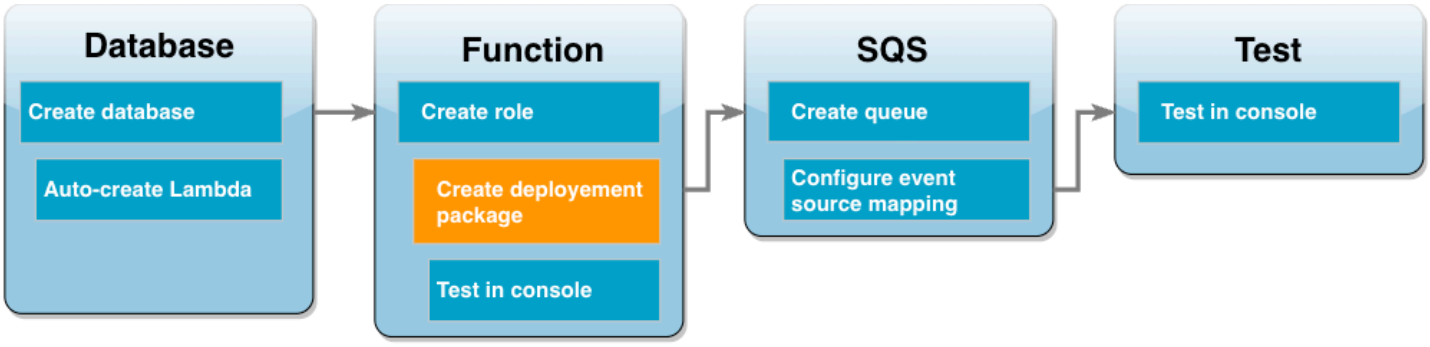
1. Open the Roles page of the IAM console and choose **Create role**.
2. For the **Trusted entity type**, choose **AWS service**, and for the **Use case**, choose **Lambda**.
3. Choose **Next**.
4. Add the IAM managed policies by doing the following:
    a. Using the policy search box, search for **`AWSLambdaSQSQueueExecutionRole`**.
    b. In the results list, select the check box next to the role, then choose **Clear filters**.
    c. Using the policy search box, search for **`AWSLambdaVPCAccessExecutionRole`**.
    d. In the results list, select the check box next to the role, then choose **Next**.
5. For the **Role name**, enter **`lambda-vpc-sqs-role`**, then choose **Create role**.

Later in the tutorial, you need the Amazon Resource Name (ARN) of the execution role you just created.

To find the execution role ARN

1. Open the Roles page of the IAM console and choose your role (`lambda-vpc-sqs-role`).
2. Copy the **ARN** displayed in the **Summary** section.

Create a Lambda deployment package



The following example Python code uses the PyMySQL package to open a connection to your database. The first time you invoke your function, it also creates a new table called `Customer`. The table uses the following schema, where `CustID` is the primary key:

```
Customer(CustID, Name)
```

The function also uses PyMySQL to add records to this table. The function adds records using customer IDs and names specified in messages you will add to your Amazon SQS queue.

The code creates the connection to your database outside of the handler function. Creating the connection in the initialization code allows the connection to be re-used by subsequent invocations of your function and improves performance. In a production application, you can also use provisioned concurrency to initialize a requested number of database connections. These connections are available as soon as your function is invoked.

```
import sys
import logging
import pymysql
import json
import os

# rds settings
user_name = os.environ['USER_NAME']
password = os.environ['PASSWORD']
rds_proxy_host = os.environ['RDS_PROXY_HOST']
db_name = os.environ['DB_NAME']

logger = logging.getLogger()
logger.setLevel(logging.INFO)
```

```
# create the database connection outside of the handler to allow connections to be
# re-used by subsequent function invocations.
try:
        conn = pymysql.connect(host=rds_proxy_host, user=user_name, passwd=password,
 db=db_name, connect_timeout=5)
except pymysql.MySQLError as e:
    logger.error("ERROR: Unexpected error: Could not connect to MySQL instance.")
    logger.error(e)
    sys.exit(1)

logger.info("SUCCESS: Connection to RDS for MySQL instance succeeded")

def lambda_handler(event, context):
    """
    This function creates a new RDS database table and writes records to it
    """
    message = event['Records'][0]['body']
    data = json.loads(message)
    CustID = data['CustID']
    Name = data['Name']

    item_count = 0
    sql_string = f"insert into Customer (CustID, Name) values(%s, %s)"

    with conn.cursor() as cur:
        cur.execute("create table if not exists Customer ( CustID  int NOT NULL, Name
 varchar(255) NOT NULL, PRIMARY KEY (CustID))")
        cur.execute(sql_string, (CustID, Name))
        conn.commit()
        cur.execute("select * from Customer")
        logger.info("The following items have been added to the database:")
        for row in cur:
            item_count += 1
            logger.info(row)
    conn.commit()

    return "Added %d items to RDS for MySQL table" %(item_count)
```

Note

In this example, your database access credentials are stored as environment variables. In production applications, we recommend that you use AWS Secrets Manager as a more

secure option. Note that if your Lambda function is in a VPC, to connect to Secrets Manager you need to create a VPC endpoint. See How to connect to Secrets Manager service within a Virtual Private Cloud to learn more.

To include the PyMySQL dependency with your function code, create a .zip deployment package. The following commands work for Linux, macOS, or Unix:

To create a .zip deployment package

1. Save the example code as a file named `lambda_function.py`.
2. In the same directory in which you created your `lambda_function.py` file, create a new directory named `package` and install the PyMySQL library.

```
mkdir package
pip install --target package pymysql
```

3. Create a zip file containing your application code and the PyMySQL library. In Linux or MacOS, run the following CLI commands. In Windows, use your preferred zip tool to create the `lambda_function.zip` file. Your `lambda_function.py` source code file and the folders containing your dependencies must be installed at the root of the .zip file.

```
cd package
zip -r ../lambda_function.zip .
cd ..
zip lambda_function.zip lambda_function.py
```

You can also create your deployment package using a Python virtual environment. See Deploy Python Lambda functions with .zip file archives.

Update the Lambda function

Using the .zip package you just created, you now update your Lambda function using the Lambda console. To enable your function to access your database, you also need to configure environment variables with your access credentials.

To update the Lambda function

1. Open the Functions page of the Lambda console and choose your function `LambdaFunctionWithRDS`.
2. In the **Runtime settings** tab, select **Edit** to change the **Runtime** of the function to **Python 3.10**.
3. Change the **Handler** to `lambda_function.lambda_handler`.
4. In the **Code** tab, choose **Upload from** and then **.zip file**.
5. Select the `lambda_function.zip` file you created in the previous stage and choose **Save**.

Now configure the function with the execution role you created earlier. This grants the function the permissions it needs to access your database instance and poll an Amazon SQS queue.

To configure the function's execution role

1. In the Functions page of the Lambda console, select the **Configuration** tab, then choose **Permissions**.
2. In **Execution role**, choose **Edit**.
3. In **Existing role**, choose your execution role (`lambda-vpc-sqs-role`).
4. Choose **Save**.

To configure your function's environment variables

1. In the Functions page of the Lambda console, select the **Configuration** tab, then choose **Environment variables**.
2. Choose **Edit**.
3. To add your database access credentials, do the following:

    a. Choose **Add environment variable**, then for **Key** enter **`USER_NAME`** and for **Value** enter **`admin`**.

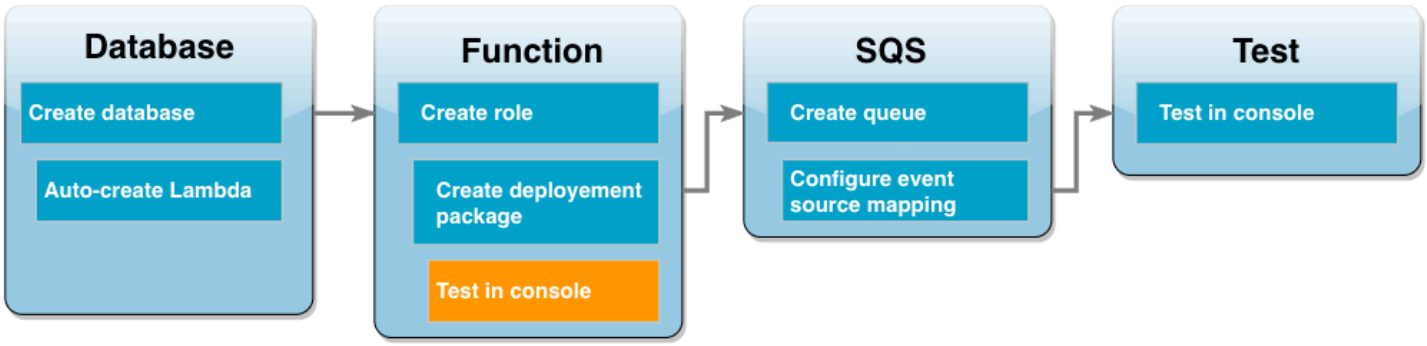
    b. Choose **Add environment variable**, then for **Key** enter **`DB_NAME`** and for **Value** enter **`ExampleDB`**.

    c. Choose **Add environment variable**, then for **Key** enter **`PASSWORD`** and for **Value** enter the password you chose when you created your database.

d. Choose **Add environment variable**, then for **Key** enter **`RDS_PROXY_HOST`** and for **Value** enter the RDS Proxy endpoint you noted earlier.

e. Choose **Save**.

Test your Lambda function in the console



You can now use the Lambda console to test your function. You create a test event which mimics the data your function will receive when you invoke it using Amazon SQS in the final stage of the tutorial. Your test event contains a JSON object specifying a customer ID and customer name to add to the `Customer` table your function creates.

To test the Lambda function

1. Open the Functions page of the Lambda console and choose your function.
2. Choose the **Test** section.
3. Choose **Create new event** and enter **`myTestEvent`** for the event name.
4. Copy the following code into **Event JSON** and choose **Save**.

```
{
  "Records": [
    {
      "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
      "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
      "body": "{\n     \"CustID\": 1021,\n     \"Name\": \"Martha Rivera\"\n}",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOLO23YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
      },
```

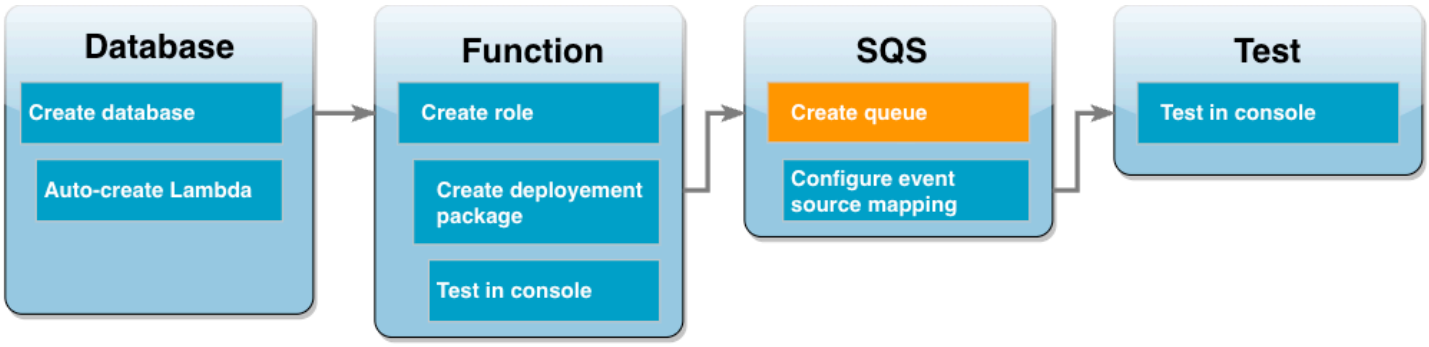
```
      "messageAttributes": {},
      "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
      "eventSource": "aws:sqs",
      "eventSourceARN": "arn:aws:sqs:us-west-2:123456789012:my-queue",
      "awsRegion": "us-west-2"
    }
  ]
}
```

5. Choose **Test**.

In the **Execution results** tab, you should see results similar to the following displayed in the **Function Logs**:

```
[INFO] 2023-02-14T19:31:35.149Z bdd06682-00c7-4d6f-9abb-89f4bbb4a27f The following
 items have been added to the database:
[INFO] 2023-02-14T19:31:35.149Z bdd06682-00c7-4d6f-9abb-89f4bbb4a27f (1021, 'Martha
 Rivera')
```

Create an Amazon SQS queue

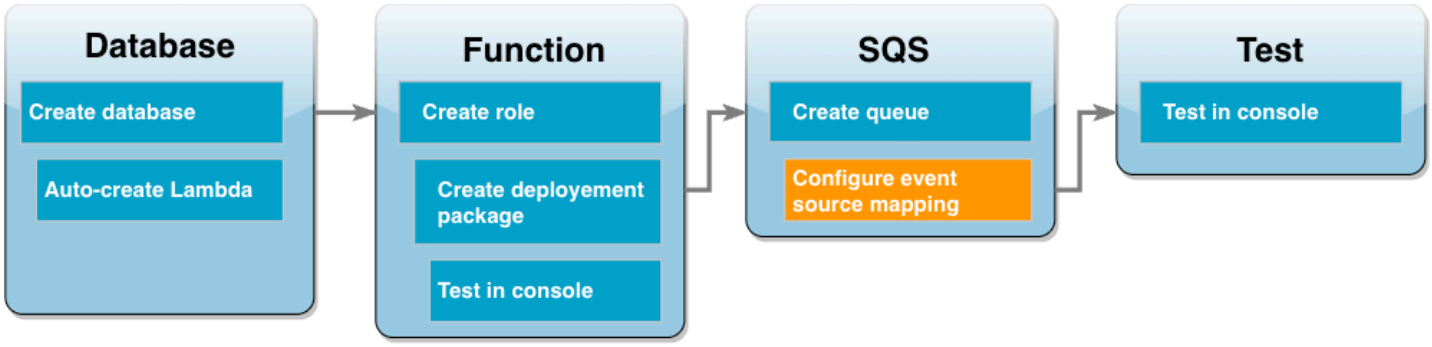


You have successfully tested the integration of your Lambda function and Amazon RDS database instance. Now you create the Amazon SQS queue you will use to invoke your Lambda function in the final stage of the tutorial.

To create the Amazon SQS queue (console)

1. Open the Queues page of the Amazon SQS console and select **Create queue**.
2. Leave the **Type** as **Standard** and enter **`LambdaRDSQueue`** for the name of your queue.
3. Leave all the default options selected and choose **Create queue**.

Create an event source mapping to invoke your Lambda function



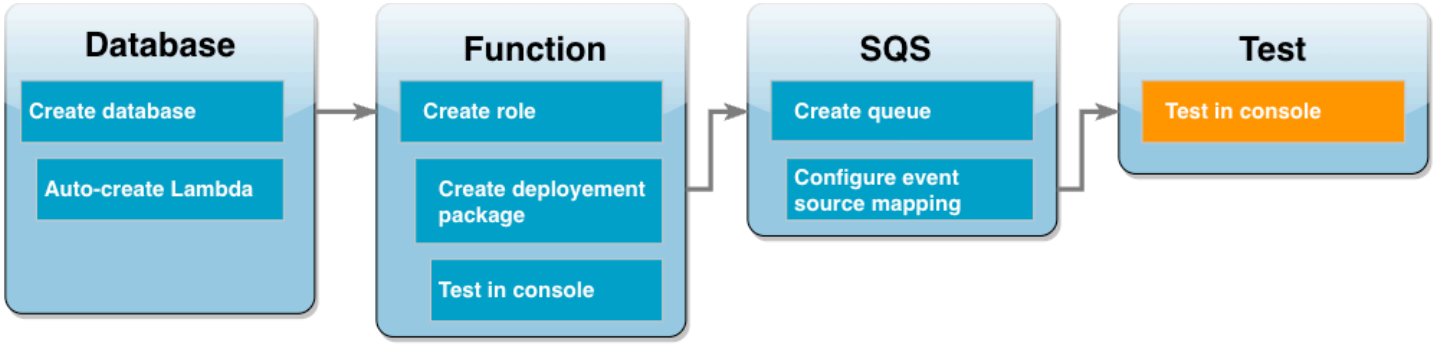
An event source mapping is a Lambda resource which reads items from a stream or queue and invokes a Lambda function. When you configure an event source mapping, you can specify a batch size so that records from your stream or queue are batched together into a single payload. In this example, you set the batch size to 1 so that your Lambda function is invoked every time you send a message to your queue. You can configure the event source mapping using either the AWS CLI or the Lambda console.

To create an event source mapping (console)

1. Open the Functions page of the Lambda console and select your function (`LambdaFunctionWithRDS`).
2. In the **Function overview** section, choose **Add trigger**.
3. For the source, select **Amazon SQS**, then select the name of your queue (`LambdaRDSQueue`).
4. For **Batch size**, enter **1**.
5. Leave all the other options set to the default values and choose **Add**.

You are now ready to test your complete setup by adding a message to your Amazon SQS queue.

Test and monitor your setup



To test your complete setup, add messages to your Amazon SQS queue using the console. You then use CloudWatch Logs to confirm that your Lambda function is writing records to your database as expected.

To test and monitor your setup

1. Open the Queues page of the Amazon SQS console and select your queue (`LambdaRDSQueue`).
2. Choose **Send and receive messages** and paste the following JSON into the **Message body** in the **Send message** section.

```
{
    "CustID": 1054,
    "Name": "Richard Roe"
}
```

3. Choose **Send message**.

   Sending your message to the queue will cause Lambda to invoke your function through your event source mapping. To confirm that Lambda has invoked your function as expected, use CloudWatch Logs to verify that the function has written the customer name and ID to your database table.

4. Open the Log groups page of the CloudWatch console and select the log group for your function (`/aws/lambda/LambdaFunctionWithRDS`).
5. In the **Log streams** section, choose the most recent log stream.

   Your table should contain two customer records, one from each invocation of your function. In the log stream, you should see messages similar to the following:

```
[INFO] 2023-02-14T19:06:43.873Z 45368126-3eee-47f7-88ca-3086ae6d3a77 The following
 items have been added to the database:
[INFO] 2023-02-14T19:06:43.873Z 45368126-3eee-47f7-88ca-3086ae6d3a77 (1021, 'Martha
 Rivera')
[INFO] 2023-02-14T19:06:43.873Z 45368126-3eee-47f7-88ca-3086ae6d3a77 (1054,
 'Richard Roe')
```

Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you're no longer using, you prevent unnecessary charges to your AWS account.

To delete the Lambda function

1. Open the Functions page of the Lambda console.
2. Select the function that you created.
3. Choose **Actions**, **Delete**.
4. Choose **Delete**.

To delete the execution role

1. Open the Roles page of the IAM console.
2. Select the execution role that you created.
3. Choose **Delete role**.
4. Choose **Yes, delete**.

To delete the MySQL DB instance

1. Open the Databases page of the Amazon RDS console.
2. Select the database you created.
3. Choose **Actions**, **Delete**.
4. Clear the **Create final snapshot** check box.
5. Enter **`delete me`** in the text box.
6. Choose **Delete**.

To delete the Amazon SQS queue

1. Sign in to the AWS Management Console and open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Select the queue you created.
3. Choose **Delete**.

4. Enter **`delete`** in the text box.
5. Choose **Delete**.

Amazon RDS tutorials and sample code

The AWS documentation includes several tutorials that guide you through common Amazon RDS use cases. Many of these tutorials show you how to use Amazon RDS with other AWS services. In addition, you can access sample code in .

> **Note**
>
> You can find more tutorials at the AWS Database Blog. For information about training, see AWS Training and Certification.

Topics

- Tutorials in this guide
- Tutorials in other AWS guides
- Tutorials and sample code in GitHub
- AWS Database Cookbook
- AWS workshop and lab content portal for Amazon RDS PostgreSQL
- AWS workshop and lab content portal for Amazon RDS MySQL
- Using this service with an AWS SDK

Tutorials in this guide

The following tutorials in this guide show you how to perform common tasks with Amazon RDS:

- Tutorial: Create a VPC for use with a DB instance (IPv4 only)

  Learn how to include a DB instance in a virtual private cloud (VPC) based on the Amazon VPC service. In this case, the VPC shares data with a web server that is running on an Amazon EC2 instance in the same VPC.
- Tutorial: Create a VPC for use with a DB instance (dual-stack mode)

  Learn how to include a DB instance in a virtual private cloud (VPC) based on the Amazon VPC service. In this case, the VPC shares data with an Amazon EC2 instance in the same VPC. In this

tutorial, you create the VPC for this scenario that works with a database running in dual-stack mode.

- Tutorial: Create a web server and an Amazon RDS DB instance

  Learn how to install an Apache web server with PHP and create a MySQL database. The web server runs on an Amazon EC2 instance using Amazon Linux, and the MySQL database is a MySQL DB instance. Both the Amazon EC2 instance and the DB instance run in an Amazon VPC.

- Tutorial: Restore an Amazon RDS DB instance from a DB snapshot

  Learn how to restore a DB instance from a DB snapshot.

- Tutorial: Using a Lambda function to access an Amazon RDS database

  Learn how to create a Lambda function from the RDS console to access a database through a proxy, create a table, add a few records, and retrieve the records from the table. You also learn how to invoke the Lambda function and verify the query results.

- Tutorial: Specify which DB instances to stop by using tags

  Learn how to use tags to specify which DB instances to stop.

- Tutorial: Log DB instance state changes using Amazon EventBridge

  Learn how to log a DB instance state change using Amazon EventBridge and AWS Lambda.

- Tutorial: Creating an Amazon CloudWatch alarm for Multi-AZ DB cluster replica lag for Amazon RDS

  Learn how to create a CloudWatch alarm that sends an Amazon SNS message when replica lag for a Multi-AZ DB cluster has exceeded a threshold. An alarm watches the `ReplicaLag` metric over a time period that you specify. The action is a notification sent to an Amazon SNS topic or Amazon EC2 Auto Scaling policy.

Tutorials in other AWS guides

The following tutorials in other AWS guides show you how to perform common tasks with Amazon RDS:

- Tutorial: Rotating a Secret for an AWS Database in the *AWS Secrets Manager User Guide*

Learn how to create a secret for an AWS database and configure the secret to rotate on a schedule. You trigger one rotation manually, and then confirm that the new version of the secret continues to provide access.

- Tutorials and samples in the *AWS Elastic Beanstalk Developer Guide*

  Learn how to deploy applications that use Amazon RDS databases with AWS Elastic Beanstalk.

- Using Data from an Amazon RDS Database to Create an Amazon ML Datasource in the *Amazon Machine Learning Developer Guide*

  Learn how to create an Amazon Machine Learning (Amazon ML) datasource object from data stored in a MySQL DB instance.

- Manually Enabling Access to an Amazon RDS Instance in a VPC in the *Amazon Quick User Guide*

  Learn how to enable Quick access to an Amazon RDS DB instance in a VPC.

Tutorials and sample code in GitHub

The following tutorials and sample code in GitHub show you how to perform common tasks with Amazon RDS:

- Creating the Amazon Relational Database Service item tracker

  Learn how to create an application that tracks and reports on work items. This application uses Amazon RDS, Amazon Simple Email Service, Elastic Beanstalk, and SDK for Java 2.x.

AWS Database Cookbook

The AWS DB Cookbook is a comprehensive database guide that teaches you how to build, deploy, and manage high-performing, cost-effective database solutions on AWS. Step-by-step tutorials guide you through creating production-ready applications and deploying the apps with CloudFormation templates. You'll learn essential AWS services as you build infrastructure, implement networking, develop serverless architectures, manage databases, and integrate generative AI. Learn AWS best practices that help you create secure, scalable solutions while optimizing costs. Whether you're new to AWS or an experienced professional, the AWS DB Cookbook helps you develop skills to solve common database challenges and implement enterprise-ready solutions. The cookbook includes the following sections:

- **Getting started with AWS for DB applications** – Learn AWS fundamentals like how to set up your account and Jupyter Notebook environment.
- **Database fundamentals** – Explore essential database concepts and compare AWS database services to choose the right solution for your workloads.
- **Serverless web app with Amazon Aurora** – Build an end-to-end retail application with Amazon Aurora PostgreSQL that handles inventory, orders, and customer data.
- **Monitoring and observability** – Set up performance tracking and configure alerts to identify potential database issues before they impact your applications.
- **Scaling with Amazon Aurora** – Learn to build resilient multi-Region deployments with Aurora DSQL, and how to scale your databases up for more processing power or out across multiple instances for greater capacity.
- **Optimization performance and cost** – Optimize your database performance and reduce costs with proven tuning strategies.
- **Journey to AWS purpose-built databases** – Build a secure, reliable infrastructure that scales your generative AI solutions and data-driven applications from prototype to enterprise deployment.
- **GenAI applications with RAG** – Build an intelligent search system for insurance and healthcare documents that uses Retrieval Augmented Generation (RAG) to deliver accurate, context-aware results.

AWS workshop and lab content portal for Amazon RDS PostgreSQL

The following collection of workshops and other hands-on content helps you to gain an understanding of the Amazon RDS PostgreSQL features and capabilities:

- Creating a DB instance

  Learn how to create the DB instance.

- Performance Monitoring with RDS Tools

  Learn how to use AWS and SQL tools(Cloudwatch, Enhanced Monitoring, Slow Query Logs, Performance Insights, PostgreSQL Catalog Views) to understand performance issues and identify ways to improve performance of your database.

AWS workshop and lab content portal for Amazon RDS MySQL

The following collection of workshops and other hands-on content helps you to gain an understanding of the Amazon RDS MySQL features and capabilities:

- Creating a DB instance

  Learn how to create the DB instance.

- Using Performance Insights

  Learn how to monitor and tune your DB instance using Performance insights.

Using this service with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	AWS Tools for PowerShell code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples

SDK documentation	Code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to this service, see Code examples for Amazon RDS using AWS SDKs.

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Best practices for Amazon RDS

Learn best practices for working with Amazon RDS. As new best practices are identified, we will keep this section up to date.

Topics

- Amazon RDS basic operational guidelines
- DB instance RAM recommendations
- Keeping database engine versions up to date
- AWS database drivers
- Using Enhanced Monitoring to identify operating system issues
- Using metrics to identify performance issues
- Tuning queries
- Best practices for working with MySQL
- Best practices for working with MariaDB
- Best practices for working with Oracle
- Best practices for working with PostgreSQL
- Best practices for working with SQL Server
- Working with DB parameter groups
- Best practices for automating DB instance creation
- Amazon RDS new features video

> **Note**
>
> For common recommendations for Amazon RDS, see Recommendations from Amazon RDS.

Amazon RDS basic operational guidelines

The following are basic operational guidelines that everyone should follow when working with Amazon RDS. Note that the Amazon RDS Service Level Agreement requires that you follow these guidelines:

- Use metrics to monitor your memory, CPU, replica lag, and storage usage. You can set up Amazon CloudWatch to notify you when the usage patterns change or when your deployment approaches capacity limits. This allows you to maintain system performance and availability.
- Scale up your DB instance when you are approaching storage capacity limits. You should have some buffer in storage and memory to accommodate unforeseen increases in demand from your applications.
- Enable automatic backups and set the backup window to occur during the daily low in write IOPS. That's when a backup is least disruptive to your database usage.
- If your database workload requires more I/O than you have provisioned, recovery after a failover or database failure will be slow. To increase the I/O capacity of a DB instance, do any or all of the following:
  - Migrate to a different DB instance class with high I/O capacity.
  - Convert from magnetic storage to either General Purpose or Provisioned IOPS storage, depending on how much of an increase you need. For information on available storage types, see Amazon RDS storage types.

    If you convert to Provisioned IOPS storage, make sure you also use a DB instance class that is optimized for Provisioned IOPS. For information on Provisioned IOPS, see Provisioned IOPS SSD storage.
  - If you are already using Provisioned IOPS storage, provision additional throughput capacity.
- If your client application is caching the Domain Name Service (DNS) data of your DB instances, set a time-to-live (TTL) value of less than 30 seconds. The underlying IP address of a DB instance can change after a failover. Caching the DNS data for an extended time can thus lead to connection failures. Your application might try to connect to an IP address that's no longer in service.
- Test failover for your DB instance to understand how long the process takes for your particular use case. Also test failover to ensure that the application that accesses your DB instance can automatically connect to the new DB instance after failover occurs.

DB instance RAM recommendations

An Amazon RDS performance best practice is to allocate enough RAM so that your *working set* resides almost completely in memory. The working set is the data and indexes that are frequently in use on your instance. The more you use the DB instance, the more the working set will grow.

To tell if your working set is almost all in memory, check the ReadIOPS metric (using Amazon CloudWatch) while the DB instance is under load. The value of ReadIOPS should be small and stable. In some cases, scaling up the DB instance class to a class with more RAM results in a dramatic drop in ReadIOPS. In these cases, your working set was not almost completely in memory. Continue to scale up until ReadIOPS no longer drops dramatically after a scaling operation, or ReadIOPS is reduced to a very small amount. For information on monitoring a DB instance's metrics, see Viewing metrics in the Amazon RDS console.

Keeping database engine versions up to date

Regularly upgrade your database engine version to maintain security, performance, and compliance. Amazon RDS releases new minor and major versions that include security patches, performance enhancements, and new features. Running an outdated database engine can expose your workloads to known vulnerabilities, compatibility issues, and reduced support from AWS and database vendors.

To minimize disruption, consider the following when you plan upgrades:

- **Test in a staging environment** – Validate the new version against your workload before you upgrade production databases.
- **Use Amazon RDS managed upgrades** – Enable automatic minor version upgrades for easier patching.
- **Schedule major version upgrades** – Review release notes, test application compatibility, and plan a controlled upgrade window.

Regular upgrades help ensure your database remains secure, optimized, and aligned with AWS best practices.

AWS database drivers

We recommend the AWS suite of drivers for application connectivity. The drivers have been designed to provide support for faster switchover and failover times, and authentication with AWS Secrets Manager, AWS Identity and Access Management (IAM), and Federated Identity. The AWS drivers rely on monitoring DB instance status and being aware of the instance topology to determine the new writer. This approach reduces switchover and failover times to single-digit seconds, compared to tens of seconds for open-source drivers.

As new service features are introduced, the goal of the AWS suite of drivers is to have built-in support for these service features.

For more information, see Connecting to DB instances with the AWS drivers.

Using Enhanced Monitoring to identify operating system issues

When Enhanced Monitoring is enabled, Amazon RDS provides metrics in real time for the operating system (OS) that your DB instance runs on. You can view the metrics for your DB instance using the console. You can also consume the Enhanced Monitoring JSON output from Amazon CloudWatch Logs in a monitoring system of your choice. For more information about Enhanced Monitoring, see Monitoring OS metrics with Enhanced Monitoring.

Using metrics to identify performance issues

To identify performance issues caused by insufficient resources and other common bottlenecks, you can monitor the metrics available for your Amazon RDS DB instance.

Viewing performance metrics

You should monitor performance metrics on a regular basis to see the average, maximum, and minimum values for a variety of time ranges. If you do so, you can identify when performance is degraded. You can also set Amazon CloudWatch alarms for particular metric thresholds so you are alerted if they are reached.

To troubleshoot performance issues, it's important to understand the baseline performance of the system. When you set up a DB instance and run it with a typical workload, capture the average, maximum, and minimum values of all performance metrics. Do so at a number of different intervals (for example, one hour, 24 hours, one week, two weeks). This can give you an idea of what is normal. It helps to get comparisons for both peak and off-peak hours of operation. You can then use this information to identify when performance is dropping below standard levels.

If you use Multi-AZ DB clusters, monitor the time difference between the latest transaction on the writer DB instance and the latest applied transaction on a reader DB instance. This difference is called *replica lag*. For more information, see Replica lag and Multi-AZ DB clusters.

You can view the combined Performance Insights and CloudWatch metrics in the Performance Insights dashboard and monitor your DB instance. To use this monitoring view, Performance

Insights must be turned on for your DB instance. For information about this monitoring view, see Viewing combined metrics with the Performance Insights dashboard.

You can create a performance analysis report for a specific time period and view the insights identified and the recommendations to resolve the issues. For more information see, Creating a performance analysis report in Performance Insights.

To view performance metrics

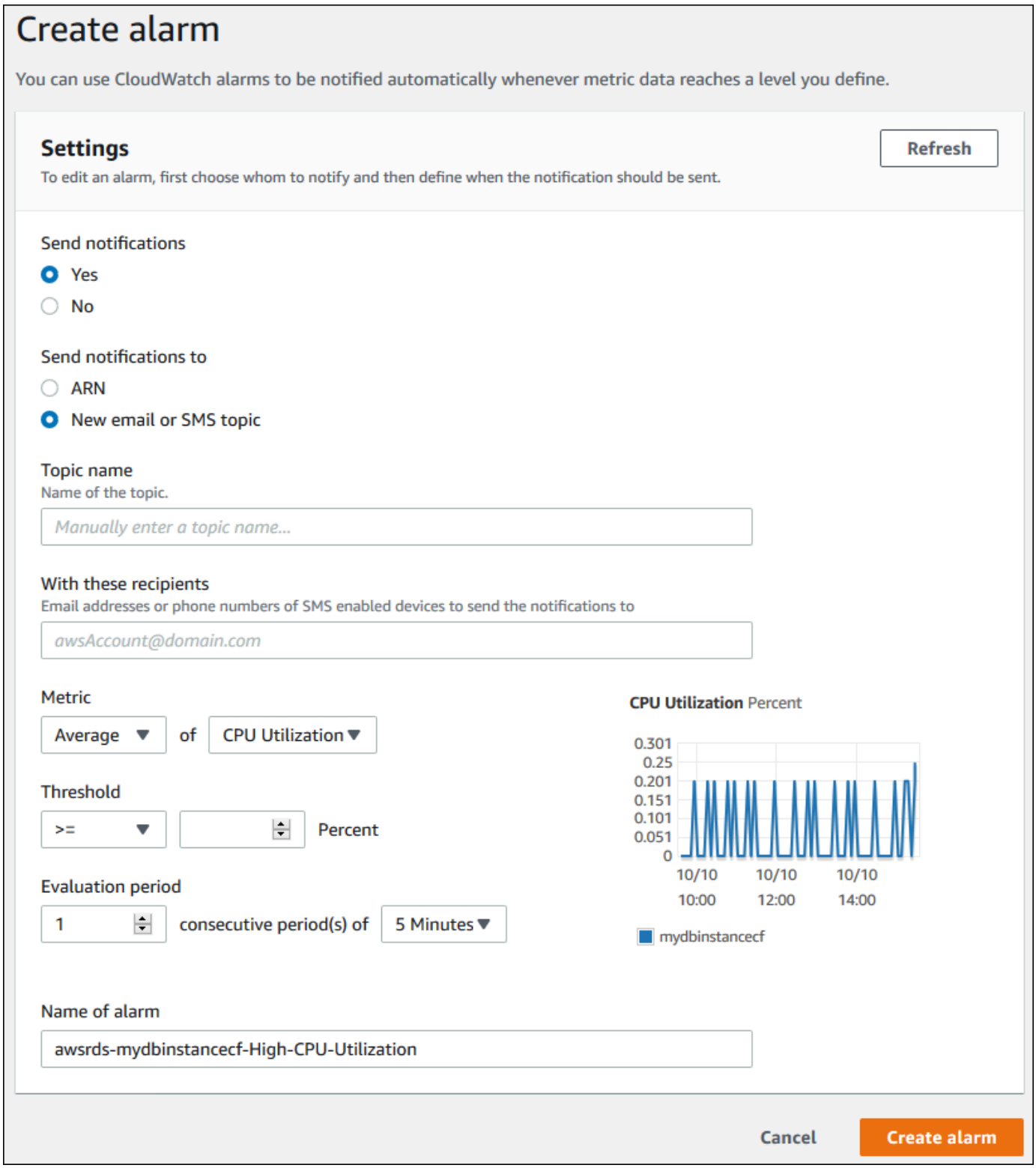
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose a DB instance.
3. Choose **Monitoring**.

   The dashboard provides the performance metrics. The metrics default to show the information for the last three hours.
4. Use the numbered buttons in the upper-right to page through the additional metrics, or adjust the settings to see more metrics.
5. Choose a performance metric to adjust the time range in order to see data for other than the current day. You can change the **Statistic**, **Time Range**, and **Period** values to adjust the information displayed. For example, you might want to see the peak values for a metric for each day of the last two weeks. If so, set **Statistic** to **Maximum**, **Time Range** to **Last 2 Weeks**, and **Period** to **Day**.

You can also view performance metrics using the CLI or API. For more information, see Viewing metrics in the Amazon RDS console.

To set a CloudWatch alarm

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose a DB instance.
3. Choose **Logs & events**.
4. In the **CloudWatch alarms** section, choose **Create alarm**.



5. For **Send notifications**, choose **Yes**, and for **Send notifications to**, choose **New email or SMS topic**.

6. For **Topic name**, enter a name for the notification, and for **With these recipients**, enter a comma-separated list of email addresses and phone numbers.
7. For **Metric**, choose the alarm statistic and metric to set.
8. For **Threshold**, specify whether the metric must be greater than, less than, or equal to the threshold, and specify the threshold value.
9. For **Evaluation period**, choose the evaluation period for the alarm. For **consecutive period(s) of**, choose the period during which the threshold must have been reached in order to trigger the alarm.
10. For **Name of alarm**, enter a name for the alarm.
11. Choose **Create Alarm**.

The alarm appears in the **CloudWatch alarms** section.

Evaluating performance metrics

A DB instance has a number of different categories of metrics, and how to determine acceptable values depends on the metric.

CPU

- CPU Utilization – Percentage of computer processing capacity used.

Memory

- Freeable Memory – How much RAM is available on the DB instance, in bytes. The red line in the Monitoring tab metrics is marked at 75% for CPU, Memory and Storage Metrics. If instance memory consumption frequently crosses that line, then this indicates that you should check your workload or upgrade your instance.
- Swap Usage – How much swap space is used by the DB instance, in bytes.

Disk space

- Free Storage Space – How much disk space is not currently being used by the DB instance, in megabytes.

Input/output operations

- Read IOPS, Write IOPS – The average number of disk read or write operations per second.
- Read Latency, Write Latency – The average time for a read or write operation in milliseconds.
- Read Throughput, Write Throughput – The average number of megabytes read from or written to disk per second.
- Queue Depth – The number of I/O operations that are waiting to be written to or read from disk.

Network traffic

- Network Receive Throughput, Network Transmit Throughput – The rate of network traffic to and from the DB instance in bytes per second.

Database connections

- DB Connections – The number of client sessions that are connected to the DB instance.

For more detailed individual descriptions of the performance metrics available, see Monitoring Amazon RDS metrics with Amazon CloudWatch.

Generally speaking, acceptable values for performance metrics depend on what your baseline looks like and what your application is doing. Investigate consistent or trending variances from your baseline. Advice about specific types of metrics follows:

- **High CPU or RAM consumption –** High values for CPU or RAM consumption might be appropriate. For example, they might be so if they are in keeping with your goals for your application (like throughput or concurrency) and are expected.
- **Disk space consumption –** Investigate disk space consumption if space used is consistently at or above 85 percent of the total disk space. See if it is possible to delete data from the instance or archive data to a different system to free up space.
- **Network traffic –** For network traffic, talk with your system administrator to understand what expected throughput is for your domain network and internet connection. Investigate network traffic if throughput is consistently lower than expected.
- **Database connections –** Consider constraining database connections if you see high numbers of user connections in conjunction with decreases in instance performance and response time. The best number of user connections for your DB instance will vary based on your instance class

and the complexity of the operations being performed. To determine the number of database connections, associate your DB instance with a parameter group. In this group, set the *User Connections* parameter to other than 0 (unlimited). You can either use an existing parameter group or create a new one. For more information, see Parameter groups for Amazon RDS.

- **IOPS metrics –** The expected values for IOPS metrics depend on disk specification and server configuration, so use your baseline to know what is typical. Investigate if values are consistently different than your baseline. For best IOPS performance, make sure your typical working set will fit into memory to minimize read and write operations.

For issues with performance metrics, a first step to improve performance is to tune the most used and most expensive queries. Tune them to see if doing so lowers the pressure on system resources. For more information, see Tuning queries.

If your queries are tuned and an issue persists, consider upgrading your Amazon RDS DB instance classes. You might upgrade it to one with more of the resource (CPU, RAM, disk space, network bandwidth, I/O capacity) that is related to the issue.

Tuning queries

One of the best ways to improve DB instance performance is to tune your most commonly used and most resource-intensive queries. Here, you tune them to make them less expensive to run. For information on improving queries, use the following resources:

- MySQL – See Optimizing SELECT statements in the MySQL documentation. For additional query tuning resources, see MySQL performance tuning and optimization resources.
- Oracle – See Database SQL Tuning Guide in the Oracle Database documentation.
- SQL Server – See Analyzing a query in the Microsoft documentation. You can also use the execution-, index-, and I/O-related data management views (DMVs) described in System Dynamic Management Views in the Microsoft documentation to troubleshoot SQL Server query issues.

  A common aspect of query tuning is creating effective indexes. For potential index improvements for your DB instance, see Database Engine Tuning Advisor in the Microsoft documentation. For information on using Tuning Advisor on RDS for SQL Server, see Analyzing your database workload on an Amazon RDS for SQL Server DB instance with Database Engine Tuning Advisor.
- PostgreSQL – See Using EXPLAIN in the PostgreSQL documentation to learn how to analyze a query plan. You can use this information to modify a query or underlying tables in order to improve query performance.

For information about how to specify joins in your query for the best performance, see Controlling the planner with explicit JOIN clauses.

- MariaDB – See Query optimizations in the MariaDB documentation.

Best practices for working with MySQL

Both table sizes and number of tables in a MySQL database can affect performance.

Table size

Typically, operating system constraints on file sizes determine the effective maximum table size for MySQL databases. So, the limits usually aren't determined by internal MySQL constraints.

On a MySQL DB instance, avoid tables in your database growing too large. Although the general storage limit is 64 TiB, provisioned storage limits restrict the maximum size of a MySQL table file to 16 TiB. Partition your large tables so that file sizes are well under the 16 TiB limit. This approach can also improve performance and recovery time. For more information, see MySQL file size limits in Amazon RDS.

Very large tables (greater than 100 GB in size) can negatively affect performance for both reads and writes (including DML statements and especially DDL statements). Indexes on larges tables can significantly improve select performance, but they can also degrade the performance of DML statements. DDL statements, such as `ALTER TABLE`, can be significantly slower for the large tables because those operations might completely rebuild a table in some cases. These DDL statements might lock the tables for the duration of the operation.

The amount of memory required by MySQL for reads and writes depends on the tables involved in the operations. It is a best practice to have at least enough RAM to the hold the indexes of *actively* used tables. To find the ten largest tables and indexes in a database, use the following query:

```
select table_schema, TABLE_NAME, dat, idx from
(SELECT table_schema, TABLE_NAME,
       ( data_length ) / 1024 / 1024 as dat,
       ( index_length ) / 1024 / 1024 as idx
FROM information_schema.TABLES
order by 3 desc ) a
order by 3 desc
limit 10;
```

Number of tables

Your underlying file system might have a limit on the number of files that represent tables. However, MySQL has no limit on the number of tables. Despite this, the total number of tables in the MySQL InnoDB storage engine can contribute to the performance degradation, regardless of the size of those tables. To limit the operating system impact, you can split the tables across multiple databases in the same MySQL DB instance. Doing so might limit the number of files in a directory but won't solve the overall problem.

When there is performance degradation because of a large number of tables (more than 10 thousand), it is caused by MySQL working with storage files, including opening and closing them. To address this issue, you can increase the size of the `table_open_cache` and `table_definition_cache` parameters. However, increasing the values of those parameters might significantly increase the amount of memory MySQL uses, and might even use all of the available memory. For more information, see How MySQL Opens and Closes Tables in the MySQL documentation.

In addition, too many tables can significantly affect MySQL startup time. Both a clean shutdown and restart and a crash recovery can be affected, especially in versions prior to MySQL 8.0.

We recommend having fewer than 10,000 tables total across all of the databases in a DB instance. For a use case with a large number of tables in a MySQL database, see One Million Tables in MySQL 8.0.

Storage engine

The point-in-time restore and snapshot restore features of Amazon RDS for MySQL require a crash-recoverable storage engine. These features are supported for the InnoDB storage engine only. Although MySQL supports multiple storage engines with varying capabilities, not all of them are optimized for crash recovery and data durability. For example, the MyISAM storage engine doesn't support reliable crash recovery and might prevent a point-in-time restore or snapshot restore from working as intended. This might result in lost or corrupt data when MySQL is restarted after a crash.

InnoDB is the recommended and supported storage engine for MySQL DB instances on Amazon RDS. InnoDB instances can also be migrated to Aurora, while MyISAM instances can't be migrated. However, MyISAM performs better than InnoDB if you require intense, full-text search capability. If you still choose to use MyISAM with Amazon RDS, following the steps outlined in Automated

backups with unsupported MySQL storage engines can be helpful in certain scenarios for snapshot restore functionality.

If you want to convert existing MyISAM tables to InnoDB tables, you can use the process outlined in Converting Tables from MyISAM to InnoDB in the MySQL documentation. MyISAM and InnoDB have different strengths and weaknesses, so you should fully evaluate the impact of making this switch on your applications before doing so.

In addition, Federated Storage Engine is currently not supported by Amazon RDS for MySQL.

Best practices for working with MariaDB

Both table sizes and number of tables in a MariaDB database can affect performance.

Table size

Typically, operating system constraints on file sizes determine the effective maximum table size for MariaDB databases. So, the limits usually aren't determined by internal MariaDB constraints.

On a MariaDB DB instance, avoid tables in your database growing too large. Although the general storage limit is 64 TiB, provisioned storage limits restrict the maximum size of a MariaDB table file to 16 TiB. Partition your large tables so that file sizes are well under the 16 TiB limit. This approach can also improve performance and recovery time.

Very large tables (greater than 100 GB in size) can negatively affect performance for both reads and writes (including DML statements and especially DDL statements). Indexes on larges tables can significantly improve select performance, but they can also degrade the performance of DML statements. DDL statements, such as `ALTER TABLE`, can be significantly slower for the large tables because those operations might completely rebuild a table in some cases. These DDL statements might lock the tables for the duration of the operation.

The amount of memory required by MariaDB for reads and writes depends on the tables involved in the operations. It is a best practice to have at least enough RAM to the hold the indexes of *actively* used tables. To find the ten largest tables and indexes in a database, use the following query:

```
select table_schema, TABLE_NAME, dat, idx from
(SELECT table_schema, TABLE_NAME,
```

```
       ( data_length ) / 1024 / 1024 as dat,
       ( index_length ) / 1024 / 1024 as idx
FROM information_schema.TABLES
order by 3 desc ) a
order by 3 desc
limit 10;
```

Number of tables

Your underlying file system might have a limit on the number of files that represent tables. However, MariaDB has no limit on the number of tables. Despite this, the total number of tables in the MariaDB InnoDB storage engine can contribute to the performance degradation, regardless of the size of those tables. To limit the operating system impact, you can split the tables across multiple databases in the same MariaDB DB instance. Doing so might limit the number of files in a directory but doesn't solve the overall problem.

When there is performance degradation because of a large number of tables (more than 10,000), it's caused by MariaDB working with storage files. This work includes MariaDB opening and closing storage files. To address this issue, you can increase the size of the `table_open_cache` and `table_definition_cache` parameters. However, increasing the values of those parameters might significantly increase the amount of memory MariaDB uses. It might even use all of the available memory. For more information, see Optimizing table_open_cache in the MariaDB documentation.

In addition, too many tables can significantly affect MariaDB startup time. Both a clean shutdown and restart and a crash recovery can be affected. We recommend having fewer than ten thousand tables total across all of the databases in a DB instance.

Storage engine

The point-in-time restore and snapshot restore features of Amazon RDS for MariaDB require a crash-recoverable storage engine. Although MariaDB supports multiple storage engines with varying capabilities, not all of them are optimized for crash recovery and data durability. For example, although Aria is a crash-safe replacement for MyISAM, it might still prevent a point-in-time restore or snapshot restore from working as intended. This might result in lost or corrupt data when MariaDB is restarted after a crash. InnoDB is the recommended and supported storage engine for MariaDB DB instances on Amazon RDS. If you still choose to use Aria with Amazon RDS, following the steps outlined in Automated backups with unsupported MariaDB storage engines can be helpful in certain scenarios for snapshot restore functionality.

If you want to convert existing MyISAM tables to InnoDB tables, you can use the process outlined in Converting Tables from MyISAM to InnoDB in the MariaDB documentation. MyISAM and InnoDB have different strengths and weaknesses, so you should fully evaluate the impact of making this switch on your applications before doing so.

Best practices for working with Oracle

For information about best practices for working with Amazon RDS for Oracle, see Best practices for running Oracle database on Amazon Web Services.

A 2020 AWS virtual workshop included a presentation on running production Oracle databases on Amazon RDS. A video of the presentation is available here.

Best practices for working with PostgreSQL

Of two important areas where you can improve performance with RDS for PostgreSQL, one is when loading data into a DB instance. Another is when using the PostgreSQL autovacuum feature. The following sections cover some of the practices we recommend for these areas.

For information on how Amazon RDS implements other common PostgreSQL DBA tasks, see Common DBA tasks for Amazon RDS for PostgreSQL.

Loading data into a PostgreSQL DB instance

When loading data into an Amazon RDS for PostgreSQL DB instance, modify your DB instance settings and your DB parameter group values. Set these to allow for the most efficient importing of data into your DB instance.

Modify your DB instance settings to the following:

- Disable DB instance backups (set backup_retention to 0)
- Disable Multi-AZ

Modify your DB parameter group to include the following settings. Also, test the parameter settings to find the most efficient settings for your DB instance.

- Increase the value of the `maintenance_work_mem` parameter. For more information about PostgreSQL resource consumption parameters, see the PostgreSQL documentation.

- Increase the value of the `max_wal_size` and `checkpoint_timeout` parameters to reduce the number of writes to the write-ahead log (WAL) log.
- Disable the `synchronous_commit` parameter.
- Disable the PostgreSQL autovacuum parameter.
- Make sure that none of the tables you're importing are unlogged. Data stored in unlogged tables can be lost during a failover. For more information, see CREATE TABLE UNLOGGED.

Use the `pg_dump -Fc` (compressed) or `pg_restore -j` (parallel) commands with these settings.

After the load operation completes, return your DB instance and DB parameters to their normal settings.

Working with the PostgreSQL autovacuum feature

The autovacuum feature for PostgreSQL databases is a feature that we strongly recommend you use to maintain the health of your PostgreSQL DB instance. Autovacuum automates the execution of the VACUUM and ANALYZE command Using autovacuum is required by PostgreSQL, not imposed by Amazon RDS, and its use is critical to good performance. The feature is enabled by default for all new Amazon RDS for PostgreSQL DB instances, and the related configuration parameters are appropriately set by default.

Your database administrator needs to know and understand this maintenance operation. For the PostgreSQL documentation on autovacuum, see The Autovacuum Daemon.

Autovacuum is not a "resource free" operation, but it works in the background and yields to user operations as much as possible. When enabled, autovacuum checks for tables that have had a large number of updated or deleted tuples. It also protects against loss of very old data due to transaction ID wraparound. For more information, see Preventing transaction ID wraparound failures.

Autovacuum should not be thought of as a high-overhead operation that can be reduced to gain better performance. On the contrary, tables that have a high velocity of updates and deletes will quickly deteriorate over time if autovacuum is not run.

> **Important**
>
> Not running autovacuum can result in an eventual required outage to perform a much more intrusive vacuum operation. In some cases, an RDS for PostgreSQL DB instance might

become unavailable because of an over-conservative use of autovacuum. In these cases, the PostgreSQL database shuts down to protect itself. At that point, Amazon RDS must perform a single-user-mode full vacuum directly on the DB instance. This full vacuum can result in a multi-hour outage. Thus, we strongly recommend that you do not turn off autovacuum, which is turned on by default.

The autovacuum parameters determine when and how hard autovacuum works. The`autovacuum_vacuum_threshold` and `autovacuum_vacuum_scale_factor` parameters determine when autovacuum is run. The `autovacuum_max_workers`, `autovacuum_nap_time`, `autovacuum_cost_limit`, and `autovacuum_cost_delay` parameters determine how hard autovacuum works. For more information about autovacuum, when it runs, and what parameters are required, see Routine Vacuuming in the PostgreSQL documentation.

The following query shows the number of "dead" tuples in a table named table1:

```
SELECT relname, n_dead_tup, last_vacuum, last_autovacuum FROM
pg_catalog.pg_stat_all_tables
WHERE n_dead_tup > 0 and relname = 'table1';
```

The results of the query will resemble the following:

```
relname | n_dead_tup | last_vacuum | last_autovacuum
---------+------------+-------------+-----------------
 tasks   |   81430522 |             |
(1 row)
```

Amazon RDS for PostgreSQL best practices video

The 2020 AWS re:Invent conference included a presentation on new features and best practices for working with PostgreSQL on Amazon RDS. A video of the presentation is available here.

Best practices for working with SQL Server

Best practices for a Multi-AZ deployment with a SQL Server DB instance include the following:

- Use Amazon RDS DB events to monitor failovers. For example, you can be notified by text message or email when a DB instance fails over. For more information about Amazon RDS events, see Working with Amazon RDS event notification.

- If your application caches DNS values, set time to live (TTL) to less than 30 seconds. Setting TTL as so is a good practice in case there is a failover. In a failover, the IP address might change and the cached value might no longer be in service.
- We recommend that you *do not* enable the following modes because they turn off transaction logging, which is required for Multi-AZ:
  - Simple recover mode
  - Offline mode
  - Read-only mode
- Test to determine how long it takes for your DB instance to failover. Failover time can vary due to the type of database, the instance class, and the storage type you use. You should also test your application's ability to continue working if a failover occurs.
- To shorten failover time, do the following:
  - Ensure that you have sufficient Provisioned IOPS allocated for your workload. Inadequate I/O can lengthen failover times. Database recovery requires I/O.
  - Use smaller transactions. Database recovery relies on transactions, so if you can break up large transactions into multiple smaller transactions, your failover time should be shorter.
- Take into consideration that during a failover, there will be elevated latencies. As part of the failover process, Amazon RDS automatically replicates your data to a new standby instance. This replication means that new data is being committed to two different DB instances. So there might be some latency until the standby DB instance has caught up to the new primary DB instance.
- Deploy your applications in all Availability Zones. If an Availability Zone does go down, your applications in the other Availability Zones will still be available.

When working with a Multi-AZ deployment of SQL Server, remember that Amazon RDS creates replicas for all SQL Server databases on your instance. If you don't want specific databases to have secondary replicas, set up a separate DB instance that doesn't use Multi-AZ for those databases.

Amazon RDS for SQL Server best practices video

The 2019 AWS re:Invent conference included a presentation on new features and best practices for working with SQL Server on Amazon RDS. A video of the presentation is available here.

Working with DB parameter groups

We recommend that you try out DB parameter group changes on a test DB instance before applying parameter group changes to your production DB instances. Improperly setting DB engine parameters in a DB parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying DB engine parameters and back up your DB instance before modifying a DB parameter group.

For information about backing up your DB instance, see Backing up, restoring, and exporting data.

Best practices for automating DB instance creation

It's an Amazon RDS best practice to create a DB instance with the preferred minor version of the database engine. You can use the AWS CLI, Amazon RDS API, or AWS CloudFormation to automate DB instance creation. When you use these methods, you can specify only the major version and Amazon RDS automatically creates the instance with the preferred minor version. For example, if PostgreSQL 12.5 is the preferred minor version, and if you specify version 12 with `create-db-instance`, the DB instance will be version 12.5.

To determine the preferred minor version, you can run the `describe-db-engine-versions` command with the `--default-only` option as shown in the following example.

```
aws rds describe-db-engine-versions --default-only --engine postgres

{
    "DBEngineVersions": [
        {
            "Engine": "postgres",
            "EngineVersion": "12.5",
            "DBParameterGroupFamily": "postgres12",
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 12.5-R1",
            ...some output truncated...
        }
    ]
}
```

For information on creating DB instances programmatically, see the following resources:

- Using the AWS CLI – create-db-instance

- Using the Amazon RDS API – CreateDBInstance
- Using AWS CloudFormation – AWS::RDS::DBInstance

Amazon RDS new features video

The 2023 AWS re:Invent conference included a presentation on new Amazon RDS features. A video of the presentation is available here.

Programmatic access to Amazon RDS

Amazon RDS provides you with the following tools to manage your Amazon RDS resources programmatically.

AWS Command Line Interface (AWS CLI)

You can create and manage your RDS resources by using the AWS CLI in a command-line shell. The AWS CLI provides direct access to the APIs for AWS services, such as Amazon RDS. For syntax and examples for the commands for Amazon RDS, see rds in the *AWS CLI Command Reference*.

AWS CloudFormation

With this AWS Infrastructure as Code (IaC) tool, you can create templates that describe all of the Amazon RDS resources that you want, and AWS CloudFormation provisions and configures those resources for you. For more information, see the section called “Creating resources with AWS CloudFormation”.

AWS software development kits (SDKs)

AWS provides SDKs for many popular technologies and programming languages. They make it easier for you to call AWS services from within your applications in that language or technology. For more information about these SDKs, see Tools for developing and managing applications on AWS.

Amazon RDS API

This API is the protocol-level interface for Amazon RDS. When using this API, you must format every HTTPS request correctly and add a valid digital signature to every request. For more information, see *Amazon RDS API reference*.

Console-to-Code

With this tool, you can generate code for actions that you perform in the Amazon RDS console, and use that code in other tools such as AWS CloudFormation. For more information, see the section called “Console-to-Code”.

Use Console-to-Code to generate code for your Amazon RDS console actions

The console provides a guided path for creating resources and testing prototypes. If you want to create the same resources at scale, you'll need automation code. Console-to-Code is a feature of Amazon Q Developer that can help you get started with your automation code. Console-to-Code records your console actions, including default values and parameters values that you provide. It then uses generative AI to suggest code in your preferred language and format for the actions that you choose. Because the console workflow makes sure the parameter values that you specify are valid together, the code that you generate by using Console-to-Code has compatible parameter values. You can use the code as a starting point, and then customize it to make it production-ready for your specific use case.

For example, with Console-to-Code, you can record creating an RDS DB instance and choose to generate code in AWS CloudFormation JSON format. Then, you can copy that code and customize it for use in your AWS CloudFormation template.

Console-to-Code can currently generate infrastructure-as-code (IaC) in the following languages and formats:

- CDK Java
- CDK Python
- CDK TypeScript
- CloudFormation JSON
- CloudFormation YAML

For more information and instructions on how to use Console-to-Code, see Automating AWS services with Amazon Q Developer Console-to-Code in the *Amazon Q Developer User Guide*.

Configuring an Amazon RDS DB instance

This section shows how to set up your Amazon RDS DB instance. Before creating a DB instance, decide on the DB instance class that will run the DB instance. Also, decide where the DB instance will run by choosing an AWS Region. Next, create the DB instance.

You can configure a DB instance with an option group and a DB parameter group.

- An *option group* specifies features, called options, that are available for a particular Amazon RDS DB instance.
- A *DB parameter group* acts as a container for engine configuration values that are applied to one or more DB instances.

The options and parameters that are available depend on the DB engine and DB engine version. You can specify an option group and a DB parameter group when you create a DB instance. You can also modify a DB instance to specify them.

Topics

- Creating an Amazon RDS DB instance
- Creating Amazon RDS resources with AWS CloudFormation
- Connecting to an Amazon RDS DB instance
- Working with option groups
- Parameter groups for Amazon RDS
- Creating an Amazon ElastiCache cache using Amazon RDS DB instance settings
- Auto migrating EC2 databases to Amazon RDS using AWS Database Migration Service
- Tutorial: Creating a MySQL DB instance with a custom parameterand new option group

Creating an Amazon RDS DB instance

The basic building block of Amazon RDS is the DB instance, where you create your databases. You choose the engine-specific characteristics of the DB instance when you create it. You also choose the storage capacity, CPU, memory, and so on of the AWS instance on which the database server runs.

Topics

- DB instance prerequisites
- Creating a DB instance
- Settings for DB instances

DB instance prerequisites

> **Important**
>
> Before you can create an Amazon RDS DB instance, complete the tasks in Setting up your Amazon RDS environment.

The following are prerequisites for creating an RDS DB instance.

Topics

- Configure the network for the DB instance
- Additional prerequisites

Configure the network for the DB instance

You can create an Amazon RDS DB instance only in a virtual private cloud (VPC) based on the Amazon VPC service. Also, it must be in an AWS Region that has at least two Availability Zones. The DB subnet group that you choose for the DB instance must cover at least two Availability Zones. This configuration ensures that you can configure a Multi-AZ deployment when you create the DB instance or easily move to one in the future.

To set up connectivity between your new DB instance and an Amazon EC2 instance in the same VPC, do so when you create the DB instance. To connect to your DB instance from resources other than EC2 instances in the same VPC, configure the network connections manually.

Topics

- Configure automatic network connectivity with an EC2 instance
- Configure the network manually

Configure automatic network connectivity with an EC2 instance

When you create an RDS DB instance, you can use the AWS Management Console to set up connectivity between an EC2 instance and the new DB instance. When you do so, RDS configures your VPC and network settings automatically. The DB instance is created in the same VPC as the EC2 instance so that the EC2 instance can access the DB instance.

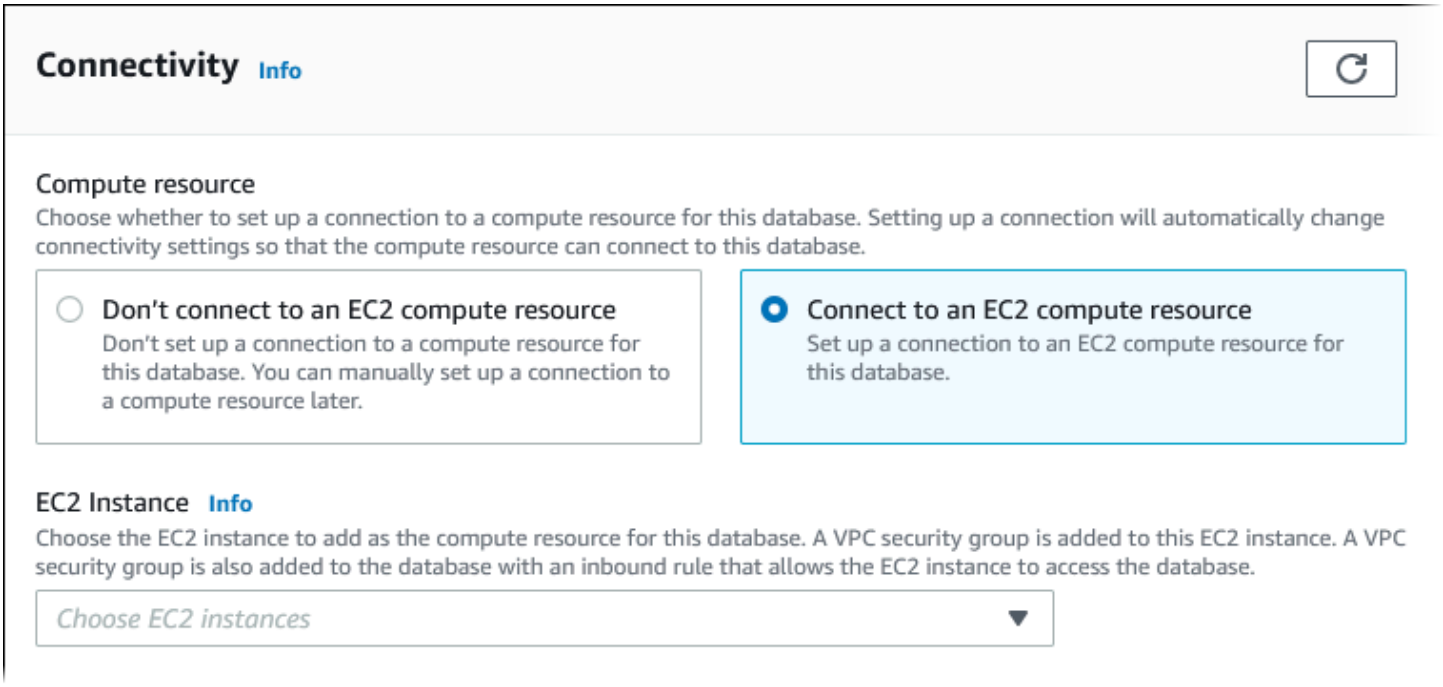
The following are requirements for connecting an EC2 instance with the DB instance:

- The EC2 instance must exist in the AWS Region before you create the DB instance.

  If no EC2 instances exist in the AWS Region, the console provides a link to create one.
- The user who is creating the DB instance must have permissions to perform the following operations:
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Using this option creates a private DB instance. The DB instance uses a DB subnet group with only private subnets to restrict access to resources within the VPC.

To connect an EC2 instance to the DB instance, choose **Connect to an EC2 compute resource** in the **Connectivity** section on the **Create database** page.



When you choose **Connect to an EC2 compute resource**, RDS sets the following options automatically. You can't change these settings unless you choose not to set up connectivity with an EC2 instance by choosing **Don't connect to an EC2 compute resource**.

Console option	Automatic setting
Network type	RDS sets network type to **IPv4**. Currently, dual-stack mode isn't supported when you set up a connection between an EC2 instance and the DB instance.
Virtual Private Cloud (VPC)	RDS sets the VPC to the one associated with the EC2 instance.
DB subnet group	RDS requires a DB subnet group with a private subnet in the same Availability Zone as the EC2 instance. If a DB subnet group that meets this requirement exists, then RDS uses the

Console option	Automatic setting
	existing DB subnet group. By default, this option is set to **Automatic setup**. When you choose **Automatic setup** and there is no DB subnet group that meets this requirement, the following action happens. RDS uses three available private subnets in three Availability Zones where one of the Availability Zones is the same as the EC2 instance. If a private subnet isn't available in an Availability Zone, RDS creates a private subnet in the Availability Zone. Then RDS creates the DB subnet group. When a private subnet is available, RDS uses the route table associated with the subnet and adds any subnets it creates to this route table. When no private subnet is available, RDS creates a route table without internet gateway access and adds the subnets it creates to the route table. RDS also allows you to use existing DB subnet groups. Select **Choose existing** if you want to use an existing DB subnet group of your choice.
Public access	RDS chooses **No** so that the DB instance isn't publicly accessible. For security, it is a best practice to keep the database private and make sure it isn't accessible from the internet.

Console option	Automatic setting
VPC security group (firewall)	RDS creates a new security group that is associated with the DB instance. The security group is named `rds-ec2-`*n*, where *n* is a number. This security group includes an inbound rule with the EC2 VPC security group (firewall) as the source. This security group that is associated with the DB instance allows the EC2 instance to access the DB instance. RDS also creates a new security group that is associated with the EC2 instance. The security group is named `ec2-rds-`*n*, where *n* is a number. This security group includes an outbound rule with the VPC security group of the DB instance as the source. This security group allows the EC2 instance to send traffic to the DB instance. You can add another new security group by choosing **Create new** and typing the name of the new security group. You can add existing security groups by choosing **Choose existing** and selecting security groups to add.
Availability Zone	When you choose **Single DB instance** in **Availability & durability** (Single-AZ deployment), RDS chooses the Availability Zone of the EC2 instance. When you choose **Multi-AZ DB instance** in **Availability & durability** (Multi-AZ DB instance deployment), RDS chooses the Availability Zone of the EC2 instance for one DB instance in the deployment. RDS randomly chooses a different Availability Zone for the other DB instance. Either the primary DB instance or the standby replica is created in the same Availability Zone as the EC2 instance. When you choose **Multi-AZ DB instance**, there is the possibility of cross Availability Zone costs if the DB instance and EC2 instance are in different Availability Zones.

For more information about these settings, see Settings for DB instances.

If you change these settings after the DB instance is created, the changes might affect the connection between the EC2 instance and the DB instance.

Configure the network manually

To connect to your DB instance from resources other than EC2 instances in the same VPC, configure the network connections manually. If you use the AWS Management Console to create your DB instance, you can have Amazon RDS automatically create a VPC for you. Or you can use an existing VPC or create a new VPC for your DB instance. With any approach, your VPC requires at least one subnet in each of at least two Availability Zones for use with an RDS DB instance.

By default, Amazon RDS creates the DB instance an Availability Zone automatically for you. To choose a specific Availability Zone, you need to change the **Availability & durability** setting to **Single DB instance**. Doing so exposes an **Availability Zone** setting that lets you choose from among the Availability Zones in your VPC. However, if you choose a Multi-AZ deployment, RDS chooses the Availability Zone of the primary or writer DB instance automatically, and the **Availability Zone** setting doesn't appear.

In some cases, you might not have a default VPC or haven't created a VPC. In these cases, you can have Amazon RDS automatically create a VPC for you when you create a DB instance using the console. Otherwise, do the following:

- Create a VPC with at least one subnet in each of at least two of the Availability Zones in the AWS Region where you want to deploy your DB instance. For more information, see Working with a DB instance in a VPC and Tutorial: Create a VPC for use with a DB instance (IPv4 only).
- Specify a VPC security group that authorizes connections to your DB instance. For more information, see Provide access to your DB instance in your VPC by creating a security group and Controlling access with security groups.
- Specify an RDS DB subnet group that defines at least two subnets in the VPC that can be used by the DB instance. For more information, see Working with DB subnet groups.

If you want to connect to a resource that isn't in the same VPC as the DB instance, see the appropriate scenarios in Scenarios for accessing a DB instance in a VPC.

Additional prerequisites

Before you create your DB instance, consider the following additional prerequisites:

- If you are connecting to AWS using AWS Identity and Access Management (IAM) credentials, your AWS account must have certain IAM policies. These grant the permissions required to perform Amazon RDS operations. For more information, see Identity and access management for Amazon RDS.

  To use IAM to access the RDS console, sign in to the AWS Management Console with your IAM user credentials. Then go to the Amazon RDS console at https://console.aws.amazon.com/rds/.

- To tailor the configuration parameters for your DB instance, specify a DB parameter group with the required parameter settings. For information about creating or modifying a DB parameter group, see Parameter groups for Amazon RDS.

  > **Important**
  >
  > If you are using the BYOL model for Amazon RDS for Db2, before creating a DB instance, you must first create a custom parameter group that contains your IBM Site ID and IBM Customer ID. For more information, see Bring your own license (BYOL) for Db2.

- Determine the TCP/IP port number to specify for your DB instance. The firewalls at some companies block connections to the default ports for RDS DB instances. If your company firewall blocks the default port, choose another port for your DB instance. The default ports for Amazon RDS DB engines are:

RDS for Db2	RDS for MariaDB	RDS for MySQL	RDS for Oracle	RDS for PostgreSQL	RDS for SQL Server
50000	3306	3306	1521	5432	1433

  For RDS for SQL Server, the following ports are reserved, and you can't use them when you create a DB instance: `1234`, `1434`, `3260`, `3343`, `3389`, `47001`, and `49152-49156`.

Creating a DB instance

You can create an Amazon RDS DB instance using the AWS Management Console, the AWS CLI, or the RDS API.

> **Note**
>
> For RDS for Db2, we recommend that you set up items needed for your license model before you create an RDS for Db2 DB instance. For more information, see Amazon RDS for Db2 licensing options.

Console

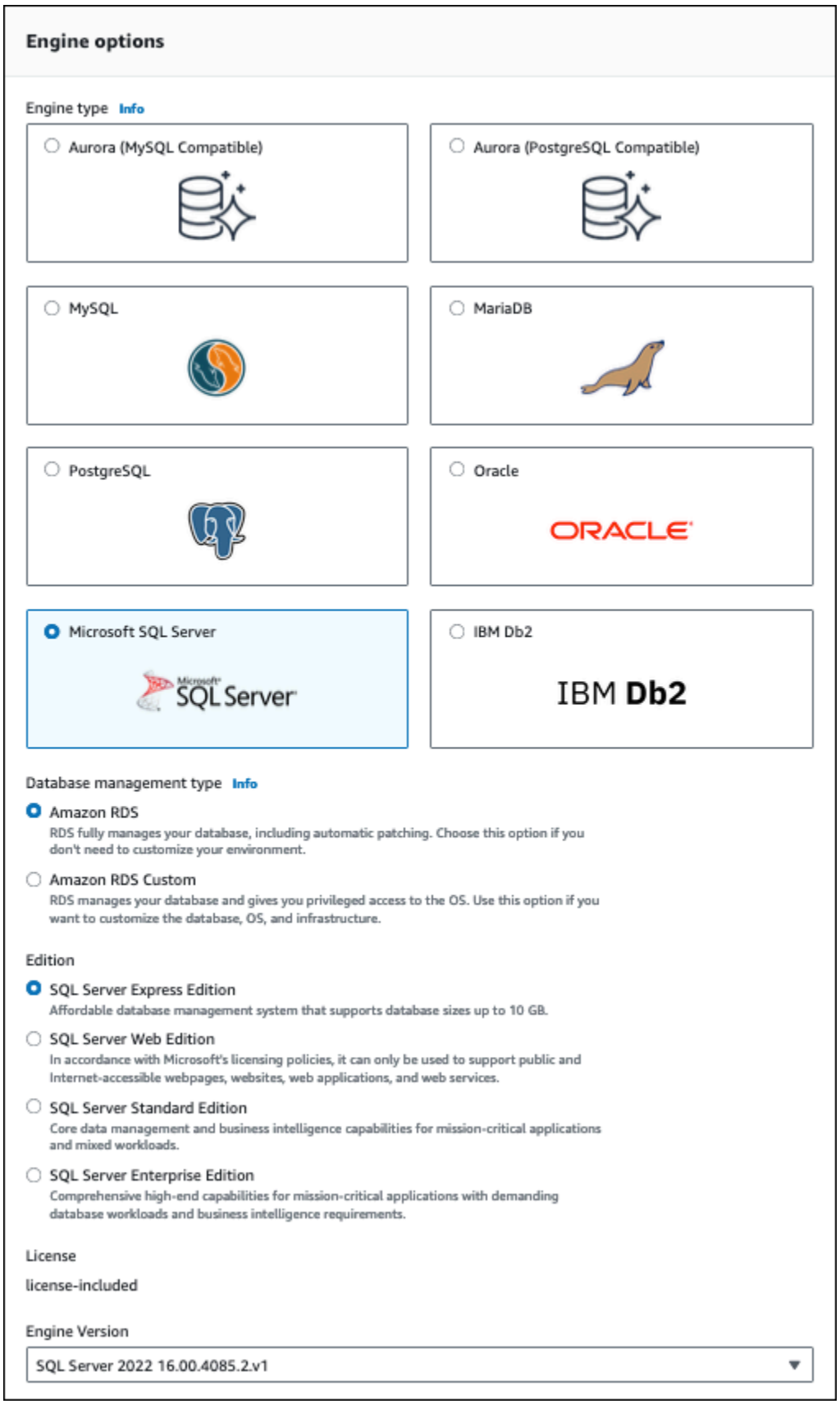
You can create a DB instance by using the AWS Management Console with **Easy create** enabled or not enabled. With **Easy create** enabled, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default setting for other configuration options. With **Easy create** not enabled, you specify more configuration options when you create a database, including ones for availability, security, backups, and maintenance.

> **Note**
>
> In the following procedure, **Standard create** is enabled, and **Easy create** isn't enabled. This procedure uses Microsoft SQL Server as an example.
> For examples that use **Easy create** to walk you through creating and connecting to sample DB instances for each engine, see Getting started with Amazon RDS.

To create a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**, then choose **Standard create**.
5. For **Engine type**, choose IBM Db2, MariaDB, Microsoft SQL Server, MySQL, Oracle, or PostgreSQL.

   Microsoft SQL Server is shown here.


Engine options
Engine type Info
Aurora (MySQL Compatible)
Aurora (PostgreSQL Compatible)
MySQL
MariaDB
PostgreSQL
Oracle
ORACLE
Microsoft SQL Server
Microsoft SQL Server
IBM Db2
IBM Db2
Database management type Info
Amazon RDS
RDS fully manages your database, including automatic patching. Choose this option if you don't need to customize your environment.
Amazon RDS Custom
RDS manages your database and gives you privileged access to the OS. Use this option if you want to customize the database, OS, and infrastructure.
Edition
SQL Server Express Edition
Affordable database management system that supports database sizes up to 10 GB.
SQL Server Web Edition
In accordance with Microsoft's licensing policies, it can only be used to support public and Internet-accessible webpages, websites, web applications, and web services.
SQL Server Standard Edition
Core data management and business intelligence capabilities for mission-critical applications and mixed workloads.
SQL Server Enterprise Edition
Comprehensive high-end capabilities for mission-critical applications with demanding database workloads and business intelligence requirements.
License
license-included
Engine Version
SQL Server 2022 16.00.4085.2.v1

6. For **Database management type**, if you're using Oracle or SQL Server choose **Amazon RDS** or **Amazon RDS Custom**.

   Amazon RDS is shown here. For more information on RDS Custom, see *Amazon RDS Custom*.

7. For **Edition**, if you're using Db2, Oracle, or SQL Server, choose the DB engine edition that you want to use.

   MySQL has only one option for the edition, and MariaDB and PostgreSQL have none.

8. For **Version**, choose the engine version.

9. In **Templates**, choose the template that matches your use case. If you choose **Production**, the following are preselected in a later step:

   - **Multi-AZ** failover option
   - **Provisioned IOPS SSD (io1)** storage option
   - **Enable deletion protection** option

   We recommend these features for any production environment.

   > **Note**
   >
   > Template choices vary by edition.

10. In the **Settings** section, open **Credential Settings**. Then do the following:

    a. (Optional) Change the **Master username** value.

    b. Choose either of the following credentials management options:

       - **Managed in AWS Secrets Manager**

         In **Select the encryption key**, choose either a KMS key that Secrets Manager creates or a key that you have created.

         > **Note**
         >
         > We recommend AWS Secrets Manager as the most secure technique for managing credentials. Additional charges apply. For more information, see Password management with Amazon RDS and AWS Secrets Manager.

- **Self managed**

  To specify a password, clear the **Auto generate a password** check box if it is selected. Enter the same password in **Master password** and **Confirm master password**.

11. (Optional) Set up a connection to a compute resource for this DB instance.

    You can configure connectivity between an Amazon EC2 instance and the new DB instance during DB instance creation. For more information, see Configure automatic network connectivity with an EC2 instance.

12. In the **Connectivity** section under **VPC security group (firewall)**, if you select **Create new**, a VPC security group is created with an inbound rule that allows your local computer's IP address to access the database.

13. For the remaining sections, specify your DB instance settings. For information about each setting, see Settings for DB instances.

14. Choose **Create database**.

    If you chose to use an automatically generated password, the **View credential details** button appears on the **Databases** page.

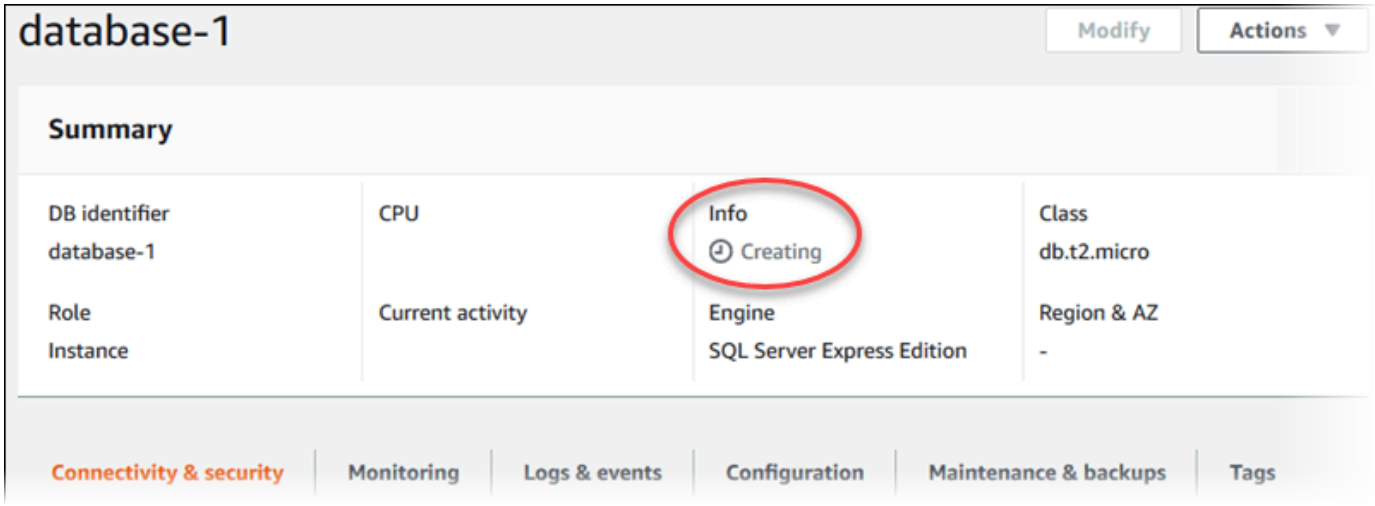
    To view the master username and password for the DB instance, choose **View credential details**.

    To connect to the DB instance as the master user, use the username and password that appear.

    > **Important**
    >
    > You can't view the master user password again. If you don't record it, you might have to change it. If you need to change the master user password after the DB instance is available, modify the DB instance to do so. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

15. For **Databases**, choose the name of the new DB instance.

    On the RDS console, the details for the new DB instance appear. The DB instance has a status of **Creating** until the DB instance is created and ready for use. When the state changes to **Available**, you can connect to the DB instance. Depending on the DB instance class and storage allocated, it can take several minutes for the new instance to be available.



AWS CLI

> **Note**
>
> If you want to use Db2 license through AWS Marketplace, you must first subscribe to AWS Marketplace and register with IBM by using the AWS Management Console. For more information, see Subscribing to Db2 Marketplace listings and registering with IBM.

To create a DB instance by using the AWS CLI, call the create-db-instance command with the following parameters:

- `--db-instance-identifier`
- `--db-instance-class`
- `--vpc-security-group-ids`
- `--db-subnet-group`
- `--engine`
- `--master-username`
- `--master-user-password` or `--manage-master-user-password`
- `--allocated-storage`
- `--backup-retention-period`

For information about each setting, see Settings for DB instances.

This example uses Microsoft SQL Server.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --engine sqlserver-se \
    --db-instance-identifier mymsftsqlserver \
    --allocated-storage 250 \
    --db-instance-class db.t3.large \
    --vpc-security-group-ids mysecuritygroup \
    --db-subnet-group mydbsubnetgroup \
    --master-username masterawsuser \
    --manage-master-user-password \
    --backup-retention-period 3
```

For Windows:

```
aws rds create-db-instance ^
    --engine sqlserver-se ^
    --db-instance-identifier mydbinstance ^
    --allocated-storage 250 ^
    --db-instance-class db.t3.large ^
    --vpc-security-group-ids mysecuritygroup ^
    --db-subnet-group mydbsubnetgroup ^
    --master-username masterawsuser ^
    --manage-master-user-password ^
    --backup-retention-period 3
```

This command produces output similar to the following.

```
DBINSTANCE  mydbinstance  db.t3.large  sqlserver-se  250  sa  creating  3  ****  n
 10.50.2789
SECGROUP  default  active
PARAMGRP  default.sqlserver-se-14  in-sync
```

RDS API

> **Note**
>
> If you want to use Db2 license through AWS Marketplace, you must first subscribe to AWS Marketplace and register with IBM by using the AWS Management Console. For more information, see Subscribing to Db2 Marketplace listings and registering with IBM.

To create a DB instance by using the Amazon RDS API, call the CreateDBInstance operation.

For information about each setting, see Settings for DB instances.

Settings for DB instances

In the following table, you can find details about settings that you choose when you create a DB instance. The table also shows the DB engines for which each setting is supported.

You can create a DB instance using the console, the create-db-instance CLI command, or the CreateDBInstance RDS API operation.

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Allocated storage	The amount of storage to allocate for your DB instance (in gibibytes). In some cases, allocating a higher amount of storage for your DB instance than the size of your database can improve I/O performance.
	
	For more information, see Amazon RDS DB instance storage.
		CLI option:
		
		`--allocated-storage`
		
		API parameter:
		
		`AllocatedStorage`
			All

Architecture settings	If you choose **Oracle multitenant architecture**, RDS for Oracle creates a container database (CDB). If you don't choose this option, RDS for Oracle	**CLI option:**	Oracle

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines
	creates a non-CDB. A non-CDB uses the traditional Oracle database architecture. A CDB can contain pluggable databases (PDBs) whereas a non-CDB cannot. Oracle Database 21c uses the CDB architecture only. Oracle Database 19c can use either the CDB or non-CDB architecture. Releases lower than Oracle Database 19c use the non-CDB architecture only. For more information, see Overview of RDS for Oracle CDBs.	`--engine oracle-ee-cdb` (Oracle multitenant) `--engine oracle-se2-cdb` (Oracle multitenant) `--engine oracle-ee` (traditional) `--engine oracle-se2` (traditional) **API parameter:** `Engine`	

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines
Architecture configuration	These settings are only valid when you choose **Oracle multitenant architecture** for **Architecture settings**. Choose either of the following additional settings: • With the **Multi-tenant configuration**, your RDS for Oracle CDB instance can contain 1–30 tenant databases, depending on the database edition and any required option licenses. In the context of an Oracle database, a tenant database is a PDB. Application PDBs and proxy PDBs aren't supported. Your DB instance is created with 1 initial tenant database. Choose values for **Tenant database name**, **Tenant database master username**, **Tenant database master password**, and **Tenant database character set**. The multi-tenant configuration is permanent. Thus, you can't convert the multi-tenant configuration back to the single-tenant configuration. The minimum supported release update (RU) for the multi-tenant configuration is 19.0.0.0.ru-2022-01.rur-2022.r1.	**CLI option:** `--multi-tenant` (multi-tenant configuration) `--no-multi-tenant` (single-tenant configuration) **API parameter:** `MultiTenant`	Oracle

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines
	Note The Amazon RDS configuration is called "multi-tenant" rather than "multitenant" because it is a capability of Amazon RDS, not just the Oracle DB engine. Similarly, the RDS term "tenant" refers to any tenant in an RDS configuration, not just Oracle PDBs. In the RDS documentation, the unhyphenated term "Oracle multitenant" refers exclusively to the Oracle database CDB architecture, which is compatible with both on-premises and RDS deployments. • With the **Single-tenant configuration**, your RDS for Oracle CDB contains 1 PDB. This is the default configuration when you create a CDB. You can't delete the initial PDB or add more PDBs. You can later convert the single-tenant configuration of your CDB to the multi-tenant conf iguration, but you can't then convert		

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines
	back to the single-tenant configuration. Regardless of which configuration you choose, your CDB contains a single initial PDB. In the multi-tenant configuration, you can create more PDBs later using RDS APIs. For more information, see Overview of RDS for Oracle CDBs.		

Auto minor version upgrade	Choose **Enable auto minor version upgrade** to enable your DB instance to receive preferred minor DB engine version upgrades automatically when they become available. This is the default behavior. Amazon RDS performs automatic minor version upgrades in the maintenance window. If you don't choose **Enable auto minor version upgrade**, your DB instance isn't upgraded automatically when new minor versions become available.
	
	For more information, see Automatically upgrading the minor engine version.
		CLI option:
		
		`--auto-minor-version-upgrade`
		
		`--no-auto-minor-version-upgrade`
		
		API parameter:
		
		`AutoMinorVersionUpgrade`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Availability zone	The Availability Zone for your DB instance. Use the default value of **No Preference** unless you want to specify an Availability Zone.
	
	For more information, see Regions, Availability Zones, and Local Zones.
		CLI option:
		
		`--availability-zone`
		
		API parameter:
		
		`AvailabilityZone`
			All

AWS KMS key	Only available if **Encryption** is set to **Enable encryption**. Choose the AWS KMS key to use for encrypting this DB instance. For more information, see Encrypting Amazon RDS resources.	**CLI option:**
		
		`--kms-key-id`
		
		API parameter:
		
		`KmsKeyId`
			All

AWS License Manager configuration	Enter a name for an AWS License Manager license configuration. The name must be 100 characters or less, and only include a-z, A-Z, and 0-9. For more information, see the section called "Integrating with AWS License Manager".	**CLI option:** For more information, see AWS License Manager CLI. **API parameter:** For more information, see AWS License Manager API.	Db2

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines
Backup replication	Choose **Enable replication in another AWS Region** to create backups in an additional Region for disaster recovery. Then choose the **Destination Region** for the additional backups.	Not available when creating a DB instance. For information on enabling cross-Regi on backups using the AWS CLI or RDS API, see Enabling cross-Region automated backups for Amazon RDS.	Db2 Oracle PostgreSQL SQL Server

Backup retention period	The number of days that you want automatic backups of your DB instanc e to be retained. For any nontrivial DB instance, set this value to **1** or greater.
	
	For more information, see Introduction to backups.
		CLI option:
		
		`--backup-retention-period`
		
		API parameter:
		
		`BackupRetentionPeriod`
			All

Backup target	Choose **AWS Cloud** to store automated backups and manual snapshots in the parent AWS Region. Choose **Outposts (on-premises)** to store them locally on your Outpost.
	
	This option setting applies only to RDS on Outposts. For more information, see Creating DB instances for Amazon RDS on AWS Outposts.
		CLI option:
		
		`--backup-target`
		
		API parameter:
		
		`BackupTarget`
			MySQL, PostgreSQL, SQL Server

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Backup window	The time period during which Amazon RDS automatically takes a backup of your DB instance. Unless you have a specific time that you want to have your database backed up, use the default of **No Preference**.
	
	For more information, see Introduction to backups.
		CLI option:
		
		`--preferred-backup-window`
		
		API parameter:
		
		`PreferredBackupWindow`
			All

Certificate authority	The certificate authority (CA) for the server certificate used by the DB instance.
	
	For more information, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
		CLI option:
		
		`--ca-certificate-identifier`
		
		RDS API parameter:
		
		`CACertificateIdentifier`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Character set	The character set for your DB instance. The default value of **AL32UTF8** for the DB character set is for the Unicode 5.0 UTF-8 Universal character set. You can't change the DB character set after you create the DB instance.
	
	In a single-tenant configuration, a non-default DB character set affects only the PDB, not the CDB. For more information, see Single-tenant configuration of the CDB architecture.
	
	The DB character set is different from the national character set, which is called the NCHAR character set. Unlike the DB character set, the NCHAR character set specifies the encoding for NCHAR data types (NCHAR, NVARCHAR2, and NCLOB) columns without affecting database metadata.
	
	For more information, see RDS for Oracle character sets.
		CLI option:
		
		`--character-set-name`
		
		API parameter:
		
		`CharacterSetName`
			Oracle

Collation	A server-level collation for your DB instance.
	
	For more information, see Server-level collation for Microsoft SQL Server.
		CLI option:
		
		`--character-set-name`
		
		API parameter:
		
		`CharacterSetName`
			SQL Server

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Copy tags to snapshots	This option copies any DB instance tags to a DB snapshot when you create a snapshot.
	
	For more information, see Tagging Amazon RDS resources.
		CLI option:
		
		`--copy-tags-to-snapshot`
		
		`--no-copy-tags-to-snapshot`
		
		RDS API parameter:
		
		`CopyTagsToSnapshot`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Database authentication	The database authentication option that you want to use.
	
	Choose **Password authentication** to authenticate database users with database passwords only.
	
	Choose **Password and IAM DB authentication** to authenticate database users with database passwords and user credentials through users and roles. For more information, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL. This option is only supported for MySQL and PostgreSQL.
	
	Choose **Password and Kerberos authentication** to authenticate database users with database passwords and Kerberos authentication through an AWS Managed Microsoft AD created with Directory Service. Next, choose the directory or choose **Create a new Directory**.
	
	For more information, see one of the following:
	
	• Using Kerberos authentication for Amazon RDS for Db2
	• Using Kerberos authentication for Amazon RDS for MySQL
		IAM:
		
		CLI option:
		
		`--enable-iam-database-authentication`
		
		`--no-enable-iam-database-authentication`
		
		RDS API parameter:
		
		`EnableIAMDatabaseAuthentication`
		
		Kerberos:
		
		CLI option:
		
		`--domain`
		
		`--domain-iam-role-name`
		
		RDS API parameter:
		
		`Domain`
		
		`DomainIAMRoleName`
			Varies by authentication type

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines
	• Configuring Kerberos authentication for Amazon RDS for Oracle • Using Kerberos authentication with Amazon RDS for PostgreSQL		
Database management type	Choose **Amazon RDS** if you don't need to customize your environment. Choose **Amazon RDS Custom** if you want to customize the database, OS, and infrastructure. For more information, see Amazon RDS Custom.	For the CLI and API, you specify the database engine type.	Oracle SQL Server

Database port	The port that you want to access the DB instance through. The default port is shown.
	
	Note
	The firewalls at some companies block connections to the default MariaDB, MySQL, and PostgreSQL ports. If your company firewall blocks the default port, enter another port for your DB instance.
		CLI option:
		
		`--port`
		
		RDS API parameter:
		
		`Port`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

DB engine version	The version of database engine that you want to use.	**CLI option:**
		`--engine-version`
		RDS API parameter:
		`EngineVersion`
			All

DB instance class	The configuration for your DB instance. For example, a **db.t3.small** DB instance class has 2 GiB memory, 2 vCPUs, 1 virtual core, a variable ECU, and a moderate I/O capacity.
	If possible, choose a DB instance class large enough that a typical query working set can be held in memory. When working sets are held in memory, the system can avoid writing to disk, which improves performance. For more information, see DB instance classes.
	In RDS for Oracle, you can select **Include additional memory configurations**. These configurations are optimized for a high ratio of memory to vCPU. For example, **db.r5.6xlarge.tpc2.mem4x** is a db.r5.8x DB instance that has 2 threads per core (tpc2) and 4x the memory of a standard db.r5.6xlarge DB instance. For more information, see RDS for Oracle DB instance classes.
		CLI option:
		`--db-instance-class`
		RDS API parameter:
		`DBInstanceClass`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

DB instance identifier	The name for your DB instance. Name your DB instances in the same way that you name your on-premises servers. Your DB instance identifier can contain up to 63 alphanumeric characters, and must be unique for your account in the AWS Region you chose.	**CLI option:**
		`--db-instance-identifier`
		RDS API parameter:
		`DBInstanceIdentifier`
			All

DB parameter group	A parameter group for your DB instance. You can choose the default parameter group, or you can create a custom parameter group.
	If you are using the BYOL model for RDS for Db2, before creating a DB instance, you must first create a custom parameter group that contains your IBM Site ID and IBM Customer ID. For more information, see Bring your own license (BYOL) for Db2.
	For more information, see Parameter groups for Amazon RDS.
		CLI option:
		`--db-parameter-group-name`
		RDS API parameter:
		`DBParameterGroupName`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

DB subnet group	The DB subnet group you want to use for the DB cluster.
	Select **Choose existing** to use an existing DB subnet group. Then choose the required subnet group from the **Existing DB subnet groups** dropdown list.
	
	Choose **Automatic setup** to let RDS select a compatible DB subnet group. If none exist, RDS creates a new subnet group for your cluster.
	
	For more information, see Working with DB subnet groups.
		CLI option:
		
		`--db-subnet-group-name`
		
		RDS API parameter:
		
		`DBSubnetGroupName`
			All

Dedicated Log Volume	Use a dedicated log volume (DLV) to store database transaction logs on a storage volume that's separate from the volume containing the database tables.
	
	For more information, see Using a dedicated log volume (DLV).
		CLI option:
		
		`--dedicated-log-volume`
		
		RDS API parameter:
		
		`DedicatedLogVolume`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Deletion protection	**Enable deletion protection** to prevent your DB instance from being deleted. If you create a production DB instance with the AWS Management Console, deletion protection is enabled by default.
	
	For more information, see Deleting a DB instance.
		CLI option:
		
		`--deletion-protection`
		
		`--no-deletion-protection`
		
		RDS API parameter:
		
		`DeletionProtection`
			All

Encryption	**Enable Encryption** to enable encryption at rest for this DB instance.
	
	For more information, see Encrypting Amazon RDS resources.
		CLI option:
		
		`--storage-encrypted`
		
		`--no-storage-encrypted`
		
		RDS API parameter:
		
		`StorageEncrypted`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Enhanced Monitoring	**Enable enhanced monitoring** to enable gathering metrics in real time for the operating system that your DB instance runs on.
	
	For more information, see Monitoring OS metrics with Enhanced Monitoring.
		CLI options:
		
		`--monitoring-interval`
		
		`--monitoring-role-arn`
		
		RDS API parameters:
		
		`MonitoringInterval`
		
		`MonitoringRoleArn`
			All

Engine type	Choose the database engine to be used for this DB instance.	**CLI option:**
		
		`--engine`
		
		RDS API parameter:
		
		`Engine`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Initial database name	The name for the database on your DB instance. If you don't provide a name, Amazon RDS doesn't create a database on the DB instance (except for Oracle and PostgreSQL). The name can't be a word reserved by the database engine, and has other constraints depending on the DB engine.
	
	Db2:
	
	• It must contain 1–8 alphanumeric characters.
	• It must start with a-z, A-Z, @, $, or #, and be followed by a-z, A-Z, 0-9, _, @, #, or $.
	• It can't contain spaces.
	• For more information, see Additional considerations.
	
	MariaDB and MySQL:
	
	• It must contain 1–64 alphanumeric characters.
	
	Oracle:
	
	•
		CLI option:
		
		`--db-name`
		
		RDS API parameter:
		
		`DBName`
			All except SQL Server

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines
	It must contain 1–8 alphanumeric characters. • It can't be NULL. The default value is ORCL. • It must begin with a letter. PostgreSQL: • It must contain 1–63 alphanumeric characters. • It must begin with a letter or an underscore. Subsequent characters can be letters, underscores, or digits (0-9). • The initial database name is postgres.		

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

License	Valid values for the license model:
	• **bring-your-own-license** or **marketplace-license** for Db2.
	• **general-public-license** for MariaDB.
	• **license-included** for Microsoft SQL Server.
	• **general-public-license** for MySQL.
	• **license-included** or **bring-your-own-license** for Oracle.
	• **postgresql-license** for PostgreSQL.
		CLI option:
		`--license-model`
		RDS API parameter:
		`LicenseModel`
			All

Log exports	The types of database log files to publish to Amazon CloudWatch Logs.
	For more information, see Publishing database logs to Amazon CloudWatch Logs.
		CLI option:
		`--enable-cloudwatch-logs-exports`
		RDS API parameter:
		`EnableCloudwatchLogsExports`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Maintenance window	The 30-minute window in which pending modifications to your DB instance are applied. If the time period doesn't matter, choose **No Preference**.
	
	For more information, see Amazon RDS maintenance window.
		CLI option:
		
		`--preferred-maintenance-window`
		
		RDS API parameter:
		
		`PreferredMaintenanceWindow`
			All

Manage master credentials in AWS Secrets Manager	Select **Manage master credentials in AWS Secrets Manager** to manage the master user password in a secret in Secrets Manager.
	
	Optionally, choose a KMS key to use to protect the secret. Choose from the KMS keys in your account, or enter the key from a different account.
	
	For more information, see Password management with Amazon RDS and AWS Secrets Manager.
		CLI option:
		
		`--manage-master-user-password | --no-manage-master-user-password`
		
		`--master-user-secret-kms-key-id`
		
		RDS API parameter:
		
		`ManageMasterUserPassword`
		
		`MasterUserSecretKmsKeyId`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Master password	The password for your master user account. The password has the following number of printable ASCII characters (excluding /, ", a space, and @) depending on the DB engine:
	• Db2: 8–255
	• Oracle: 8–30
	• MariaDB and MySQL: 8–41
	• SQL Server and PostgreSQL: 8–128
		CLI option:
		`--master-user-password`
		RDS API parameter:
		`MasterUserPassword`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Master username	The name that you use as the master username to log in to your DB instance with all database privileges. Note the following naming restrictions:
	• The name can contain 1–16 alphanumeric characters and underscores.
	• The first character must be a letter.
	• The name can't be a word reserved by the database engine.
	
	You can't change the master username after you create the DB instance.
	
	For Db2, we recommend that you use the same master username as your self-managed Db2 instance name.
	
	For more information on privileges granted to the master user, see Master user account privileges.
		CLI option:
		`--master-username`
		RDS API parameter:
		`MasterUsername`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Microsoft SQL Server Windows Aut hentication	**Enable Microsoft SQL Server Windows authentication**, then **Browse Directory** to choose the directory where you want to allow authorized domain users to authenticate with this SQL Server ins tance using Windows Authentication.	**CLI options:**
		`--domain`
		`--domain-iam-role-name`
		RDS API parameters:
		`Domain`
		`DomainIAMRoleName`
			SQL Server

Multi-AZ deploymen t	**Create a standby instance** to create a passive secondary replica of your DB instance in another Availability Zone for failover support. We recommend Multi-AZ for production workloads to maintai n high availability.
	For development and testing, you can choose **Do not create a standby instance**.
	For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.
		CLI option:
		`--multi-az`
		`--no-multi-az`
		RDS API parameter:
		`MultiAZ`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

National character set (NCHAR)	The national character set for your DB instance, commonly called the NCHAR character set. You can set the national character set to either AL16UTF16 (default) or UTF-8. You can't change the national character set after you create the DB instance.
	
	The national character set is different from the DB character set. Unlike the DB character set, the national character set specifies the encoding only for NCHAR data types (NCHAR, NVARCHAR2, and NCLOB) columns without affecting database metadata.
	
	For more information, see RDS for Oracle character sets.
		CLI option:
		
		`--nchar-character-set-name`
		
		API parameter:
		
		`NcharCharacterSetName`
			Oracle

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Network type	The IP addressing protocols supported by the DB instance.
	
	IPv4 (the default) to specify that resources can communicate with the DB instance only over the Internet Protocol version 4 (IPv4) addressing protocol.
	
	Dual-stack mode to specify that resources can communicate with the DB instance over IPv4, Internet Protocol version 6 (IPv6), or both. Use dual-stack mode if you have any resources that must communicate with your DB instance over the IPv6 addressing protocol. Also, make sure that you associate an IPv6 CIDR block with all subnets in the DB subnet group that you specify.
	
	For more information, see Amazon RDS IP addressing.
		CLI option:
		
		`--network-type`
		
		RDS API parameter:
		
		`NetworkType`
			All

Option group	An option group for your DB instance. You can choose the default option group or you can create a custom option group.
	
	For more information, see Working with option groups.
		CLI option:
		
		`--option-group-name`
		
		RDS API parameter:
		
		`OptionGroupName`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Performance Insights	**Enable Performance Insights** to monitor your DB instance load so that you can analyze and troubleshoot your database performance.
	
	Choose a retention period to determine how much Performance Insights data history to keep. The retention setting is **Default (7 days)**. To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see Pricing and data retention for Performance Insights.
	
	Choose a KMS key to use to protect the key used to encrypt this database volume. Choose from the KMS keys in your account, or enter the key from a different account.
	
	For more information, see Monitoring DB load with Performance Insights on Amazon RDS.
		CLI options:
		
		`--enable-performance-insights`
		
		`--no-enable-performance-insights`
		
		`--performance-insights-retention-period`
		
		`--performance-insights-kms-key-id`
		
		RDS API parameters:
		
		`EnablePerformanceInsights`
		
		`PerformanceInsightsRetentionPeriod`
		
		`PerformanceInsightsKMSKeyId`
			All except Db2

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Provisioned IOPS	The Provisioned IOPS (I/O operations per second) value for the DB instance. This setting is available only if you choose one of the following for **Storage type**:
	• **General purpose SSD (gp3)**
	• **Provisioned IOPS SSD (io1)**
	• **Provisioned IOPS SSD (io2)**
	
	For more information, see Amazon RDS DB instance storage.
		CLI option:
		`--iops`
		RDS API parameter:
		`Iops`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Public access	**Yes** to give the DB instance a public IP address, meaning that it's accessible outside the VPC. To be publicly accessible, the DB instance also has to be in a public subnet in the VPC.
	
	No to make the DB instance accessible only from inside the VPC.
	
	For more information, see Hiding a DB instance in a VPC from the internet.
	
	To connect to a DB instance from outside of its VPC, the DB instance must be publicly accessible. Also, access must be granted using the inbound rules of the DB instance's security group. In addition, other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance.
	
	If your DB instance isn't publicly accessible, use an AWS Site-to-Site VPN connection or an Direct Connect connection to access it from a private network. For more information, see Internetwork traffic privacy.
		CLI option:
		
		`--publicly-accessible`
		
		`--no-publicly-accessible`
		
		RDS API parameter:
		
		`PubliclyAccessible`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

RDS Extended Support	Select **Enable RDS Extended Support** to allow supported major engine versions to continue running past the RDS end of standard support date.
	
	When you create a DB instance, Amazon RDS defaults to RDS Extended Support. To prevent the creation of a new DB instance after the RDS end of standard support date and to avoid charges for RDS Extended Support, disable this setting. Your existing DB instances won't incur charges until the RDS Extended Support pricing start date.
	
	For more information, see Amazon RDS Extended Support with Amazon RDS.
		CLI option:
		
		`--engine-lifecycle-support`
		
		RDS API parameter:
		
		`EngineLifecycleSupport`
			MySQL
			
			PostgreSQL

RDS Proxy	Choose **Create an RDS Proxy** to create a proxy for your DB instance. Amazon RDS automatically creates an IAM role and a Secrets Manager secret for the proxy. For more information, see Amazon RDS Proxy.	Not available when creating a DB instance.	MariaDB MySQL PostgreSQL

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Storage autoscaling	**Enable storage autoscaling** to enable Amazon RDS to automatically increase storage when needed to avoid having your DB instance run out of storage space.
	
	Use **Maximum storage threshold** to set the upper limit for Amazon RDS to automatically increase storage for your DB instance. The default is 1,000 GiB.
	
	For more information, see Managing capacity automatically with Amazon RDS storage autoscaling.
		CLI option:
		
		`--max-allocated-storage`
		
		RDS API parameter:
		
		`MaxAllocatedStorage`
			All

Storage throughput	The storage throughput value for the DB instance. This setting is available only if you choose **General purpose SSD (gp3)** for **Storage type**.
	
	For more information, see gp3 storage (recommended).
		CLI option:
		
		`--storage-throughput`
		
		RDS API parameter:
		
		`StorageThroughput`
			All

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Storage type	The storage type for your DB instance.
	
	If you choose **General Purpose SSD (gp3)**, you can provision additional provisioned IOPS and storage throughput under **Advanced settings**.
	
	If you choose **Provisioned IOPS SSD (io1)** or **Provisioned IOPS SSD (io2)**, enter the **Provisioned IOPS** value.
	
	For more information, see Amazon RDS storage types.
		CLI option:
		
		`--storage-type`
		
		RDS API parameter:
		
		`StorageType`
			All

Additional storage volumes	You can add up to three additional storage volumes to your RDS for Oracle or RDS for SQL Server DB instance. You can configure each additional volume to use gp3 or io2 storage types. You can also specify different allocated storage, IOPS, and throughput settings to optimize for your workload requirements.
	
	Make sure you specify the volume names as `rdsdbdata2` , `rdsdbdata3` , or `rdsdbdata4` .
	
	The combined storage across the primary and additional volumes cannot exceed 256 TiB.
		CLI option:
		
		`--additional-storage-volumes`
		
		RDS API parameter:
		
		`AdditionalStorageVolumes`
			Oracle and SQL Server

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Subnet group	A DB subnet group to associate with this DB instance.
	
	For more information, see Working with DB subnet groups.
		CLI option:
		
		`--db-subnet-group-name`
		
		RDS API parameter:
		
		`DBSubnetGroupName`
			All

Tenant database name	The name of your initial PDB in the multi-tenant configuration of the Oracle architecture. This setting is available only if you choose **Multi-tenant configuration** for **Architecture configuration**.
	
	The tenant database name must differ from the name of your CDB, which is named RDSCDB. You can't change the CDB name.
		CLI option:
		
		`--db-name`
		
		RDS API parameter:
		
		`DBName`
			Oracle

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Tenant database master username	The name that you use as the master username to log in to your tenant database (PDB) with all database privileges. This setting is available only if you choose **Multi-tenant configuration** for **Architecture configuration**.
	
	Note the following naming restrictions:
	
	• The name can contain 1–16 alphanumeric characters and underscores.
	• The first character must be a letter.
	• The name can't be a word reserved by the database engine.
	
	You can't do the following:
	
	• Change the tenant master username after you create the tenant database.
	• Log in with the tenant master username to the CDB.
		CLI option:
		
		`--master-username`
		
		RDS API parameter:
		
		`MasterUsername`
			Oracle

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Tenant database master password	The password for the master user account of your tenant database (PDB). This setting is available only if you choose **Multi-tenant configuration** for **Architecture configuration**.
	
	The password has 8–30 printable ASCII characters, excluding /, ", a space, and @.
		CLI option:
		
		`--master-password`
		
		RDS API parameter:
		
		`MasterPassword`
			Oracle

Tenant database character set	The character set of the initial tenant database. This setting is available only if you choose **Multi-tenant configuration** for **Architecture configuration**. Only RDS for Oracle CDB instances are supported.
	
	The default value of **AL32UTF8** for the tenant database character set is for the Unicode 5.0 UTF-8 Universal character set. You can choose a tenant database character set that is different from the character set of the CDB.
	
	For more information, see RDS for Oracle character sets.
		CLI option:
		
		`--character-set-name`
		
		RDS API parameter:
		
		`CharacterSetName`
			Oracle

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines

Tenant database national character set	The national character set for your tenant database, commonly called the NCHAR character set. This setting is available only if you choose **Multi-tenant configuration** for **Architecture configuration**. Only RDS for Oracle CDB instances are supported.
	
	You can set the national character set to either **AL16UTF16** (default) or **UTF-8**. You can't change the national character set after you create the tenant database.
	
	The tenant database national character set is different from the tenant database character set. The national character set specifies the encoding only for columns that use the NCHAR data type (NCHAR, NVARCHAR2, and NCLOB) and doesn't affect database metadata.
	
	For more information, see RDS for Oracle character sets.
		CLI option:
		
		`--nchar-character-set-name`
		
		API parameter:
		
		`NcharCharacterSetName`
			Oracle

Time zone	The time zone for your DB instance. If you don't choose a time zone, your DB instance uses the default time zone. You can't change the time zone after the DB instance is created.
	
	For more information, see Local time zone for Amazon RDS for Db2 DB instances and Local time zone for Microsoft SQL Server DB instances.
		CLI option:
		
		`--timezone`
		
		RDS API parameter:
		
		`Timezone`
			Db2
			
			SQL Server
			
			RDS Custom for SQL Server

Console setting	Setting description	CLI option and RDS API parameter	Supported DB engines
Virtual Private Cloud (VPC)	A VPC based on the Amazon VPC service to associate with this DB instance. For more information, see Amazon VPC and Amazon RDS.	For the CLI and API, you specify the VPC security group IDs.	All

VPC security group (firewall)	The security group to associate with the DB instance.
	
	For more information, see Overview of VPC security groups.
		CLI option:
		
		`--vpc-security-group-ids`
		
		RDS API parameter:
		
		`VpcSecurityGroupIds`
			All

Creating Amazon RDS resources with AWS CloudFormation

Amazon RDS is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want (such as DB instances and DB parameter groups), and CloudFormation provisions and configures those resources for you.

When you use CloudFormation, you can reuse your template to set up your RDS resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

RDS and CloudFormation templates

CloudFormation templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use CloudFormation Designer to help you get started with CloudFormation templates. For more information, see What is CloudFormation Designer? in the *AWS CloudFormation User Guide*.

RDS supports creating resources in CloudFormation. For more information, including examples of JSON and YAML templates for these resources, see the RDS resource type reference in the *AWS CloudFormation User Guide*.

Learn more about CloudFormation

To learn more about CloudFormation, see the following resources:

- AWS CloudFormation
- AWS CloudFormation User Guide
- CloudFormation API Reference
- AWS CloudFormation Command Line Interface User Guide

Connecting to an Amazon RDS DB instance

Before you can connect to a DB instance, you must create the DB instance. For information, see Creating an Amazon RDS DB instance. After Amazon RDS provisions your DB instance, use any standard client application or utility for your DB engine to connect to the DB instance. In the connection string, specify the DNS address from the DB instance endpoint as the host parameter. Also, specify the port number from the DB instance endpoint as the port parameter.

For more information about finding connection information for an Amazon RDS DB instance or scenarios for accessing a DB instance in a VPC, see the following topics.

- Finding the connection information for an Amazon RDS DB instance
- Scenarios for accessing a DB instance in a VPC

Finding the connection information for an Amazon RDS DB instance

The connection information for a DB instance includes its endpoint, port, and a valid database user, such as the master user. For example, for a MySQL DB instance, suppose that the endpoint value is `mydb.123456789012.us-east-1.rds.amazonaws.com`. In this case, the port value is `3306`, and the database user is `admin`. Given this information, you specify the following values in a connection string:

- For host or host name or DNS name, specify `mydb.123456789012.us-east-1.rds.amazonaws.com`.
- For port, specify `3306`.
- For user, specify `admin`.

The endpoint is unique for each DB instance, and the values of the port and user can vary. The following list shows the most common port for each DB engine:

- Db2 – 50000
- MariaDB – 3306
- Microsoft SQL Server – 1433
- MySQL – 3306
- Oracle – 1521

- PostgreSQL – 5432

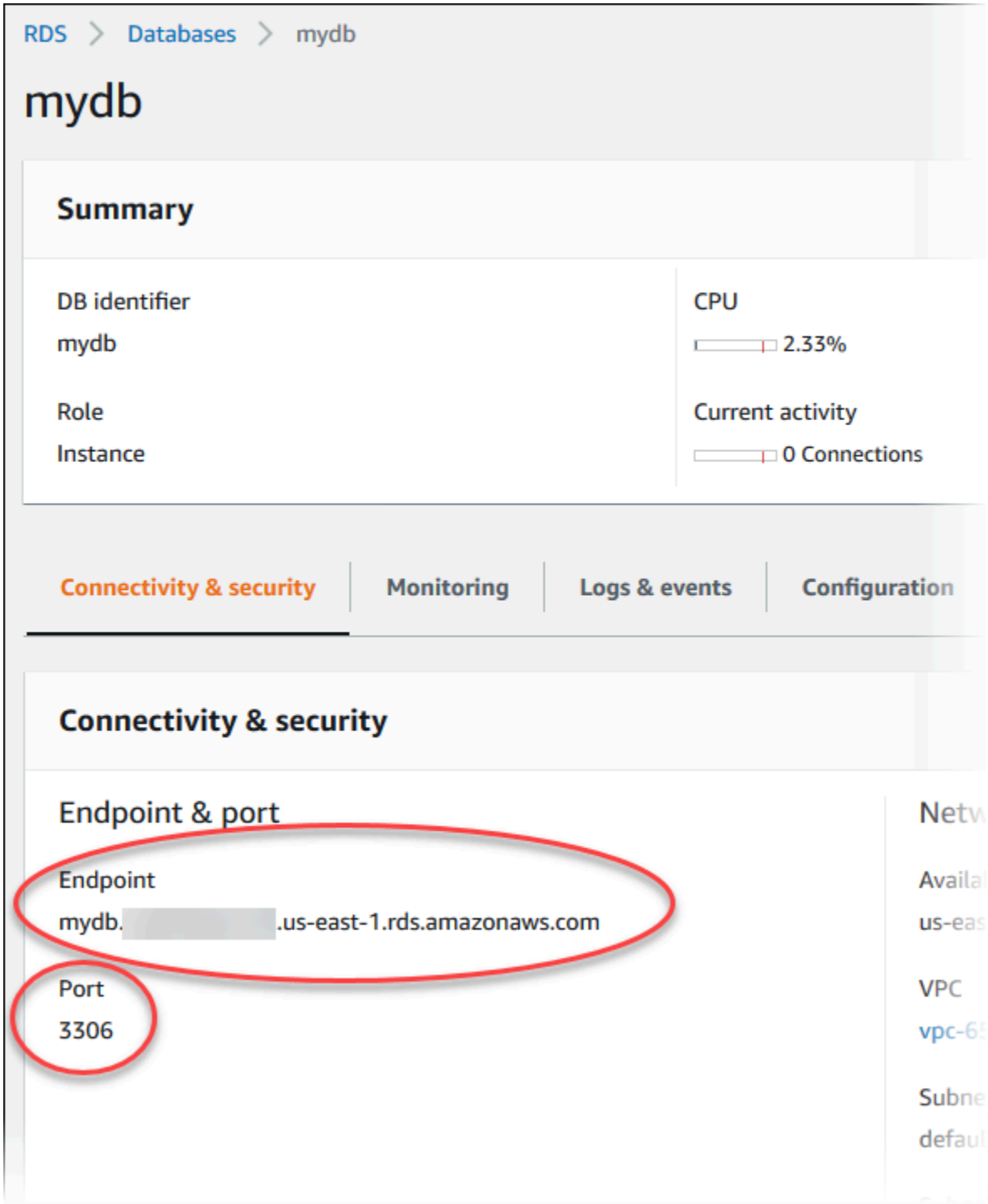
To connect to a DB instance, use any client for a DB engine. For example, you might use the mysql utility to connect to a MariaDB or MySQL DB instance. You might use Microsoft SQL Server Management Studio to connect to a SQL Server DB instance. You might use Oracle SQL Developer to connect to an Oracle DB instance. Similarly, you might use the psql command line utility to connect to a PostgreSQL DB instance.

To find the connection information for a DB instance, use the AWS Management Console. You can also use the AWS Command Line Interface (AWS CLI) describe-db-instances command or the RDS API DescribeDBInstances operation.

Console

To find the connection information for a DB instance in the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases** to display a list of your DB instances.
3. Choose the name of the DB instance to display its details.
4. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



5. If you need to find the master user name, choose the **Configuration** tab and view the **Master username** value.

AWS CLI

To find the connection information for a DB instance by using the AWS CLI, call the describe-db-instances command. In the call, query for the DB instance ID, endpoint, port, and master user name.

For Linux, macOS, or Unix:

```
aws rds describe-db-instances \
  --query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

For Windows:

```
aws rds describe-db-instances ^
  --query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

Your output should be similar to the following.

```
[
    [
        "mydb",
        "mydb.123456789012.us-east-1.rds.amazonaws.com",
        3306,
        "admin"
    ],
    [
        "myoracledb",
        "myoracledb.123456789012.us-east-1.rds.amazonaws.com",
        1521,
        "dbadmin"
    ],
    [
        "mypostgresqldb",
        "mypostgresqldb.123456789012.us-east-1.rds.amazonaws.com",
        5432,
        "postgresadmin"
    ]
]
```

RDS API

To find the connection information for a DB instance by using the Amazon RDS API, call the DescribeDBInstances operation. In the output, find the values for the endpoint address, endpoint port, and master user name.

Scenarios for accessing a DB instance in a VPC

Using Amazon Virtual Private Cloud (Amazon VPC), you can launch AWS resources, such as Amazon RDS DB instances, into a virtual private cloud (VPC). When you use Amazon VPC, you have control over your virtual networking environment. You can choose your own IP address range, create subnets, and configure routing and access control lists.

A VPC security group controls access to DB instances inside a VPC. Each VPC security group rule enables a specific source to access a DB instance in a VPC that is associated with that VPC security group. The source can be a range of addresses (for example, 203.0.113.0/24), or another VPC security group. By specifying a VPC security group as the source, you allow incoming traffic from all instances (typically application servers) that use the source VPC security group.

Before attempting to connect to your DB instance, configure your VPC for your use case. The following are common scenarios for accessing a DB instance in a VPC:

- **A DB instance in a VPC accessed by an Amazon EC2 instance in the same VPC** – A common use of a DB instance in a VPC is to share data with an application server that is running in an EC2 instance in the same VPC. The EC2 instance might run a web server with an application that interacts with the DB instance.
- **A DB instance in a VPC accessed by an EC2 instance in a different VPC** – In some cases, your DB instance is in a different VPC from the EC2 instance that you're using to access it. If so, you can use VPC peering to access the DB instance.
- **A DB instance in a VPC accessed by a client application through the internet** – To access a DB instance in a VPC from a client application through the internet, you configure a VPC with a single public subnet. You also configure an internet gateway to enable communication over the internet.

  To connect to a DB instance from outside of its VPC, the DB instance must be publicly accessible. Also, access must be granted using the inbound rules of the DB instance's security group, and other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance.

- **A DB instance in a VPC accessed by a private network** – If your DB instance isn't publicly accessible, you can use one of the following options to access it from a private network:
  - An AWS Site-to-Site VPN connection
  - An Direct Connect connection
  - An AWS Client VPN connection

For more information, see Scenarios for accessing a DB instance in a VPC.

Connecting to DB instances with the AWS drivers

The AWS suite of drivers has been designed to provide support for faster switchover and failover times, and authentication with AWS Secrets Manager, AWS Identity and Access Management (IAM), and Federated Identity. The AWS drivers rely on monitoring DB instance status and being aware of the instance topology to determine the new primary instance. This approach reduces switchover and failover times to single-digit seconds, compared to tens of seconds for open-source drivers.

The following table lists the features supported for each of the drivers. As new service features are introduced, the goal of the AWS suite of drivers is to have built-in support for these service features.

Feature	AWS JDBC Driver	AWS Python Driver	AWS ODBC Driver for MySQL
Failover support	Yes	Yes	Yes
Enhanced failover monitoring	Yes	Yes	Yes
Read/write splitting	Yes	Yes	No
Driver metadata connection	Yes	N/A	N/A
Telemetry	Yes	Yes	No
Secrets Manager	Yes	Yes	Yes
IAM authentication	Yes	Yes	Yes

Feature	AWS JDBC Driver	AWS Python Driver	AWS ODBC Driver for MySQL
Federated Identity (AD FS)	Yes	Yes	No
Federated Identity (Okta)	Yes	No	No
Multi-AZ DB clusters	Yes	Yes	No

For more information on the AWS drivers, see the corresponding language driver for your RDS for MariaDB, RDS for MySQL, or RDS for PostgreSQL DB instance.

> **Note**
>
> The only features supported for RDS for MariaDB are authentication with AWS Secrets Manager, AWS Identity and Access Management (IAM), and Federated Identity.

Connecting to a DB instance that's running a specific DB engine

To learn how to connect to a DB instance that is running a specific DB engine, follow the instructions for your DB engine:

- RDS for Db2
- RDS for MariaDB
- RDS for SQL Server
- RDS for MySQL
- RDS for Oracle
- RDS for PostgreSQL

Managing connections with RDS Proxy

You can also use Amazon RDS Proxy to manage connections to RDS for MariaDB, RDS for Microsoft SQL Server, RDS for MySQL, and RDS for PostgreSQL DB instances. RDS Proxy allows applications

to pool and share database connections to improve scalability. For more information, see Amazon RDS Proxy.

Database authentication options

Amazon RDS supports the following ways to authenticate database users:

- **Password authentication** – Your DB instance performs all administration of user accounts. You create users and specify passwords with SQL statements. The SQL statements you can use depend on your DB engine.
- **AWS Identity and Access Management (IAM) database authentication** – You don't need to use a password when you connect to a DB instance. Instead, you use an authentication token.
- **Kerberos authentication** – You use external authentication of database users using Kerberos and Microsoft Active Directory. Kerberos is a network authentication protocol that uses tickets and symmetric-key cryptography to eliminate the need to transmit passwords over the network. Kerberos has been built into Active Directory and is designed to authenticate users to network resources, such as databases.

IAM database authentication and Kerberos authentication are available only for specific DB engines and versions.

For more information, see Database authentication with Amazon RDS .

Encrypted connections

You can use Secure Socket Layer (SSL) or Transport Layer Security (TLS) from your application to encrypt a connection to a DB instance. Each DB engine has its own process for implementing SSL/TLS. For more information, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

Working with option groups

Some DB engines offer additional features that make it easier to manage data and databases, and to provide additional security for your database. Amazon RDS uses option groups to enable and configure these features. An *option group* can specify features, called options, that are available for a particular Amazon RDS DB instance. Options can have settings that specify how the option works. When you associate a DB instance with an option group, the specified options and option settings are enabled for that DB instance.

Amazon RDS supports options for the following database engines:

Database engine	Relevant documentation
`Db2`	Options for RDS for Db2 DB instances
`MariaDB`	Options for MariaDB database engine
`Microsoft SQL Server`	Options for the Microsoft SQL Server database engine
`MySQL`	Options for MySQL DB instances
`Oracle`	Adding options to Oracle DB instances
`PostgreSQL`	PostgreSQL does not use options and option groups. PostgreSQL uses extensions and modules to provide additional features. For more information, see Supported PostgreSQL extension versions.

Option groups overview

Amazon RDS provides an empty default option group for each new DB instance. You can't modify or delete this default option group, but any new option group that you create derives its settings from the default option group. To apply an option to a DB instance, you must do the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add one or more options to the option group.
3. Associate the option group with the DB instance.

To associate an option group with a DB instance, modify the DB instance. For more information, see Modifying an Amazon RDS DB instance.

Both DB instances and DB snapshots can be associated with an option group. In some cases, you might restore from a DB snapshot or perform a point-in-time restore for a DB instance. In these cases, the option group associated with the DB snapshot or DB instance is, by default, associated with the restored DB instance. You can associate a different option group with a restored DB instance. However, the new option group must contain any persistent or permanent options that were included in the original option group. Persistent and permanent options are described following.

Options require additional memory to run on a DB instance. Thus, you might need to launch a larger instance to use them, depending on your current use of your DB instance. For example, Oracle Enterprise Manager Database Control uses about 300 MB of RAM. If you enable this option for a small DB instance, you might encounter performance problems or out-of-memory errors.

Persistent and permanent options

Two types of options, persistent and permanent, require special consideration when you add them to an option group.

Persistent options can't be removed from an option group while DB instances are associated with the option group. An example of a persistent option is the TDE option for Microsoft SQL Server transparent data encryption (TDE). You must disassociate all DB instances from the option group before a persistent option can be removed from the option group. In some cases, you might restore or perform a point-in-time restore from a DB snapshot. In these cases, if the option group associated with that DB snapshot contains a persistent option, you can only associate the restored DB instance with that option group.

Permanent options, such as the TDE option for Oracle Advanced Security TDE, can never be removed from an option group. You can change the option group of a DB instance that is using the permanent option. However, the option group associated with the DB instance must include the same permanent option. In some cases, you might restore or perform a point-in-time restore from a DB snapshot. In these cases, if the option group associated with that DB snapshot contains a permanent option, you can only associate the restored DB instance with an option group with that permanent option.

For Oracle DB instances, you can copy shared DB snapshots that have the options `Timezone` or `OLS` (or both). To do so, specify a target option group that includes these options when you copy the DB snapshot. The OLS option is permanent and persistent only for Oracle DB instances running Oracle version 12.2 or higher. For more information about these options, see Oracle time zone and Oracle Label Security.

VPC considerations

The option group associated with the DB instance is linked to the DB instance's VPC. This means that you can't use the option group assigned to a DB instance if you try to restore the instance to a different VPC. If you restore a DB instance to a different VPC, you can do one of the following:

- Assign the default option group to the DB instance.
- Assign an option group that is linked to that VPC.
- Create a new option group and assign it to the DB instance.

With persistent or permanent options, such as Oracle TDE, you must create a new option group. This option group must include the persistent or permanent option when restoring a DB instance into a different VPC.

Option settings control the behavior of an option. For example, the Oracle Advanced Security option `NATIVE_NETWORK_ENCRYPTION` has a setting that you can use to specify the encryption algorithm for network traffic to and from the DB instance. Some options settings are optimized for use with Amazon RDS and cannot be changed.

Mutually exclusive options

Some options are mutually exclusive. You can use one or the other, but not both at the same time. The following options are mutually exclusive:

- Oracle Enterprise Manager Database Express and Oracle Management Agent for Enterprise Manager Cloud Control.
- Oracle native network encryption and Oracle Secure Sockets Layer.

Creating an option group

You can create a new option group that derives its settings from the default option group. You then add one or more options to the new option group. Or, if you already have an existing option

group, you can copy that option group with all of its options to a new option group. For more information, see Copying an option group.

After you create a new option group, it has no options. To learn how to add options to the option group, see Adding an option to an option group. After you have added the options you want, you can then associate the option group with a DB instance. This way, the options become available on the DB instance. For information about associating an option group with a DB instance, see the documentation for your engine in Working with option groups.

Console

One way of creating an option group is by using the AWS Management Console.

To create a new option group by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** window, do the following:
    a. For **Name**, type a name for the option group that is unique within your AWS account. The name can contain only letters, digits, and hyphens.
    b. For **Description**, type a brief description of the option group. The description is used for display purposes.
    c. For **Engine**, choose the DB engine that you want.
    d. For **Major engine version**, choose the major version of the DB engine that you want.
5. To continue, choose **Create**. To cancel the operation instead, choose **Cancel**.

AWS CLI

To create an option group, use the AWS CLI `create-option-group` command with the following required parameters.

- `--option-group-name`
- `--engine-name`
- `--major-engine-version`

- `--option-group-description`

Example

The following example creates an option group named `testoptiongroup`, which is associated with the Oracle Enterprise Edition DB engine. The description is enclosed in quotation marks.

For Linux, macOS, or Unix:

```
aws rds create-option-group \
    --option-group-name testoptiongroup \
    --engine-name oracle-ee \
    --major-engine-version 19 \
    --option-group-description "Test option group for Oracle Database 19c EE"
```

For Windows:

```
aws rds create-option-group ^
    --option-group-name testoptiongroup ^
    --engine-name oracle-ee ^-
    --major-engine-version 19 ^
    --option-group-description "Test option group for Oracle Database 19c EE"
```

RDS API

To create an option group, call the Amazon RDS API `CreateOptionGroup` operation. Include the following parameters:

- `OptionGroupName`
- `EngineName`
- `MajorEngineVersion`
- `OptionGroupDescription`

Copying an option group

You can use the AWS CLI or the Amazon RDS API copy an option group. Copying an option group can be convenient. An example is when you have an existing option group and want to include most of its custom parameters and values in a new option group. You can also make a copy of an option group that you use in production and then modify the copy to test other option settings.

> **Note**
>
> Currently, you can't copy an option group to a different AWS Region.

AWS CLI

To copy an option group, use the AWS CLI copy-option-group command. Include the following required options:

- `--source-option-group-identifier`
- `--target-option-group-identifier`
- `--target-option-group-description`

Example

The following example creates an option group named `new-option-group`, which is a local copy of the option group `my-option-group`.

For Linux, macOS, or Unix:

```
aws rds copy-option-group \
    --source-option-group-identifier my-option-group \
    --target-option-group-identifier new-option-group \
    --target-option-group-description "My new option group"
```

For Windows:

```
aws rds copy-option-group ^
    --source-option-group-identifier my-option-group ^
    --target-option-group-identifier new-option-group ^
    --target-option-group-description "My new option group"
```

RDS API

To copy an option group, call the Amazon RDS API CopyOptionGroup operation. Include the following required parameters.

- `SourceOptionGroupIdentifier`
- `TargetOptionGroupIdentifier`
- `TargetOptionGroupDescription`

Adding an option to an option group

You can add an option to an existing option group. After you have added the options you want, you can then associate the option group with a DB instance so that the options become available on the DB instance. For information about associating an option group with a DB instance, see the documentation for your specific DB engine listed at Working with option groups.

Option group changes must be applied immediately in two cases:

- When you add an option that adds or updates a port value, such as the `OEM` option.
- When you add or remove an option group with an option that includes a port value.

In these cases, choose the **Apply Immediately** option in the console. Or you can include the `--apply-immediately` option when using the AWS CLI or set the `ApplyImmediately` parameter to `true` when using the Amazon RDS API. Options that don't include port values can be applied immediately, or can be applied during the next maintenance window for the DB instance.

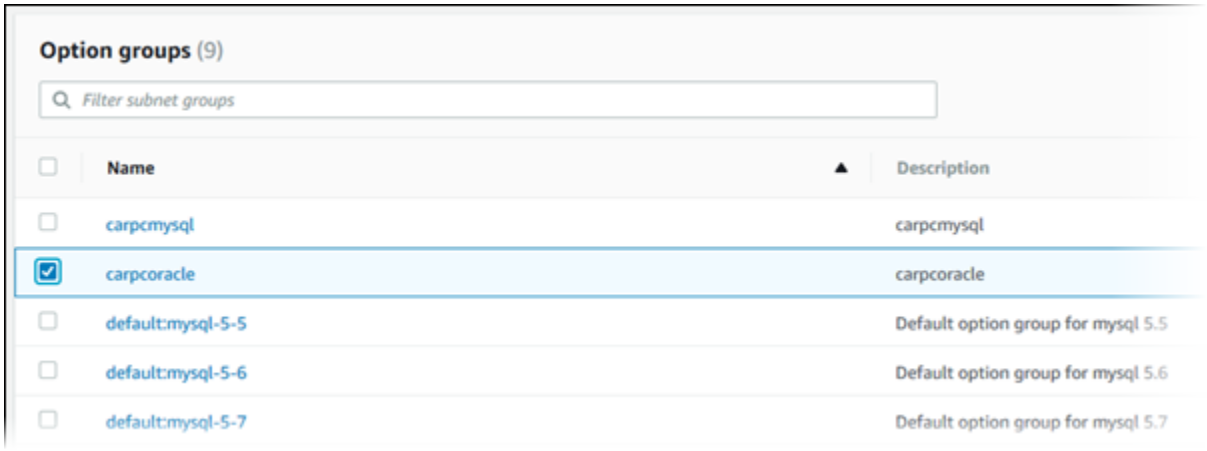
> **Note**
>
> If you specify a security group as a value for an option in an option group, manage the security group by modifying the option group. You can't change or remove this security group by modifying a DB instance. Also, the security group doesn't appear in the DB instance details in the AWS Management Console or in the output for the AWS CLI command `describe-db-instances`.

Console

You can use the AWS Management Console to add an option to an option group.

To add an option to an option group by using the console

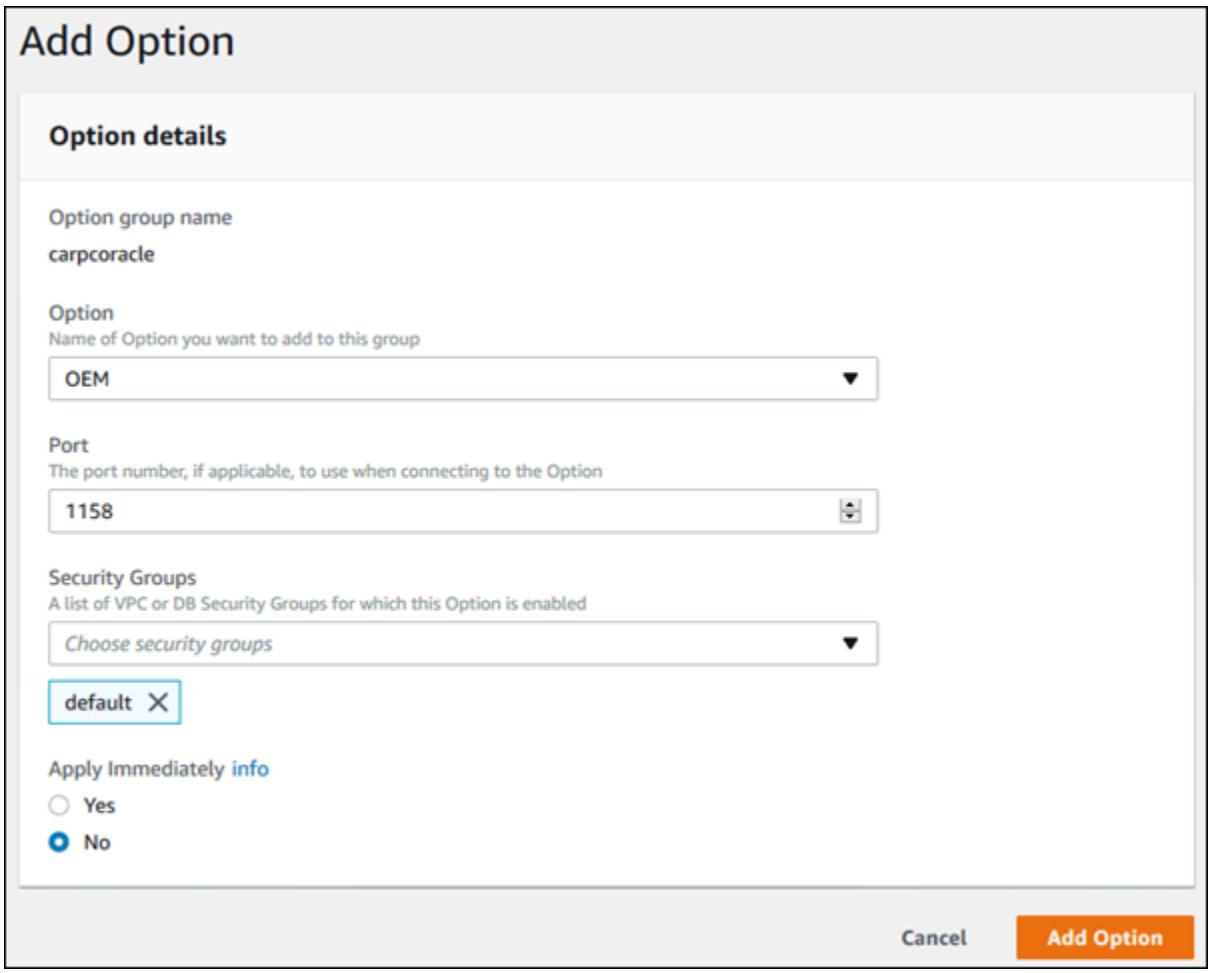
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group that you want to modify, and then choose **Add option**.



4. In the **Add option** window, do the following:

   a. Choose the option that you want to add. You might need to provide additional values, depending on the option that you select. For example, when you choose the `OEM` option, you must also type a port value and specify a security group.

   b. To enable the option on all associated DB instances as soon as you add it, for **Apply Immediately**, choose **Yes**. If you choose **No** (the default), the option is enabled for each associated DB instance during its next maintenance window.



5. When the settings are as you want them, choose **Add option**.

AWS CLI

To add an option to an option group, run the AWS CLI add-option-to-option-group command with the option that you want to add. To enable the new option immediately on all associated DB instances, include the `--apply-immediately` parameter. By default, the option is enabled for each associated DB instance during its next maintenance window. Include the following required parameter:

- `--option-group-name`

Example

The following example adds the `Timezone` option, with the `America/Los_Angeles` setting, to an option group named `testoptiongroup` and immediately enables it.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name testoptiongroup \
    --options "OptionName=Timezone,OptionSettings=[{Name=TIME_ZONE,Value=America/
Los_Angeles}]" \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name testoptiongroup ^
    --options "OptionName=Timezone,OptionSettings=[{Name=TIME_ZONE,Value=America/
Los_Angeles}]" ^
    --apply-immediately
```

Command output is similar to the following:

```
...{
       "OptionName": "Timezone",
       "OptionDescription": "Change time zone",
       "Persistent": true,
       "Permanent": false,
       "OptionSettings": [
           {
               "Name": "TIME_ZONE",
               "Value": "America/Los_Angeles",
               "DefaultValue": "UTC",
               "Description": "Specifies the timezone the user wants to change the
 system time to",
               "ApplyType": "DYNAMIC",
               "DataType": "STRING",
               "AllowedValues": "Africa/Cairo,...",
               "IsModifiable": true,
               "IsCollection": false
           }
       ],
```

```
        "DBSecurityGroupMemberships": [],
        "VpcSecurityGroupMemberships": []
    }...
```

Example

The following example adds the Oracle OEM option to an option group. It also specifies a custom port and a pair of Amazon EC2 VPC security groups to use for that port.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name testoptiongroup \
    --options OptionName=OEM,Port=5500,VpcSecurityGroupMemberships="sg-test1,sg-test2"
 \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name testoptiongroup ^
    --options OptionName=OEM,Port=5500,VpcSecurityGroupMemberships="sg-test1,sg-test2"
 ^
    --apply-immediately
```

Command output is similar to the following:

```
OPTIONGROUP  False  oracle-ee  19 arn:aws:rds:us-east-1:1234567890:og:testoptiongroup
 Test Option Group  testoptiongroup vpc-test
OPTIONS Oracle 12c EM Express   OEM     False   False   5500
VPCSECURITYGROUPMEMBERSHIPS     active  sg-test1
VPCSECURITYGROUPMEMBERSHIPS     active  sg-test2
```

Example

The following example adds the Oracle option `NATIVE_NETWORK_ENCRYPTION` to an option group and specifies the option settings. If no option settings are specified, default values are used.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
```

```
    --option-group-name testoptiongroup \
    --options '[{"OptionSettings":
[{"Name":"SQLNET.ENCRYPTION_SERVER","Value":"REQUIRED"},
{"Name":"SQLNET.ENCRYPTION_TYPES_SERVER","Value":"AES256,AES192,DES"}],"OptionName":"NATIVE_NET
 \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name testoptiongroup ^
    --options "OptionSettings"=[{"Name"="SQLNET.ENCRYPTION_SERVER","Value"="REQUIRED"},
{"Name"="SQLNET.ENCRYPTION_TYPES_SERVER","Value"="AES256\,AES192\,DES"}],"OptionName"="NATIVE_N
 ^
    --apply-immediately
```

Command output is similar to the following:

```
...{
    "OptionName": "NATIVE_NETWORK_ENCRYPTION",
    "OptionDescription": "Native Network Encryption",
    "Persistent": false,
    "Permanent": false,
    "OptionSettings": [
        {
            "Name": "SQLNET.ENCRYPTION_TYPES_SERVER",
             "Value": "AES256,AES192,DES",
            "DefaultValue":
 "RC4_256,AES256,AES192,3DES168,RC4_128,AES128,3DES112,RC4_56,DES,RC4_40,DES40",
            "Description": "Specifies list of encryption algorithms in order of
 intended use",
            "ApplyType": "STATIC",
            "DataType": "STRING",
            "AllowedValues":
 "RC4_256,AES256,AES192,3DES168,RC4_128,AES128,3DES112,RC4_56,DES,RC4_40,DES40",
            "IsModifiable": true,
            "IsCollection": true
        },
        {
            "Name": "SQLNET.ENCRYPTION_SERVER",
            "Value": "REQUIRED",
            "DefaultValue": "REQUESTED",
            "Description": "Specifies the desired encryption behavior",
```

```
            "ApplyType": "STATIC",
            "DataType": "STRING",
            "AllowedValues": "ACCEPTED,REJECTED,REQUESTED,REQUIRED",
            "IsModifiable": true,
            "IsCollection": false
        },...
```

RDS API

To add an option to an option group using the Amazon RDS API, call the ModifyOptionGroup operation with the option that you want to add. To enable the new option immediately on all associated DB instances, include the `ApplyImmediately` parameter and set it to `true`. By default, the option is enabled for each associated DB instance during its next maintenance window. Include the following required parameter:

- `OptionGroupName`

Listing the options and option settings for an option group

You can list all the options and option settings for an option group.

Console

You can use the AWS Management Console to list all of the options and option settings for an option group.

To list the options and option settings for an option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the name of the option group to display its details. The options and option settings in the option group are listed.

AWS CLI

To list the options and option settings for an option group, use the AWS CLI `describe-option-groups` command. Specify the name of the option group whose options and settings you want to view. If you don't specify an option group name, all option groups are described.

Example

The following example lists the options and option settings for all option groups.

```
aws rds describe-option-groups
```

Example

The following example lists the options and option settings for an option group named `testoptiongroup`.

```
aws rds describe-option-groups --option-group-name testoptiongroup
```

RDS API

To list the options and option settings for an option group, use the Amazon RDS API `DescribeOptionGroups` operation. Specify the name of the option group whose options and settings you want to view. If you don't specify an option group name, all option groups are described.

Modifying an option setting

After you have added an option that has modifiable option settings, you can modify the settings at any time. If you change options or option settings in an option group, those changes are applied to all DB instances that are associated with that option group. For more information on what settings are available for the various options, see the documentation for your engine in Working with option groups.

Option group changes must be applied immediately in two cases:

- When you add an option that adds or updates a port value, such as the `OEM` option.
- When you add or remove an option group with an option that includes a port value.

In these cases, choose the **Apply Immediately** option in the console. Or you can include the `--apply-immediately` option when using the AWS CLI or set the `ApplyImmediately` parameter to `true` when using the RDS API. Options that don't include port values can be applied immediately, or can be applied during the next maintenance window for the DB instance.

> **Note**
>
> If you specify a security group as a value for an option in an option group, you manage the security group by modifying the option group. You can't change or remove this security group by modifying a DB instance. Also, the security group doesn't appear in the DB instance details in the AWS Management Console or in the output for the AWS CLI command `describe-db-instances`.

Console

You can use the AWS Management Console to modify an option setting.

To modify an option setting by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Select the option group whose option that you want to modify, and then choose **Modify option**.
4. In the **Modify option** window, from **Installed Options**, choose the option whose setting you want to modify. Make the changes that you want.
5. To enable the option as soon as you add it, for **Apply Immediately**, choose **Yes**. If you choose **No** (the default), the option is enabled for each associated DB instance during its next maintenance window.
6. When the settings are as you want them, choose **Modify Option**.

AWS CLI

To modify an option setting, use the AWS CLI `add-option-to-option-group` command with the option group and option that you want to modify. By default, the option is enabled for each associated DB instance during its next maintenance window. To apply the change immediately to all associated DB instances, include the `--apply-immediately` parameter. To modify an option setting, use the `--settings` argument.

Example

The following example modifies the port that the Oracle Enterprise Manager Database Control (OEM) uses in an option group named `testoptiongroup` and immediately applies the change.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name testoptiongroup \
    --options OptionName=OEM,Port=5432,DBSecurityGroupMemberships=default \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name testoptiongroup ^
    --options OptionName=OEM,Port=5432,DBSecurityGroupMemberships=default ^
    --apply-immediately
```

Command output is similar to the following:

```
OPTIONGROUP   False  oracle-ee  19  arn:aws:rds:us-east-1:1234567890:og:testoptiongroup
   Test Option Group    testoptiongroup
OPTIONS Oracle 12c EM Express   OEM     False   False   5432
DBSECURITYGROUPMEMBERSHIPS   default  authorized
```

Example

The following example modifies the Oracle option `NATIVE_NETWORK_ENCRYPTION` and changes the option settings.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name testoptiongroup \
    --options '[{"OptionSettings":
[{"Name":"SQLNET.ENCRYPTION_SERVER","Value":"REQUIRED"},
{"Name":"SQLNET.ENCRYPTION_TYPES_SERVER","Value":"AES256,AES192,DES,RC4_256"}],"OptionName":"NA
 \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name testoptiongroup ^
    --options "OptionSettings"=[{"Name"="SQLNET.ENCRYPTION_SERVER","Value"="REQUIRED"},
{"Name"="SQLNET.ENCRYPTION_TYPES_SERVER","Value"="AES256\,AES192\,DES
\,RC4_256"}],"OptionName"="NATIVE_NETWORK_ENCRYPTION" ^
    --apply-immediately
```

Command output is similar to the following:

```
OPTIONGROUP   False  oracle-ee  19  arn:aws:rds:us-east-1:1234567890:og:testoptiongroup
   Test Option Group    testoptiongroup
OPTIONS Oracle Advanced Security - Native Network Encryption
 NATIVE_NETWORK_ENCRYPTION       False   False
OPTIONSETTINGS
 RC4_256,AES256,AES192,3DES168,RC4_128,AES128,3DES112,RC4_56,DES,RC4_40,DES40 STATIC
 STRING
   RC4_256,AES256,AES192,3DES168,RC4_128,AES128,3DES112,RC4_56,DES,RC4_40,DES40
 Specifies list of encryption algorithms in order of intended use
   True     True    SQLNET.ENCRYPTION_TYPES_SERVER  AES256,AES192,DES,RC4_256
OPTIONSETTINGS  ACCEPTED,REJECTED,REQUESTED,REQUIRED    STATIC  STRING  REQUESTED
 Specifies the desired encryption behavior  False   True  SQLNET.ENCRYPTION_SERVER
 REQUIRED
OPTIONSETTINGS  SHA1,MD5   STATIC  STRING  SHA1,MD5    Specifies list of
 checksumming algorithms in order of intended use      True    True
 SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER  SHA1,MD5
OPTIONSETTINGS  ACCEPTED,REJECTED,REQUESTED,REQUIRED  STATIC  STRING
 REQUESTED     Specifies the desired data integrity behavior   False   True
 SQLNET.CRYPTO_CHECKSUM_SERVER  REQUESTED
```

RDS API

To modify an option setting, use the Amazon RDS API `ModifyOptionGroup` command with the option group and option that you want to modify. By default, the option is enabled for each associated DB instance during its next maintenance window. To apply the change immediately to all associated DB instances, include the `ApplyImmediately` parameter and set it to `true`.

Removing an option from an option group

Some options can be removed from an option group, and some cannot. A persistent option cannot be removed from an option group until all DB instances associated with that option group are disassociated. A permanent option can never be removed from an option group. For more

information about what options are removable, see the documentation for your specific engine listed at Working with option groups.

If you remove all options from an option group, Amazon RDS doesn't delete the option group. DB instances that are associated with the empty option group continue to be associated with it; they just won't have any active options. Alternatively, to remove all options from a DB instance, you can associate the DB instance with the default (empty) option group.

Console

You can use the AWS Management Console to remove an option from an option group.

To remove an option from an option group by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Select the option group whose option you want to remove, and then choose **Delete option**.
4. In the **Delete option** window, do the following:
   - Select the check box for the option that you want to delete.
   - For the deletion to take effect as soon as you make it, for **Apply immediately**, choose **Yes**. If you choose **No** (the default), the option is deleted for each associated DB instance during its next maintenance window.



5. When the settings are as you want them, choose **Yes, Delete**.

AWS CLI

To remove an option from an option group, use the AWS CLI `remove-option-from-option-group` command with the option that you want to delete. By default, the option is removed from each associated DB instance during its next maintenance window. To apply the change immediately, include the `--apply-immediately` parameter.

Example

The following example removes the Oracle Enterprise Manager Database Control (OEM) option from an option group named `testoptiongroup` and immediately applies the change.

For Linux, macOS, or Unix:

```
aws rds remove-option-from-option-group \
    --option-group-name testoptiongroup \
    --options OEM \
    --apply-immediately
```

For Windows:

```
aws rds remove-option-from-option-group ^
    --option-group-name testoptiongroup ^
    --options OEM ^
    --apply-immediately
```

Command output is similar to the following:

```
OPTIONGROUP    testoptiongroup oracle-ee   19    Test option group
```

RDS API

To remove an option from an option group, use the Amazon RDS API `ModifyOptionGroup` action. By default, the option is removed from each associated DB instance during its next maintenance window. To apply the change immediately, include the `ApplyImmediately` parameter and set it to `true`.

Include the following parameters:

- `OptionGroupName`
- `OptionsToRemove.OptionName`

Deleting an option group

You can delete an option group only if it meets the following criteria:

- It is not associated with any Amazon RDS resource. An option group can be associated with a DB instance, a manual DB snapshot, or an automated DB snapshot.
- It is not a default option group.

To identify the option groups used by your DB instances and DB snapshots, you can use the following CLI commands:

```
aws rds describe-db-instances \
    --query 'DBInstances[*].
[DBInstanceIdentifier,OptionGroupMemberships[].OptionGroupName]'

aws rds describe-db-snapshots | jq -r '.DBSnapshots[] | "\(.DBInstanceIdentifier),
\(.OptionGroupName)"' | sort | uniq
```

If you try to delete an option group that is associated with an RDS resource, an error like the following is returned.

```
An error occurred (InvalidOptionGroupStateFault) when calling the DeleteOptionGroup
 operation: The option group 'optionGroupName' cannot be deleted because it is in use.
```

To find the Amazon RDS resources associated with an option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the name of the option group to show its details.

4. Check the **Associated Instances and Snapshots** section for the associated Amazon RDS resources.

If a DB instance is associated with the option group, modify the DB instance to use a different option group. For more information, see Modifying an Amazon RDS DB instance.

If a manual DB snapshot is associated with the option group, modify the DB snapshot to use a different option group. You can do so using the AWS CLI `modify-db-snapshot` command.

> **Note**
>
> You can't modify the option group of an automated DB snapshot.

Console

One way of deleting an option group is by using the AWS Management Console.

To delete an option group by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group.
4. Choose **Delete group**.
5. On the confirmation page, choose **Delete** to finish deleting the option group, or choose **Cancel** to cancel the deletion.

AWS CLI

To delete an option group, use the AWS CLI `delete-option-group` command with the following required parameter.

- `--option-group-name`

Example

The following example deletes an option group named `testoptiongroup`.

For Linux, macOS, or Unix:

```
aws rds delete-option-group \
    --option-group-name testoptiongroup
```

For Windows:

```
aws rds delete-option-group ^
    --option-group-name testoptiongroup
```

RDS API

To delete an option group, call the Amazon RDS API `DeleteOptionGroup` operation. Include the following parameter:

- `OptionGroupName`

Parameter groups for Amazon RDS

Database parameters specify how the database is configured. For example, database parameters can specify the amount of resources, such as memory, to allocate to a database.

You manage your database configuration by associating your DB instances and Multi-AZ DB clusters with parameter groups. Amazon RDS defines parameter groups with default settings. You can also define your own parameter groups with customized settings.

> **Note**
>
> Some DB engines offer additional features that you can add to your database as options in an option group. For information about option groups, see Working with option groups.

Topics

- Overview of parameter groups
- DB parameter groups for Amazon RDS DB instances
- Working with DB cluster parameter groups for Multi-AZ DB clusters
- Comparing DB parameter groups
- Specifying DB parameters

Overview of parameter groups

A *DB parameter group* acts as a container for engine configuration values that are applied to one or more DB instances.

DB cluster parameter groups apply to Multi-AZ DB clusters only. In a Multi-AZ DB cluster, the settings in the DB cluster parameter group apply to all of the DB instances in the cluster. The default DB parameter group for the DB engine and DB engine version is used for each DB instance in the DB cluster.

Topics

- Default and custom parameter groups
- Static and dynamic DB instance parameters
- Static and dynamic DB cluster parameters

- Character set parameters
- Supported parameters and parameter values

Default and custom parameter groups

If you create a DB instance without specifying a DB parameter group, the DB instance uses a default DB parameter group. Likewise, if you create a Multi-AZ DB cluster without specifying a DB cluster parameter group, the DB cluster uses a default DB cluster parameter group. Each default parameter group contains database engine defaults and Amazon RDS system defaults based on the engine, compute class, and allocated storage of the instance.

You can't modify the parameter settings of a default parameter group. Instead, you can do the following:

1. Create a new parameter group.
2. Change the settings of your desired parameters. Not all DB engine parameters in a parameter group are eligible to be modified.
3. Modify your DB instance or DB cluster to associate the new parameter group.

   When you associate a new DB parameter group with a DB instance, the association happens immediately. For information about modifying a DB instance, see Modifying an Amazon RDS DB instance. For information about modifying a Multi-AZ DB clusters, see Modifying a Multi-AZ DB cluster for Amazon RDS.

   > **Note**
   >
   > If you have modified your DB instance to use a custom parameter group, and you start the DB instance, RDS automatically reboots the DB instance as part of the startup process. For RDS for SQL Server Multi-AZ instances with AlwaysOn or Mirroring option enabled, a failover is expected when instance is rebooted after the startup process.

RDS applies the modified static and dynamic parameters in a newly associated parameter group only after the DB instance is rebooted. However, if you modify dynamic parameters in the DB parameter group after you associate it with the DB instance, these changes are applied immediately without a reboot. For more information about changing the DB parameter group, see Modifying an Amazon RDS DB instance.

If you update parameters within a DB parameter group, the changes apply to all DB instances that are associated with that parameter group. Likewise, if you update parameters within a Multi-AZ DB cluster parameter group, the changes apply to all Aurora DB clusters that are associated with that DB cluster parameter group.

If you don't want to create a parameter group from scratch, you can copy an existing parameter group with the AWS CLI copy-db-parameter-group command or copy-db-cluster-parameter-group command. You might find that copying a parameter group is useful in some cases. For example, you might want to include most of an existing DB parameter group's custom parameters and values in a new DB parameter group.

Static and dynamic DB instance parameters

DB instance parameters are either static or dynamic. They differ as follows:

- When you change a static parameter and save the DB parameter group, the parameter change takes effect after you manually reboot the associated DB instances. For static parameters, the console always uses `pending-reboot` for the `ApplyMethod`.
- When you change a dynamic parameter, by default the parameter change takes effect immediately, without requiring a reboot. When you use the AWS Management Console to change DB instance parameter values, it always uses `immediate` for the `ApplyMethod` for dynamic parameters. To defer the parameter change until after you reboot an associated DB instance, use the AWS CLI or RDS API. Set the `ApplyMethod` to `pending-reboot` for the parameter change.

> **Note**
>
> Using `pending-reboot` with dynamic parameters in the AWS CLI or RDS API on RDS for SQL Server DB instances generates an error. Use `apply-immediately` on RDS for SQL Server.

For more information about using the AWS CLI to change a parameter value, see modify-db-parameter-group. For more information about using the RDS API to change a parameter value, see ModifyDBParameterGroup.

If a DB instance isn't using the latest changes to its associated DB parameter group, the console shows a status of **pending-reboot** for the DB parameter group. This status doesn't result in an automatic reboot during the next maintenance window. To apply the latest parameter changes to that DB instance, manually reboot the DB instance.

Static and dynamic DB cluster parameters

DB cluster parameters are either static or dynamic. They differ as follows:

- When you change a static parameter and save the DB cluster parameter group, the parameter change takes effect after you manually reboot the associated DB clusters. For static parameters, the console always uses `pending-reboot` for the `ApplyMethod`.
- When you change a dynamic parameter, by default the parameter change takes effect immediately, without requiring a reboot. When you use the AWS Management Console to change DB cluster parameter values, it always uses `immediate` for the `ApplyMethod` for dynamic parameters. To defer the parameter change until after an associated DB cluster is rebooted, use the AWS CLI or RDS API. Set the `ApplyMethod` to `pending-reboot` for the parameter change.

For more information about using the AWS CLI to change a parameter value, see modify-db-cluster-parameter-group. For more information about using the RDS API to change a parameter value, see ModifyDBClusterParameterGroup.

Character set parameters

Before you create a DB instance or Multi-AZ DB cluster, set any parameters that relate to the character set or collation of your database in your parameter group. Also do so before you create a database in it. In this way, you ensure that the default database and new databases use the character set and collation values that you specify. If you change character set or collation parameters, the parameter changes aren't applied to existing databases.

For some DB engines, you can change character set or collation values for an existing database using the `ALTER DATABASE` command, for example:

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

For more information about changing the character set or collation values for a database, check the documentation for your DB engine.

Supported parameters and parameter values

To determine the supported parameters for your DB engine, view the parameters in the DB parameter group and DB cluster parameter group used by the DB instance or DB cluster. For more information, see Viewing parameter values for a DB parameter group in Amazon RDS and Viewing parameter values for a DB cluster parameter group.

In many cases, you can specify integer and Boolean parameter values using expressions, formulas, and functions. Functions can include a mathematical log expression. However, not all parameters support expressions, formulas, and functions for parameter values. For more information, see Specifying DB parameters.

Improperly setting parameters in a parameter group can have unintended adverse effects, including degraded performance and system instability. Always be cautious when modifying database parameters, and back up your data before modifying a parameter group. Try parameter group setting changes on a test DB instance or DB cluster before applying those parameter group changes to a production DB instance or DB cluster.

DB parameter groups for Amazon RDS DB instances

DB instances use DB parameter groups. The following sections describe configuring and managing DB instance parameter groups.

Topics

- Creating a DB parameter group in Amazon RDS
- Associating a DB parameter group with a DB instance in Amazon RDS
- Modifying parameters in a DB parameter group in Amazon RDS
- Resetting parameters in a DB parameter group to their default values in Amazon RDS
- Copying a DB parameter group in Amazon RDS
- Listing DB parameter groups in Amazon RDS
- Viewing parameter values for a DB parameter group in Amazon RDS
- Deleting a DB parameter group in Amazon RDS

Creating a DB parameter group in Amazon RDS

You can create a new DB parameter group using the AWS Management Console, the AWS CLI, or the RDS API.

The following limitations apply to the DB parameter group name:

- The name must be 1 to 255 letters, numbers, or hyphens.

  Default parameter group names can include a period, such as `default.mysql8.0`. However, custom parameter group names can't include a period.

- The first character must be a letter.
- The name can't end with a hyphen or contain two consecutive hyphens.

Console

To create a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
4. For **Parameter group name**, enter the name of your new DB parameter group.
5. For **Description**, enter a description for your new DB parameter group.
6. For **Engine type**, choose your DB engine.
7. For **Parameter group family**, choose a DB parameter group family.
8. For **Type**, if applicable, choose **DB Parameter Group**.
9. Choose **Create**.

AWS CLI

To create a DB parameter group, use the AWS CLI `create-db-parameter-group` command. The following example creates a DB parameter group named *mydbparametergroup* for MySQL version 8.0 with a description of "*My new parameter group*."

Include the following required parameters:

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

To list all of the available parameter group families, use the following command:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> The output contains duplicates.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family MySQL8.0 \
    --description "My new parameter group"
```

For Windows:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --db-parameter-group-family MySQL8.0 ^
    --description "My new parameter group"
```

This command produces output similar to the following:

```
DBPARAMETERGROUP  mydbparametergroup  mysql8.0  My new parameter group
```

RDS API

To create a DB parameter group, use the RDS API `CreateDBParameterGroup` operation.

Include the following required parameters:

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Associating a DB parameter group with a DB instance in Amazon RDS

You can create your own DB parameter groups with customized settings. You can associate a DB parameter group with a DB instance using the AWS Management Console, the AWS CLI, or the RDS API. You can do so when you create or modify a DB instance.

For information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS. For information about creating a DB instance, see Creating an Amazon RDS DB instance. For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

> **Note**
>
> When you associate a new DB parameter group with a DB instance, the modified static and dynamic parameters are applied only after the DB instance is rebooted. However, if you modify dynamic parameters in the DB parameter group after you associate it with the DB instance, these changes are applied immediately without a reboot.

Console

To associate a DB parameter group with a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**. The **Modify DB instance** page appears.
4. Change the **DB parameter group** setting.
5. Choose **Continue** and check the summary of modifications.
6. (Optional) Choose **Apply immediately** to apply the changes immediately. Choosing this option can cause an outage in some cases. For more information, see Using the schedule modifications setting.
7. On the confirmation page, review your changes. If they are correct, choose **Modify DB instance** to save your changes.

   Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To associate a DB parameter group with a DB instance, use the AWS CLI `modify-db-instance` command with the following options:

- `--db-instance-identifier`
- `--db-parameter-group-name`

The following example associates the `mydbpg` DB parameter group with the `database-1` DB instance. The changes are applied immediately by using `--apply-immediately`. Use `--no-apply-immediately` to apply the changes during the next maintenance window. For more information, see Using the schedule modifications setting.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier database-1 \
    --db-parameter-group-name mydbpg \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier database-1 ^
    --db-parameter-group-name mydbpg ^
    --apply-immediately
```

RDS API

To associate a DB parameter group with a DB instance, use the RDS API `ModifyDBInstance` operation with the following parameters:

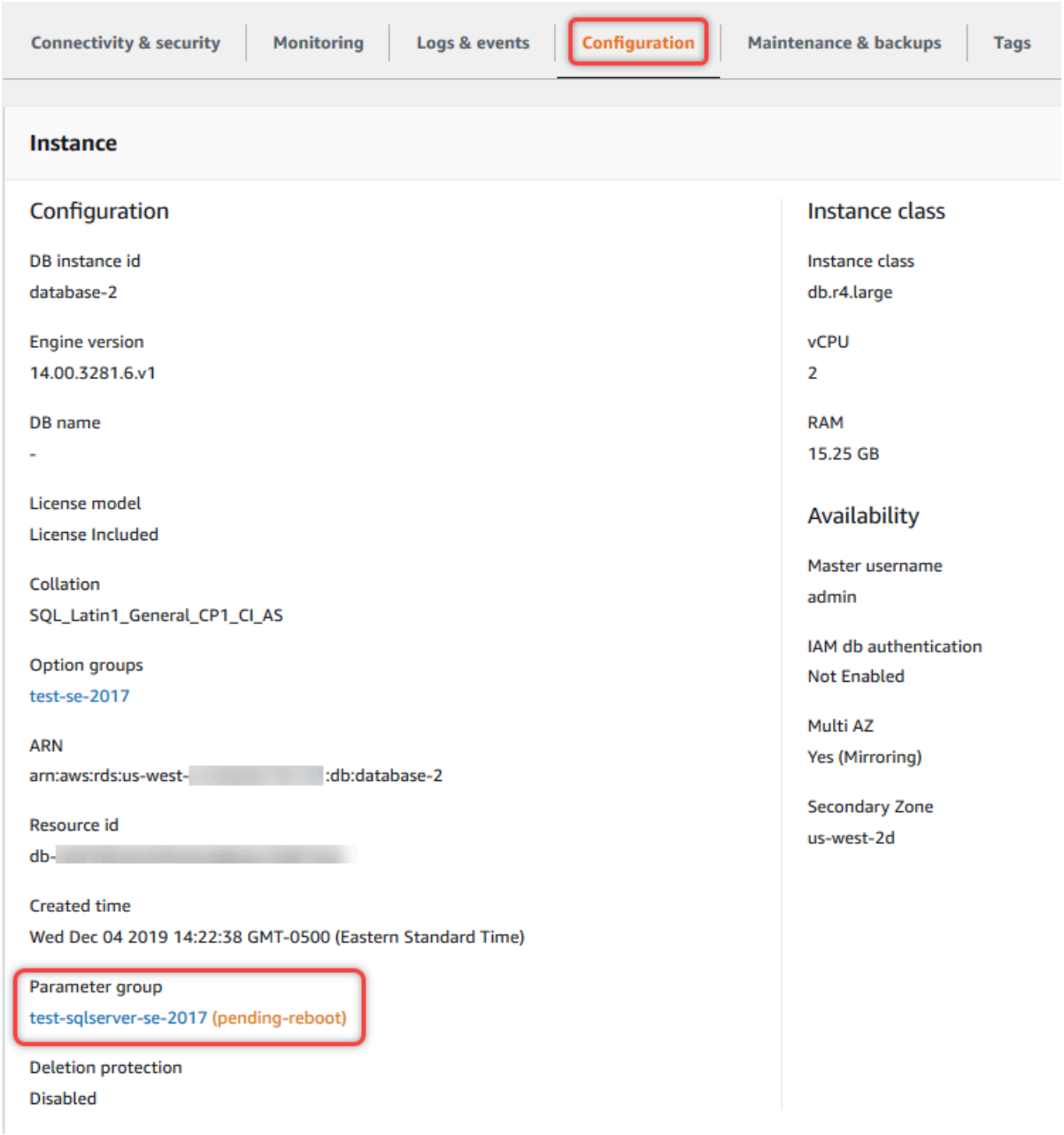
- `DBInstanceName`
- `DBParameterGroupName`

Modifying parameters in a DB parameter group in Amazon RDS

You can modify parameter values in a customer-created DB parameter group; you can't change the parameter values in a default DB parameter group. Changes to parameters in a customer-created DB parameter group are applied to all DB instances that are associated with the DB parameter group.

There are two types of parameters: dynamic parameters and static parameters. Changes to dynamic parameters are applied to the DB instance immediately without a reboot. Changes to static parameters are applied only after the DB instance is rebooted.

The RDS console shows the status of the DB parameter group associated with a DB instance on the **Configuration** tab. For example, if the DB instance isn't using the latest changes to its associated DB parameter group, the RDS console shows the DB parameter group with a status of **pending-reboot**. To apply the latest parameter changes to that DB instance, manually reboot the DB instance.



Console

To modify the parameters in a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the name of the parameter group that you want to modify.
4. For **Parameter group actions**, choose **Edit**.
5. Change the values of the parameters that you want to modify. You can scroll through the parameters using the arrow keys at the top right of the dialog box.

   You can't change values in a default parameter group.
6. Choose **Save changes**.

AWS CLI

To modify a DB parameter group, use the AWS CLI `modify-db-parameter-group` command with the following required options:

- `--db-parameter-group-name`
- `--parameters`

The following example modifies the `max_connections` and `max_allowed_packet` values in the DB parameter group named *mydbparametergroup*.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

The command produces output like the following:

```
DBPARAMETERGROUP  mydbparametergroup
```

RDS API

To modify a DB parameter group, use the RDS API ModifyDBParameterGroup operation with the following required parameters:

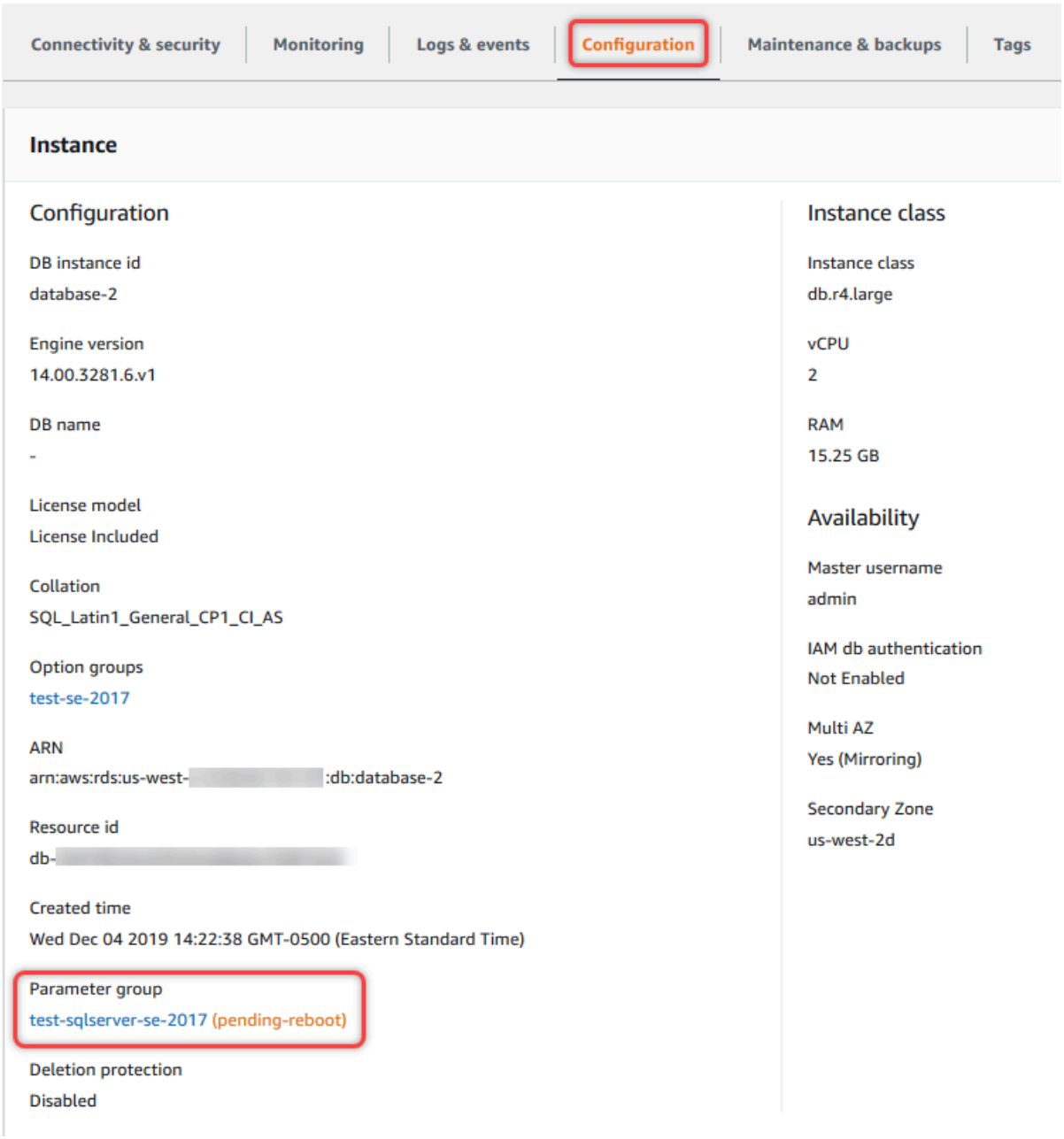
- DBParameterGroupName
- Parameters

Resetting parameters in a DB parameter group to their default values in Amazon RDS

You can reset parameter values in a customer-created DB parameter group to their default values. Changes to parameters in a customer-created DB parameter group are applied to all DB instances that are associated with the DB parameter group.

When you use the console, you can reset specific parameters to their default values. However, you can't easily reset all of the parameters in the DB parameter group at once. When you use the AWS CLI or RDS API, you can reset specific parameters to their default values. You can also reset all of the parameters in the DB parameter group at once.

Changes to some parameters are applied to the DB instance immediately without a reboot. Changes to other parameters are applied only after the DB instance is rebooted. The RDS console shows the status of the DB parameter group associated with a DB instance on the **Configuration** tab. For example, suppose that the DB instance isn't using the latest changes to its associated DB parameter group. If so, the RDS console shows the DB parameter group with a status of **pending-reboot**. To apply the latest parameter changes to that DB instance, manually reboot the DB instance.



Note

In a default DB parameter group, parameters are always set to their default values.

Console

To reset parameters in a DB parameter group to their default values

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the parameter group.
4. For **Parameter group actions**, choose **Edit**.
5. Choose the parameters that you want to reset to their default values. You can scroll through the parameters using the arrow keys at the top right of the dialog box.

   You can't reset values in a default parameter group.
6. Choose **Reset** and then confirm by choosing **Reset parameters**.

AWS CLI

To reset some or all of the parameters in a DB parameter group, use the AWS CLI `reset-db-parameter-group` command with the following required option: `--db-parameter-group-name`.

To reset all of the parameters in the DB parameter group, specify the `--reset-all-parameters` option. To reset specific parameters, specify the `--parameters` option.

The following example resets all of the parameters in the DB parameter group named *mydbparametergroup* to their default values.

Example

For Linux, macOS, or Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

For Windows:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
```

```
    --reset-all-parameters
```

The following example resets the `max_connections` and `max_allowed_packet` options to their default values in the DB parameter group named *mydbparametergroup*.

Example

For Linux, macOS, or Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" \
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

For Windows:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

The command produces output like the following:

```
DBParameterGroupName  mydbparametergroup
```

RDS API

To reset parameters in a DB parameter group to their default values, use the RDS API `ResetDBParameterGroup` command with the following required parameter: `DBParameterGroupName`.

To reset all of the parameters in the DB parameter group, set the `ResetAllParameters` parameter to `true`. To reset specific parameters, specify the `Parameters` parameter.

Copying a DB parameter group in Amazon RDS

You can copy custom DB parameter groups that you create. Copying a parameter group can be convenient solution. An example is when you have created a DB parameter group and want to include most of its custom parameters and values in a new DB parameter group. You can copy a DB parameter group by using the AWS Management Console. You can also use the AWS CLI copy-db-parameter-group command or the RDS API CopyDBParameterGroup operation.

After you copy a DB parameter group, wait at least 5 minutes before creating your first DB instance that uses that DB parameter group as the default parameter group. Doing this allows Amazon RDS to fully complete the copy action before the parameter group is used. This is especially important for parameters that are critical when creating the default database for a DB instance. An example is the character set for the default database defined by the `character_set_database` parameter. Use the **Parameter Groups** option of the Amazon RDS console or the describe-db-parameters command to verify that your DB parameter group is created.

> **Note**
>
> You can't copy a default parameter group. However, you can create a new parameter group that is based on a default parameter group.
> You can't copy a DB parameter group to a different AWS account or AWS Region.

Console

To copy a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the custom parameter group that you want to copy.
4. For **Parameter group actions**, choose **Copy**.
5. In **New DB parameter group identifier**, enter a name for the new parameter group.
6. In **Description**, enter a description for the new parameter group.
7. Choose **Copy**.

AWS CLI

To copy a DB parameter group, use the AWS CLI `copy-db-parameter-group` command with the following required options:

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

The following example creates a new DB parameter group named `mygroup2` that is a copy of the DB parameter group `mygroup1`.

Example

For Linux, macOS, or Unix:

```
aws rds copy-db-parameter-group \
    --source-db-parameter-group-identifier mygroup1 \
    --target-db-parameter-group-identifier mygroup2 \
    --target-db-parameter-group-description "DB parameter group 2"
```

For Windows:

```
aws rds copy-db-parameter-group ^
    --source-db-parameter-group-identifier mygroup1 ^
    --target-db-parameter-group-identifier mygroup2 ^
    --target-db-parameter-group-description "DB parameter group 2"
```

RDS API

To copy a DB parameter group, use the RDS API `CopyDBParameterGroup` operation with the following required parameters:

- `SourceDBParameterGroupIdentifier`
- `TargetDBParameterGroupIdentifier`
- `TargetDBParameterGroupDescription`

Listing DB parameter groups in Amazon RDS

You can list the DB parameter groups you've created for your AWS account.

> **Note**
>
> Default parameter groups are automatically created from a default parameter template when you create a DB instance for a particular DB engine and version. These default parameter groups contain preferred parameter settings and can't be modified. When you create a custom parameter group, you can modify parameter settings.

Console

To list all DB parameter groups for an AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.

   The DB parameter groups appear in a list.

AWS CLI

To list all DB parameter groups for an AWS account, use the AWS CLI `describe-db-parameter-groups` command.

Example

The following example lists all available DB parameter groups for an AWS account.

```
aws rds describe-db-parameter-groups
```

The command returns a response like the following:

```
DBPARAMETERGROUP  default.mysql8.0     mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP  mydbparametergroup   mysql8.0  My new parameter group
```

The following example describes the *mydbparamgroup1* parameter group.

For Linux, macOS, or Unix:

```
aws rds describe-db-parameter-groups \
    --db-parameter-group-name mydbparamgroup1
```

For Windows:

```
aws rds describe-db-parameter-groups ^
    --db-parameter-group-name mydbparamgroup1
```

The command returns a response like the following:

```
DBPARAMETERGROUP  mydbparametergroup1  mysql8.0  My new parameter group
```

RDS API

To list all DB parameter groups for an AWS account, use the RDS API `DescribeDBParameterGroups` operation.

Viewing parameter values for a DB parameter group in Amazon RDS

You can get a list of all parameters in a DB parameter group and their values.

Console

To view the parameter values for a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.

   The DB parameter groups appear in a list.
3. Choose the name of the parameter group to see its list of parameters.

AWS CLI

To view the parameter values for a DB parameter group, use the AWS CLI `describe-db-parameters` command with the following required parameter.

- `--db-parameter-group-name`

Example

The following example lists the parameters and parameter values for a DB parameter group named *mydbparametergroup*.

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

The command returns a response like the following:

```
DBPARAMETER  Parameter Name           Parameter Value  Source          Data Type
 Apply Type  Is Modifiable
```

```
DBPARAMETER  allow-suspicious-udfs                    engine-default   boolean
 static      false
DBPARAMETER  auto_increment_increment                 engine-default   integer
 dynamic     true
DBPARAMETER  auto_increment_offset                    engine-default   integer
 dynamic     true
DBPARAMETER  binlog_cache_size         32768          system           integer
 dynamic     true
DBPARAMETER  socket                    /tmp/mysql.sock  system         string
 static      false
```

RDS API

To view the parameter values for a DB parameter group, use the RDS API `DescribeDBParameters` command with the following required parameter.

- `DBParameterGroupName`

Deleting a DB parameter group in Amazon RDS

You can delete a DB parameter group using the AWS Management Console, AWS CLI, or RDS API. A parameter group is eligible for deletion only if it isn't associated with a DB instance.

Console

To delete a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.

   The DB parameter groups appear in a list.
3. Choose the name of the parameter groups to be deleted.
4. Choose **Actions** and then **Delete**.
5. Review the parameter group names and then choose **Delete**.

AWS CLI

To delete a DB parameter group, use the AWS CLI `delete-db-parameter-group` command with the following required parameter.

- `--db-parameter-group-name`

Example

The following example deletes a DB parameter group named *mydbparametergroup.*

```
aws rds delete-db-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

To delete a DB parameter group, use the RDS API `DeleteDBParameterGroup` command with the following required parameter.

- `DBParameterGroupName`

Working with DB cluster parameter groups for Multi-AZ DB clusters

Multi-AZ DB clusters use DB cluster parameter groups. The following sections describe configuring and managing DB cluster parameter groups.

Topics

- Creating a DB cluster parameter group
- Modifying parameters in a DB cluster parameter group
- Resetting parameters in a DB cluster parameter group
- Copying a DB cluster parameter group
- Listing DB cluster parameter groups
- Viewing parameter values for a DB cluster parameter group
- Deleting a DB cluster parameter group

Creating a DB cluster parameter group

You can create a new DB cluster parameter group using the AWS Management Console, the AWS CLI, or the RDS API.

After you create a DB cluster parameter group, wait at least 5 minutes before creating a DB cluster that uses that DB cluster parameter group. Doing this allows Amazon RDS to fully create the

parameter group before it is used by the new DB cluster. You can use the **Parameter groups** page in the Amazon RDS console or the describe-db-cluster-parameters command to verify that your DB cluster parameter group is created.

The following limitations apply to the DB cluster parameter group name:

- The name must be 1 to 255 letters, numbers, or hyphens.

  Default parameter group names can include a period, such as `default.mysql5.7`. However, custom parameter group names can't include a period.
- The first character must be a letter.
- The name can't end with a hyphen or contain two consecutive hyphens.

Console

To create a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
4. For **Parameter group name**, enter the name of the new DB cluster parameter group.
5. For **Description**, enter a description for the new DB cluster parameter group.
6. For **Engine type**, choose your database engine.
7. For **Parameter group family**, choose a DB parameter group family.
8. Choose **Create**.

AWS CLI

To create a DB cluster parameter group, use the AWS CLI `create-db-cluster-parameter-group` command.

The following example creates a DB cluster parameter group named *mydbclusterparametergroup* for RDS for MySQL version 8.0 with a description of "*My new cluster parameter group*."

Include the following required parameters:

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

To list all of the available parameter group families, use the following command:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> The output contains duplicates.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --db-parameter-group-family mysql8.0 \
    --description "My new cluster parameter group"
```

For Windows:

```
aws rds create-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --db-parameter-group-family mysql8.0 ^
    --description "My new cluster parameter group"
```

This command produces output similar to the following:

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "mysql8.0",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup2"
    }
```

```
}
```

RDS API

To create a DB cluster parameter group, use the RDS API `CreateDBClusterParameterGroup` action.

Include the following required parameters:

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Modifying parameters in a DB cluster parameter group

You can modify parameter values in a customer-created DB cluster parameter group. You can't change the parameter values in a default DB cluster parameter group. Changes to parameters in a customer-created DB cluster parameter group are applied to all DB clusters that are associated with the DB cluster parameter group.

Console

To modify a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the parameter group that you want to modify.
4. For **Parameter group actions**, choose **Edit**.
5. Change the values of the parameters you want to modify. You can scroll through the parameters using the arrow keys at the top right of the dialog box.

   You can't change values in a default parameter group.
6. Choose **Save changes**.
7. Reboot the cluster to apply the changes to it.

   If you don't reboot the cluster, then a failover operation could take longer than normal.

AWS CLI

To modify a DB cluster parameter group, use the AWS CLI `modify-db-cluster-parameter-group` command with the following required parameters:

- `--db-cluster-parameter-group-name`
- `--parameters`

The following example modifies the `server_audit_logging` and `server_audit_logs_upload` values in the DB cluster parameter group named *mydbclusterparametergroup*.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

The command produces output like the following:

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

RDS API

To modify a DB cluster parameter group, use the RDS API `ModifyDBClusterParameterGroup` command with the following required parameters:

- `DBClusterParameterGroupName`
- `Parameters`

Resetting parameters in a DB cluster parameter group

You can reset parameters to their default values in a customer-created DB cluster parameter group. Changes to parameters in a customer-created DB cluster parameter group are applied to all DB clusters that are associated with the DB cluster parameter group.

> **Note**
>
> In a default DB cluster parameter group, parameters are always set to their default values.

Console

To reset parameters in a DB cluster parameter group to their default values

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the parameter group.
4. For **Parameter group actions**, choose **Edit**.
5. Choose the parameters that you want to reset to their default values. You can scroll through the parameters using the arrow keys at the top right of the dialog box.

   You can't reset values in a default parameter group.
6. Choose **Reset** and then confirm by choosing **Reset parameters**.
7. Reboot the DB cluster.

AWS CLI

To reset parameters in a DB cluster parameter group to their default values, use the AWS CLI `reset-db-cluster-parameter-group` command with the following required option: `--db-cluster-parameter-group-name`.

To reset all of the parameters in the DB cluster parameter group, specify the `--reset-all-parameters` option. To reset specific parameters, specify the `--parameters` option.

The following example resets all of the parameters in the DB parameter group named *mydbparametergroup* to their default values.

Example

For Linux, macOS, or Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

For Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

The following example resets the `server_audit_logging` and `server_audit_logs_upload` to their default values in the DB cluster parameter group named *mydbclusterparametergroup*.

Example

For Linux, macOS, or Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
                 "ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

For Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

The command produces output like the following:

```
DBClusterParameterGroupName  mydbclusterparametergroup
```

RDS API

To reset parameters in a DB cluster parameter group to their default values, use the RDS API `ResetDBClusterParameterGroup` command with the following required parameter: `DBClusterParameterGroupName`.

To reset all of the parameters in the DB cluster parameter group, set the `ResetAllParameters` parameter to `true`. To reset specific parameters, specify the `Parameters` parameter.

Copying a DB cluster parameter group

You can copy custom DB cluster parameter groups that you create. Copying a parameter group is a convenient solution when you have already created a DB cluster parameter group and you want to include most of the custom parameters and values from that group in a new DB cluster parameter group. You can copy a DB cluster parameter group by using the AWS CLI copy-db-cluster-parameter-group command or the RDS API CopyDBClusterParameterGroup operation.

After you copy a DB cluster parameter group, wait at least 5 minutes before creating a DB cluster that uses that DB cluster parameter group. Doing this allows Amazon RDS to fully copy the parameter group before it is used by the new DB cluster. You can use the **Parameter groups** page in the Amazon RDS console or the describe-db-cluster-parameters command to verify that your DB cluster parameter group is created.

> **Note**
>
> You can't copy a default parameter group. However, you can create a new parameter group that is based on a default parameter group.
> You can't copy a DB cluster parameter group to a different AWS account or AWS Region.

Console

To copy a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.

3. In the list, choose the custom parameter group that you want to copy.
4. For **Parameter group actions**, choose **Copy**.
5. In **New DB parameter group identifier**, enter a name for the new parameter group.
6. In **Description**, enter a description for the new parameter group.
7. Choose **Copy**.

AWS CLI

To copy a DB cluster parameter group, use the AWS CLI `copy-db-cluster-parameter-group` command with the following required parameters:

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

The following example creates a new DB cluster parameter group named `mygroup2` that is a copy of the DB cluster parameter group `mygroup1`.

Example

For Linux, macOS, or Unix:

```
aws rds copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier mygroup1 \
    --target-db-cluster-parameter-group-identifier mygroup2 \
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

For Windows:

```
aws rds copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier mygroup1 ^
    --target-db-cluster-parameter-group-identifier mygroup2 ^
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

RDS API

To copy a DB cluster parameter group, use the RDS API `CopyDBClusterParameterGroup` operation with the following required parameters:

- `SourceDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupDescription`

Listing DB cluster parameter groups

You can list the DB cluster parameter groups you've created for your AWS account.

> **Note**
>
> Default parameter groups are automatically created from a default parameter template when you create a DB cluster for a particular DB engine and version. These default parameter groups contain preferred parameter settings and can't be modified. When you create a custom parameter group, you can modify parameter settings.

Console

To list all DB cluster parameter groups for an AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.

   The DB cluster parameter groups appear in the list with **DB cluster parameter group** for **Type**.

AWS CLI

To list all DB cluster parameter groups for an AWS account, use the AWS CLI `describe-db-cluster-parameter-groups` command.

Example

The following example lists all available DB cluster parameter groups for an AWS account.

```
aws rds describe-db-cluster-parameter-groups
```

The following example describes the *mydbclusterparametergroup* parameter group.

For Linux, macOS, or Unix:

```
aws rds describe-db-cluster-parameter-groups \
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

For Windows:

```
aws rds describe-db-cluster-parameter-groups ^
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

The command returns a response like the following:

```
{
    "DBClusterParameterGroups": [
        {
            "DBClusterParameterGroupName": "mydbclusterparametergroup2",
            "DBParameterGroupFamily": "mysql8.0",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
}
```

RDS API

To list all DB cluster parameter groups for an AWS account, use the RDS API `DescribeDBClusterParameterGroups` action.

Viewing parameter values for a DB cluster parameter group

You can get a list of all parameters in a DB cluster parameter group and their values.

Console

To view the parameter values for a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.

The DB cluster parameter groups appear in the list with **DB cluster parameter group** for **Type**.

3. Choose the name of the DB cluster parameter group to see its list of parameters.

AWS CLI

To view the parameter values for a DB cluster parameter group, use the AWS CLI `describe-db-cluster-parameters` command with the following required parameter.

- `--db-cluster-parameter-group-name`

Example

The following example lists the parameters and parameter values for a DB cluster parameter group named *mydbclusterparametergroup*, in JSON format.

The command returns a response like the following:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-
name mydbclusterparametergroup
```

```
{
    "Parameters": [
        {
            "ParameterName": "activate_all_roles_on_login",
            "ParameterValue": "0",
            "Description": "Automatically set all granted roles as active after the
 user has authenticated successfully.",
            "Source": "engine-default",
            "ApplyType": "dynamic",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
        {
            "ParameterName": "allow-suspicious-udfs",
```

```
            "Description": "Controls whether user-defined functions that have only an
 xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
...
```

RDS API

To view the parameter values for a DB cluster parameter group, use the RDS API `DescribeDBClusterParameters` command with the following required parameter.

- `DBClusterParameterGroupName`

In some cases, the allowed values for a parameter aren't shown. These are always parameters where the source is the database engine default.

To view the values of these parameters, you can run the following SQL statements:

- MySQL:

```
-- Show the value of a particular parameter
mysql$ SHOW VARIABLES LIKE '%parameter_name%';

-- Show the values of all parameters
mysql$ SHOW VARIABLES;
```

- PostgreSQL:

```
-- Show the value of a particular parameter
postgresql=> SHOW parameter_name;

-- Show the values of all parameters
postgresql=> SHOW ALL;
```

Deleting a DB cluster parameter group

You can delete a DB cluster parameter group using the AWS Management Console, AWS CLI, or RDS API. A DB cluster parameter group parameter group is eligible for deletion only if it isn't associated with a DB cluster.

Console

To delete parameter groups

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.

   The parameter groups appear in a list.
3. Choose the name of the DB cluster parameter groups to be deleted.
4. Choose **Actions** and then **Delete**.
5. Review the parameter group names and then choose **Delete**.

AWS CLI

To delete a DB cluster parameter group, use the AWS CLI `delete-db-cluster-parameter-group` command with the following required parameter.

- `--db-parameter-group-name`

Example

The following example deletes a DB cluster parameter group named *mydbparametergroup.*

```
aws rds delete-db-cluster-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

To delete a DB cluster parameter group, use the RDS API `DeleteDBClusterParameterGroup` command with the following required parameter.

- `DBParameterGroupName`

Comparing DB parameter groups

You can use the AWS Management Console to view the differences between two DB parameter groups.

The specified parameter groups must both be DB parameter groups, or they both must be DB cluster parameter groups. This is true even when the DB engine and version are the same. For example, you can't compare an `aurora-mysql8.0` (Aurora MySQL version 3) DB parameter group and an `aurora-mysql8.0` DB cluster parameter group.

You can compare Aurora MySQL and RDS for MySQL DB parameter groups, even for different versions, but you can't compare Aurora PostgreSQL and RDS for PostgreSQL DB parameter groups.

To compare two DB parameter groups

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the two parameter groups that you want to compare.

   > **Note**
   >
   > To compare a default parameter group to a custom parameter group, first choose the default parameter group on the **Default** tab, then choose the custom parameter group on the **Custom** tab.

4. From **Actions**, choose **Compare**.

Specifying DB parameters

DB parameter types include the following:

- Integer
- Boolean
- String
- Long
- Double

- Timestamp
- Object of other defined data types
- Array of values of type integer, Boolean, string, long, double, timestamp, or object

You can also specify integer and Boolean parameters using expressions, formulas, and functions.

For the Oracle engine, you can use the `DBInstanceClassHugePagesDefault` formula variable to specify a Boolean DB parameter. See DB parameter formula variables.

For the PostgreSQL engine, you can use an expression to specify a Boolean DB parameter. See Boolean DB parameter expressions.

Contents

- DB parameter formulas
  - DB parameter formula variables
  - DB parameter formula operators
- DB parameter functions
- Boolean DB parameter expressions
- DB parameter log expressions
- DB parameter value examples

DB parameter formulas

A DB parameter formula is an expression that resolves to an integer value or a Boolean value. You enclose the expression in braces: {}. You can use a formula for either a DB parameter value or as an argument to a DB parameter function.

Syntax

```
{FormulaVariable}
{FormulaVariable*Integer}
{FormulaVariable*Integer/Integer}
{FormulaVariable/Integer}
```

DB parameter formula variables

Each formula variable returns an integer or a Boolean value. The names of the variables are case-sensitive.

AllocatedStorage

Returns an integer representing the size, in bytes, of the data volume.

DBInstanceClassHugePagesDefault

Returns a Boolean value. Currently, it's only supported for Oracle engines.

For more information, see Turning on HugePages for an RDS for Oracle instance.

DBInstanceClassMemory

Returns an integer for the number of bytes of memory available to the database process. This number is internally calculated by starting with the total amount of memory for the DB instance class. From this, the calculation subtracts memory reserved for the operating system and the RDS processes that manage the instance. Therefore, the number is always somewhat lower than the memory figures shown in the instance class tables in DB instance classes. The exact value depends on a combination of factors. These include instance class, DB engine, and whether it applies to an RDS instance or an instance that's part of an Aurora cluster.

DBInstanceVCPU

Returns an integer representing the number of virtual central processing units (vCPUs) used by Amazon RDS to manage the instance.

EndPointPort

Returns an integer representing the port used when connecting to the DB instance.

TrueIfReplica

Returns `1` if the DB instance is a read replica and `0` if it is not. This is the default value for the `read_only` parameter in MySQL.

DB parameter formula operators

DB parameter formulas support two operators: division and multiplication.

Division operator: /

Divides the dividend by the divisor, returning an integer quotient. Decimals in the quotient are truncated, not rounded.

Syntax

```
dividend / divisor
```

The dividend and divisor arguments must be integer expressions.

*Multiplication operator: **

Multiplies the expressions, returning the product of the expressions. Decimals in the expressions are truncated, not rounded.

Syntax

```
expression * expression
```

Both expressions must be integers.

DB parameter functions

You specify the arguments of DB parameter functions as either integers or formulas. Each function must have at least one argument. Specify multiple arguments as a comma-separated list. The list can't have any empty members, such as *argument1*,,*argument3*. Function names are case-insensitive.

IF

Returns an argument.

Currently, it's only supported for Oracle engines, and the only supported first argument is `{DBInstanceClassHugePagesDefault}`. For more information, see Turning on HugePages for an RDS for Oracle instance.

Syntax

```
IF(argument1, argument2, argument3)
```

Returns the second argument if the first argument evaluates to true. Returns the third argument otherwise.

GREATEST

Returns the largest value from a list of integers or parameter formulas.

Syntax

```
GREATEST(argument1, argument2,...argumentn)
```

Returns an integer.

LEAST

Returns the smallest value from a list of integers or parameter formulas.

Syntax

```
LEAST(argument1, argument2,...argumentn)
```

Returns an integer.

SUM

Adds the values of the specified integers or parameter formulas.

Syntax

```
SUM(argument1, argument2,...argumentn)
```

Returns an integer.

Boolean DB parameter expressions

A Boolean DB parameter expression resolves to a Boolean value of 1 or 0. The expression is enclosed in quotation marks.

> **Note**
>
> Boolean DB parameter expressions are only supported for the PostgreSQL engine.

Syntax

```
"expression operator expression"
```

Both expressions must resolve to integers. An expression can be the following:

- integer constant
- DB parameter formula
- DB parameter function
- DB parameter variable

Boolean DB parameter expressions support the following inequality operators:

The greater than operator: >

Syntax

```
"expression > expression"
```

The less than operator: <

Syntax

```
"expression < expression"
```

The greater than or equal to operators: >=, =>

Syntax

```
"expression >= expression"
"expression => expression"
```

The less than or equal to operators: <=, =<

Syntax

```
"expression <= expression"
"expression =< expression"
```

Example using a Boolean DB parameter expression

The following Boolean DB parameter expression example compares the result of a parameter formula with an integer. It does so to modify the Boolean DB parameter `wal_compression` for a PostgreSQL DB instance. The parameter expression compares the number of vCPUs with the value 2. If the number of vCPUs is greater than 2, then the `wal_compression` DB parameter is set to true.

```
aws rds modify-db-parameter-group --db-parameter-group-name group-name \
--parameters "ParameterName=wal_compression,ParameterValue=\"{DBInstanceVCPU} > 2\" "
```

DB parameter log expressions

You can set an integer DB parameter value to a log expression. You enclose the expression in braces: {}. For example:

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

The `log` function represents log base 2. This example also uses the `DBInstanceClassMemory` formula variable. See DB parameter formula variables.

> **Note**
>
> Currently, you can't specify the MySQL `innodb_log_file_size` parameter with any value other than an integer.

DB parameter value examples

These examples show using formulas, functions, and expressions for the values of DB parameters.

> **Warning**
>
> Improperly setting parameters in a DB parameter group can have unintended adverse effects. These might include degraded performance and system instability. Use caution when modifying database parameters and back up your data before modifying your DB parameter group. Try out parameter group changes on a test DB instance, created using point-in-time-restores, before applying those parameter group changes to your production DB instances.

Example using the DB parameter function GREATEST

You can specify the GREATEST function in an Oracle processes parameter. Use it to set the number of user processes to the larger of either 80 or DBInstanceClassMemory divided by 9,868,951.

```
GREATEST({DBInstanceClassMemory/9868951},80)
```

Example using the DB parameter function LEAST

You can specify the LEAST function in a MySQL max_binlog_cache_size parameter value. Use it to set the maximum cache size a transaction can use in a MySQL instance to the lesser of 1 MB or DBInstanceClass/256.

```
LEAST({DBInstanceClassMemory/256},10485760)
```

Creating an Amazon ElastiCache cache using Amazon RDS DB instance settings

ElastiCache is a fully managed, in-memory caching service that provides microsecond read and write latencies that support flexible, real-time use cases. ElastiCache can help you accelerate application and database performance. You can use ElastiCache as a primary data store for use cases that don't require data durability, such as gaming leaderboards, streaming, and data analytics. ElastiCache helps remove the complexity associated with deploying and managing a distributed computing environment. For more information, see Common ElastiCache Use Cases and How ElastiCache Can Help for Memcached and Common ElastiCache Use Cases and How ElastiCache Can Help for Redis OSS. You can use the Amazon RDS console for creating ElastiCache cache.

You can operate Amazon ElastiCache in two formats. You can get started with a serverless cache or choose to design your own cache cluster. If you choose to design your own cache cluster, ElastiCache works with both the Redis OSS and Memcached engines. If you're unsure which engine you want to use, see Comparing Memcached and Redis OSS. For more information about Amazon ElastiCache, see the Amazon ElastiCache User Guide.

Topics

- Overview of ElastiCache cache creation with RDS DB instance settings
- Creating an ElastiCache cache with settings from an RDS DB instance

Overview of ElastiCache cache creation with RDS DB instance settings

You can create an ElastiCache cache from Amazon RDS using the same configuration settings as a newly created or existing RDS DB instance.

Some use cases to associate an ElastiCache cache with your DB instance:

- You can save costs and improve your performance by using ElastiCache with RDS versus running on RDS alone.

  For example, you can save up to 55% in cost and gain up to 80x faster read performance by using ElastiCache with RDS for MySQL versus RDS for MySQL alone.

- You can use the ElastiCache cache as a primary data store for applications that don't require data durability. Your applications that use Redis OSS or Memcached can use ElastiCache with almost no modification.

When you create an ElastiCache cache from RDS, the ElastiCache cache inherits the following settings from the associated RDS DB instance:

- ElastiCache connectivity settings
- ElastiCache security settings

You can also set the cache configuration settings according to your requirements.

Setting up ElastiCache in your applications

Your applications must be set up to utilize ElastiCache cache. You can also optimize and improve cache performance by setting up your applications to use caching strategies depending on your requirements.

- To access your ElastiCache cache and get started, see Getting started with ElastiCache (Redis OSS) and Getting started with ElastiCache (Memcached).
- For more information about caching strategies, see Caching strategies and best practices for Memcached and Caching strategies and best practices for Redis OSS.
- For more information about high availability in ElastiCache (Redis OSS) clusters, see High availability using replication groups.
- You might incur costs associated with backup storage, data transfer within or across regions, or use of AWS Outposts. For pricing details, see Amazon ElastiCache pricing.

Creating an ElastiCache cache with settings from an RDS DB instance

You can create an ElastiCache cache for your RDS DB instances with settings for inherited from the DB instance.

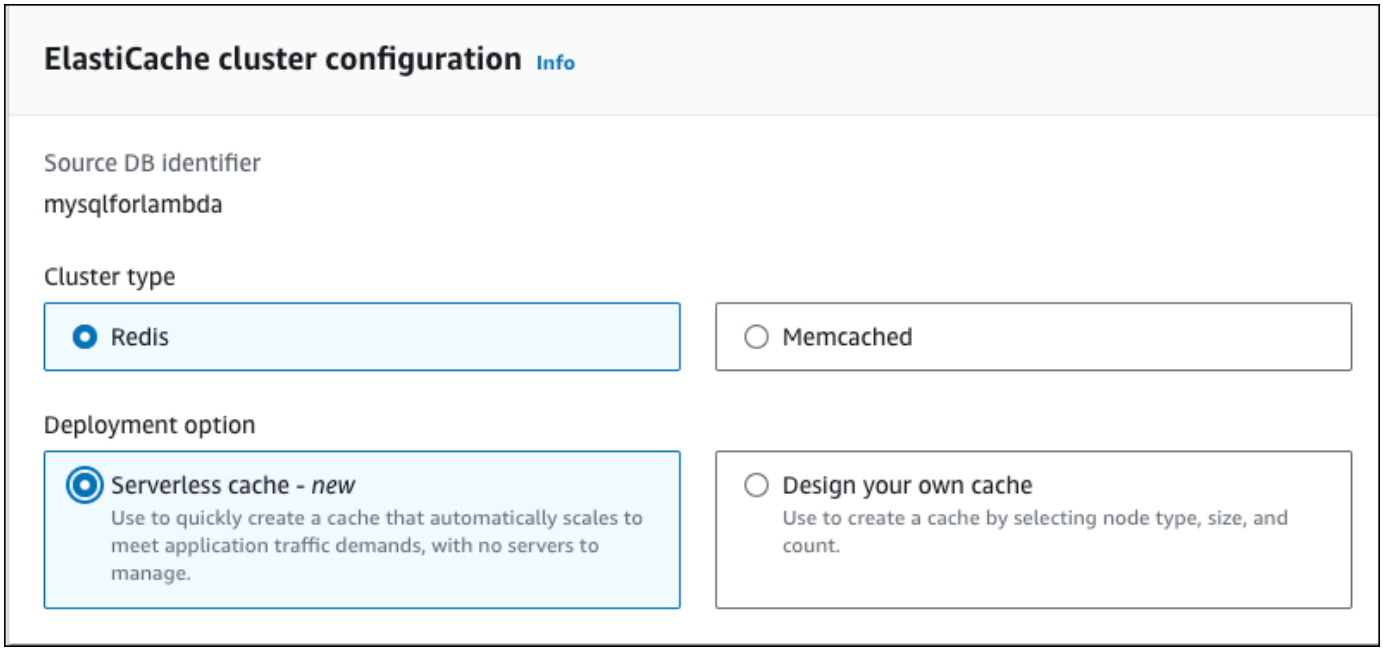
Create an ElastiCache cache with settings from a DB instance

1. To create a DB instance, follow the instructions in Creating an Amazon RDS DB instance.
2. After creating an RDS DB instance, the console displays the **Suggested add-ons** window. Select **Create an ElastiCache cluster from RDS using your DB settings**.

For an existing database, in the **Databases** page, select the required DB instance. In the **Actions** dropdown menu, choose **Create ElastiCache cluster** to create an ElastiCache cache in RDS that has the same settings as your existing RDS DB instance.

In the **ElastiCache configuration section**, the **Source DB identifier** displays which DB instance the ElastiCache cache inherits settings from.

3. Choose whether you want to create a Redis OSS or Memcached cluster. For more information, see Comparing Memcached and Redis OSS.



4. After this, choose whether you want to create a **Serverless cache** or **Design your own cache**. For more information, see Choosing between deployment options.

   If you choose **Serverless cache**:

   a. In **Cache settings**, enter values for **Name** and **Description**.

   b. Under **View default settings**, leave the default settings to establish the connection between your cache and DB instance.

   c. You can also edit the default settings by choosing **Customize default settings**. Select the **ElastiCache connectivity settings**, **ElastiCache security settings**, and **Maximum usage limits**.

5. If you choose **Design your own cache**:

a. If you chose **Redis OSS cluster**, choose whether you want to keep the cluster mode **Enabled** or **Disabled**. For more information, see Replication: Redis OSS (Cluster Mode Disabled) vs. Redis OSS (Cluster Mode Enabled).

b. Enter values for **Name**, **Description**, and **Engine version**.

   For **Engine version**, the recommended default value is the latest engine version. You can also choose an **Engine version** for the ElastiCache cache that best meets your requirements.

c. Choose the node type in the **Node type** option. For more information, see Managing nodes.

   If you choose to create a Redis OSS cluster with the **Cluster mode** set to **Enabled**, then enter the number of shards (partitions/node groups) in the **Number of shards** option.

   Enter the number of replicas of each shard in **Number of replicas**.

   > **Note**
   >
   > The selected node type, the number of shards, and the number of replicas all affect your cache performance and resource costs. Be sure these settings match your database needs. For pricing information, see Amazon ElastiCache pricing.

d. Select the **ElastiCache connectivity settings** and **ElastiCache security settings**. You can keep the default settings or customize these settings per your requirements.

6. Verify the default and inherited settings of your ElastiCache cache. Some settings can't be changed after creation.

   > **Note**
   >
   > RDS might adjust the backup window of your ElastiCache cache to meet the minimum window requirement of 60 minutes. The backup window of your source database remains the same.

7. When you're ready, choose **Create ElastiCache cache**.

The console displays a confirmation banner for the ElastiCache cache creation. Follow the link in the banner to the ElastiCache console to view the cache details. The ElastiCache console displays the newly created ElastiCache cache.

Auto migrating EC2 databases to Amazon RDS using AWS Database Migration Service

You can use the RDS console to migrate an EC2 database to RDS. RDS uses AWS Database Migration Service (AWS DMS) to migrate your source EC2 database. AWS DMS allows you to migrate relational databases into your AWS Cloud. For more information about AWS Database Migration Service, see What is AWS Database Migration Service? in the *AWS Database Migration Service User Guide*.

To begin the migration, you must create an equivalent RDS DB instance to migrate the data into. After you create your target database, you can import your EC2 database into it. For source databases smaller than 1TiB, this migration action reduces the time and resources required to migrate your data into RDS.

Overview

The RDS console allows you to migrate EC2 databases into equivalent RDS databases. You must create an RDS database to enable migration from the console.

You can migrate EC2 databases for the following databases engines:

- MySQL
- MariaDB
- PostgreSQL

The migration process involves the following steps:

- Create an equivalent database in RDS. For the databases to be equivalent, they must have the same database engine and compatible engine versions. They must also be in the same VPC. For instructions on creating your database, see Creating an Amazon RDS DB instance.
- Choose the type of replication for your database:
  - **Full load migration** – RDS copies the complete source database to the target database, creating new tables in the target when necessary.

    > **Note**
    >
    > This option causes an outage in your RDS database.

- **Full load and change data capture (CDC) migration** – Similar to full load migration, with this option, RDS copies over the complete source database to the target database. However, after the full load migration, RDS applies any captured changes in the source to the target database. Change data capture collects changes to the database logs by using the database engine's native API.

  > **Note**
  >
  > This option causes an outage in your RDS database.

- **Change data capture (CDC)** – Use this option to keep your target database available through the migration. RDS migrates ongoing changes in your source database to the target database.

- RDS creates the necessary networking resources to facilitate the migration. Once RDS creates the required resources, it notifies you about the resources created and allows you to initiate the data transfer.

  The time required to complete the migration depends on the type of replication and the size of the source database.

Prerequisites

MySQL and MariaDB

Before you begin to work with a MySQL or MariaDB database as the source database, make sure that you have the following prerequisites. These prerequisites apply to AWS-managed sources.

You must have an account for AWS DMS that has the Replication Admin role. The role needs the following privileges:

- **REPLICATION CLIENT** – This privilege is required for CDC tasks only. In other words, full-load-only tasks don't require this privilege.
- **REPLICATION SLAVE** – This privilege is required for CDC tasks only. In other words, full-load-only tasks don't require this privilege.

The AWS DMS user must also have SELECT privileges for the source tables designated for replication.

Grant the following privileges if you use MySQL-specific premigration assessments.

```
grant select on mysql.user to <dms_user>;
grant select on mysql.db to <dms_user>;
grant select on mysql.tables_priv to <dms_user>;
grant select on mysql.role_edges to <dms_user>  #only for MySQL version 8.0.11 and
 higher
```

PostgreSQL

Before migrating data from an AWS-managed PostgreSQL source database, do the following:

- We recommend that you use an AWS user account with the minimum required permissions for the PostgreSQL DB instance as the user account for the PostgreSQL source endpoint for AWS DMS. Using the master account is not recommended. The account must have the `rds_superuser` role and the `rds_replication` role. The `rds_replication` role grants permissions to manage logical slots and to stream data using logical slots.

> **Note**
>
> Some AWS DMS transactions are idle for some time before the DMS engine uses them again. By using the parameter `idle_in_transaction_session_timeout` in PostgreSQL versions 9.6 and higher, you can cause idle transactions to time out and fail.

Limitations

The following limitations apply to the auto-migrate process:

- Your target database status must be **Available** to begin source database migration.
- When migrating from a MySQL source database, your RDS account must have the Replication Admin role. You must also have the proper privileges applied for that role.
- Your EC2 instance and target database must be in the same VPC.
- You can't migrate your EC2 database to the following target databases when using the **Migrate data from EC2 database** action:
  - Database that is a member of a cluster
  - Oracle, SQL Server, and Db2 databases
  - Databases with MySQL version lower than 5.7

- Databases with PostgreSQL version lower than 10.4
- Databases with MariaDB version lower than 10.2

Creating IAM resources for homogeneous migrations

RDS uses AWS DMS to migrate your data. To access your databases and to migrate data, AWS DMS creates a serverless environment for homogeneous data migrations. In this environment, AWS DMS requires access to VPC peering, route tables, security groups, and other AWS resources. Also, AWS DMS stores logs, metrics, and progress for each data migration in Amazon CloudWatch. To create a data migration project, AWS DMS needs access to these services.

Also, AWS DMS requires access to the secrets that respresent a set of user credentials to authenticate the database connection for the source and target connection.

> **Note**
>
> By using the **Migrate data from EC2 instance** action, you can use the RDS console to generate these IAM resources. Skip this step if you use the console generated IAM resources.

You need the following IAM resources for this process:

Topics

- Creating an IAM policy for homogeneous data migrations
- Creating an IAM role for homogeneous data migrations
- Creating a secret access policy and role
- Creating an IAM role for AWS DMS to manage Amazon VPC

Creating an IAM policy for homogeneous data migrations

In this step, you create an IAM policy that provides AWS DMS with access to Amazon EC2 and CloudWatch resources. Next, create an IAM role and attach this policy.

To create an IAM policy for data migration

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. In the **Create policy** page, choose the **JSON** tab.
5. Paste the following JSON into the editor.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeRouteTables",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeVpcs",
                "ec2:DescribePrefixLists",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "servicequotas:GetServiceQuota"
            ],
            "Resource": "arn:aws:servicequotas:*:*:vpc/L-0EA8095F"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:dms-data-migration-*:log-
stream:dms-data-migration-*"
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateRoute",
                "ec2:DeleteRoute"
            ],
            "Resource": "arn:aws:ec2:*:*:route-table/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:security-group-rule/*",
                "arn:aws:ec2:*:*:route-table/*",
                "arn:aws:ec2:*:*:vpc-peering-connection/*",
                "arn:aws:ec2:*:*:vpc/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group-rule/*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress"
            ],
            "Resource": "arn:aws:ec2:*:*:security-group/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AcceptVpcPeeringConnection",
                "ec2:ModifyVpcPeeringConnectionOptions"
            ],
            "Resource": "arn:aws:ec2:*:*:vpc-peering-connection/*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:AcceptVpcPeeringConnection",
            "Resource": "arn:aws:ec2:*:*:vpc/*"
        }
    ]
}
```

6. Choose **Next: Tags** and **Next: Review.**
7. Enter **`HomogeneousDataMigrationsPolicy`** for **Name***, and choose **Create policy**.

Creating an IAM role for homogeneous data migrations

In this step, you create an IAM role that provides access to AWS Secrets Manager, Amazon EC2, and CloudWatch.

To create an IAM role for data migrations

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. On the **Select trusted entity** page, for **Trusted entity type**, choose **AWS Service**. For **Use cases for other AWS services**, choose **DMS**.

5. Select the **DMS** check box and choose **Next**.
6. On the **Add permissions** page, choose **HomogeneousDataMigrationsPolicy** that you created before. Choose **Next**.
7. On the **Name, review, and create** page, enter **HomogeneousDataMigrationsRole** for **Role name**, and choose **Create role**.
8. On the **Roles** page, enter **HomogeneousDataMigrationsRole** for **Role name**. Choose **HomogeneousDataMigrationsRole**.
9. On the **HomogeneousDataMigrationsRole** page, choose the **Trust relationships** tab. Choose **Edit trust policy**.
10. On the **Edit trust policy** page, paste the following JSON into the editor, replacing the existing text.

    JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "dms-data-migrations.amazonaws.com",
                    "dms.your_region.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

    In the preceding example, replace *your_region* with the name of your AWS Region.

    The preceding resource-based policy provides AWS DMS service principals with permissions to perform tasks according to the customer managed **HomogeneousDataMigrationsPolicy** policy.

11. Choose **Update policy**.

Creating a secret access policy and role

Follow the procedures below to create your secret access policy and role which allow DMS to access the user credentials for your source and target databases.

To create the secret access policy and role, which allows Amazon RDS to access AWS Secrets Manager to access your appropriate secret

1. Sign in to the AWS Management Console and open the AWS Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/.
2. Choose **Policies**, then choose **Create policy**.
3. Choose **JSON** and enter the following policy to enable access to and decryption of your secret.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SecretName-ABCDEF"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

   Here, *secret_arn* is the ARN of your secret, which you can get from either `SecretsManagerSecretId` as appropriate, and *kms_key_arn* is the ARN of the AWS KMS key that you are using to encrypt your secret, as in the following example.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-2:123456789012:secret:MySQLTestSecret-qeHamH"
        },
        {
             "Effect": "Allow",
             "Action": [
                        "kms:Decrypt",
                        "kms:DescribeKey"
                      ],
             "Resource": "arn:aws:kms:us-
east-2:123456789012:key/761138dc-0542-4e58-947f-4a3a8458d0fd"
        }
     ]
}
```

> **Note**
>
> If you use the default encryption key created by AWS Secrets Manager, you do not have to specify the AWS KMS permissions for *kms_key_arn*.
> If you want your policy to provide access to both secrets, simply specify an additional JSON resource object for the other *secret_arn*.

4. Review and create the policy with a friendly name and optional description.
5. Choose **Roles**, then choose **Create role**.
6. Choose **AWS service** as the type of trusted entity.
7. Choose **DMS** from the list of services as the trusted service, then choose **Next: Permissions**.
8. Look up and attach the policy you created in step 4, then proceed through adding any tags and review your role. At this point, edit the trust relationships for the role to use your Amazon RDS regional service principal as the trusted entity. This principal has the following format.

```
dms.region-name.amazonaws.com
```

Here, *region-name* is the name of your region, such as `us-east-1`. Thus, an Amazon RDS regional service principal for this region follows.

```
dms.us-east-1.amazonaws.com
dms-data-migrations.amazonaws.com
```

Creating an IAM role for AWS DMS to manage Amazon VPC

You must create an IAM role for AWS DMS to manage the VPC settings for your resources. This role must be available for successful migration.

Creating the `dms-vpc-role` for database migration

<result>

This creates the role for the DMS to manage the VPC settings for the migration.
</result>

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the console, choose **Roles** and then choose **Create role**.
3. Choose the **AWS service** option for the **Select trusted entity** option.

   For **Use case**, select **DMS**.
4. For the **Add permissions** step, select `AmazonDMSVPCManagementRole` and choose **Next**.
5. In the **Name, review, and create** page, set the **Role name** to `dms-vpc-role` and choose **Create role**.

Setting up data migration for EC2 database

To begin migrating data from your EC2 source database, you must create an equivalent RDS database. For instructions on creating your database, see Creating an Amazon RDS DB instance.

After creating your target database, use the following steps to set up the data migration:

Set up data migration project

1. Select the target database on the **Databases** page in the RDS console.
2. Choose the **Actions** dropdown and select the **Migrate data from EC2 database** option. To see the supported target databases, see Limitations.
3. Under the **Select source EC2 database** section:
    1. Check the **Engine type** and make sure it is the same as your source database.

       Also, check if the engine versions are compatible.
    2. For **EC2 instance**, choose the EC2 instance where your source database resides.
    3. For **Port**, enter the port on which your source database allows traffic.
    4. For **Secret**, choose **Create and use a new secret** if you don't have an existing secret. Enter the **Username** and **Password** for your source database. Also choose the KMS key with which to encrypt your secret.

       If you have an existing secret, select **Use an existing secret** and then choose secret from the dropdown.
    5. For **IAM role for secret**, if you have an existing IAM role, select **Use an existing IAM role** and choose an IAM role from the dropdown that can access the secret ID from the previous step.

       If you don't have existing IAM role, choose **Create and use new IAM role**. Enter the new name for your role for **IAM role name. You can see the permissions associated with this role in the link below.**
4. Under the **View target RDS database** section:
    1. Confirm the settings of your target database at the top of the section.
    2. For **Secret**, choose **Create and use a new secret** if you don't have an existing secret that holds your target database credentials.

       If you have an existing secret, select the secret from the dropdown.
    3. For **IAM role for secret**, select an IAM role that can access the secret from the previous step. You can also create a new IAM role if you don't have existing IAM role.

       If the dropdown doesn't populate the IAM roles, specify the **IAM role ARN** in the format `arn:aws:iam:`*`account_id`*`:role/`*`roleName`*.
5. Under the **Configure data migration** section:

1. Select the type of data migration by selecting between **Full load**, **Full load and change data capture (CDC)**, or **Change data capture (CDC)**. For more information about these options, see Overview.

   You can't modify the migration type afer the migration starts.
2. For **IAM role for data migration**, if you have an existing IAM role, select **Use an existing IAM role** and choose an IAM role from the dropdown that grants DMS the permissions to create the resources required for the migration. If you don't have existing IAM role, choose **Create and use new IAM role**.

6. Confirm that the **View migration settings** tab shows the required settings for your data migration to be set up successfully.
7. Select **Migrate** to complete the migration set up.

After completing these steps, you can see the resources being set up for the data migration by choosing **View details** in the progress banner in the console. Once the required resources are set up, the migration automatically starts. If you create

To migrate multiple databases into the target database, start this process again with details about the new EC2 database.

Managing data migrations

After using the **Migrate data from EC2 database** action from the RDS console, RDS starts the migration automatically.

If you used the AWS DMS console to create the migration resources, you can start the migration process.

Starting the data migration

Follow these steps to start data migration:

Starting a data migration

1. Choose the target database on the **Databases** page in the RDS console.
2. In the database details page, choose the **Data migrations** tab.
3. Under the **Data migrations** tab, the **Associated data migrations** lists the available data migrations.

Migrations set up using the RDS console start automatically once the required resources are set up.

Migrations set up using the DMS console are set to **Ready**.

To begin these migrations, select the **Actions** drop down and select **Start**.

4. This begins the data migration for your EC2 database.

Stopping the data migration

For data migrations whose replication type is full load, stopping the migration causes the process to stop and can't be resumed. Once stopped, you must restart the migration.

For migrations with replication type set to change data capture (CDC) or full load and CDC, you can stop the continuous replication process, and resume the process later.

Stopping a data migration

1. Choose the target database on the **Databases** page in the RDS console.
2. In the database details page, choose the **Data migrations** tab.
3. Under the **Data migrations** tab, the **Associated data migrations** lists the ongoing data migrations.

   To stop a migration, select a data migration and select **Stop** in the **Actions** drop down.
4. This stops the data migration for your EC2 database.

Resuming the data migration

For data migrations whose replication type is full load and change data capture (CDC) or change data capture (CDC) migration, you can resume the CDC process from the last stop point.

Resuming a data migration

1. Choose the target database on the **Databases** page in the RDS console.
2. In the database details page, choose the **Data migrations** tab.
3. Under the **Data migrations** tab, the **Associated data migrations** lists the stopped data migrations.

To resume a migration, select a data migration and select **Resume processing** in the **Actions** drop down.

4. This resume the data migration for your EC2 database.

Deleting the data migration

To delete an associated data migration, use the following instructions

Deleting a data migration

1. Choose the target database on the **Databases** page in the RDS console.
2. In the database details page, choose the **Data migrations** tab.
3. To delete a migration, select a data migration and select **Delete** in the **Actions** drop down.
4. This deletes the data migration.

Deleting a data migration that was in progress doesn't impact any data that has already been loaded to the target database.

Restarting the data migration

To restart an associated data migration from a CDC start point, use the following instructions

Restarting a data migration

1. Choose the target database on the **Databases** page in the RDS console.
2. In the database details page, choose the **Data migrations** tab.
3. To restart a migration, select a data migration and select **Restart** in the **Actions** drop down.
4. This restarts the data migration from a CDC start point.

Restarting a data migration that was in progress doesn't impact any data that has already been loaded to the target database.

Monitoring your data migrations

After the data migrations starts, you can monitor its status and progress. Data migrations of large data sets take hours to complete. To maintain the reliability, availability, and high performance of your data migration, monitor its progress regularly.

To check the status and progress of your data migration

1. Choose the target database on the **Databases** page in the RDS console.
2. In the database details page, choose the **Data migrations** tab.
3. The **Associated data migrations** section lists your data migrations. Check the **Status** column.
4. For running data migrations, the **Migration process** column displays the percentage of migrated data.
5. To monitor the process in CloudWatch, use the link in the in **CloudWatch** column.

Migration statuses

For each data migration that you run, the RDS console displays the **Status**. The following list includes the statuses:

- `Ready`: The data migration is ready to start.
- `Starting`: RDS is creating the serverless environment for your data migration.
- `Load running`: RDS is performing the full load migration.
- `Load complete, replication ongoing`: RDS completed the full load and now replicates the ongoing changes. This status only applies for full load and CDC type migrations.
- `Replication ongoing`: RDS is replicating ongoing changes. This status only applies to CDC type migrations.
- `Stopping`: RDS is stopping the data migrations. This status applies when you choose to stop the data migration from the **Actions** menu.
- `Stopped`: RDS has stopped the data migration.
- `Failed`: The data migration has failed. For more information, see the log files.
- `Restarting`: The data migration has restarted an ongoing data replication from a CDC start point.

Tutorial: Creating a MySQL DB instance with a custom parameterand new option group

In this tutorial, you create a MySQL DB instance with a custom parameter and new option group. For more information about custom parameter and option groups, see Parameter groups for Amazon RDS and Working with option groups.

> **Important**
>
> There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources you use. You can delete these resources after you complete the tutorial if they are no longer needed.

To create a DB instance with custom configurations and settings, you can use custom parameter and new option groups. Custom parameter and new option groups are particularly helpful if you work with multiple databases and want to uniformly configure settings for them.

By completing these steps, you learn:

- How to use Amazon RDS to create a MySQL DB instance with a custom parameter and new option group.
- How to use specific parameters and options for MySQL DB instances.

To complete this tutorial, perform the following tasks:

1. Create a custom parameter group with the MySQL parameters `default_password_lifetime`and `disconnect_on_expired_password`.
2. Create a new option group with MySQL option feature `MariaDB Audit Plugin`. For steps to create an option group, see Working with option groups.
3. Create a MySQL DB instance with the custom parameter group and new option group that you created.

Topics

- Prerequisites
- Create an Amazon RDS parameter group

- Modify parameter values in your custom parameter group
- Create a new Amazon RDS option group
- Add a option to your new option group
- Create MySQL DB instance with a custom parameter and a new option group

Prerequisites

This tutorial requires you to have an AWS account and a user with administrative access. If you don't already have those set up, complete the steps in the following sections:

- Sign up for an AWS account
- Create a user with administrative access

Create an Amazon RDS parameter group

In this tutorial, you learn how to create a custom parameter group with default_password_lifetime and disconnect_on_expired_password for a MySQL DB instance in the console. The `default_password_lifetime` parameter controls the number of days before the client password automatically expires. The `disconnect_on_expired_password` parameter controls whether the MySQL DB instance disconnects the client when the password expires. For more information on other parameters available for MySQL DB instances, see Parameters for MySQL .

To create a parameter group

1. Open the Amazon RDS console and choose **Parameter groups**.
2. For **Custom parameter groups**, choose **Create parameter group**.
3. Set the parameter group details.
    1. Enter a name for the parameter group.
    2. Enter a description of the parameter group.
    3. For **Engine type**, choose **MySQL Community.**
    4. For **Parameter group family**, choose **MySQL 8.0.**
4. Choose **Create**.

The new parameter group appears on the **Parameter groups** page in the Amazon RDS console. The following steps illustrate how to modify parameter values to customize your parameter group.

Modify parameter values in your custom parameter group

Use the following steps to modify the parameter values in the parameter group that you created in Create an Amazon RDS parameter group.

To modify parameter values in your parameter group

1. Open the Amazon RDS console and choose **Parameter groups**.
2. For **Custom parameter groups**, choose the name of the parameter group you created.
3. Choose **Edit**.
4. In the **Filter parameters** search box, search for the custom parameter `default_password_lifetime`.
5. Select the check box next to the parameter and enter a value the number of days to set for this password lifetime parameter.
6. Select **Save Changes**.
7. Repeat the same steps for the parameter `disconnect_on_expired_password`. When you choose this parameter, you are prompted to select a value of 0 or 1 from the dropdown menu. Select 1 to disconnect on expired password.

The custom parameter group is now available to associate with Amazon RDS for MySQL 8.0 DB instance. Next, create a new option group for your DB instance.

Create a new Amazon RDS option group

Create a new option group with the option MariaDB Audit Plugin. This plugin logs server activity for security and compliance. For more information on other custom options available for MySQL DB instances, see Options for MySQL DB instances.

To create an option group

1. Open the Amazon RDS console and choose **Option groups**.
2. For **Option Groups**, choose **Create group**.
3. Set the option group details.

- Enter a name for the option group.
- Enter a description of the option group.
- For **Engine**, select **mysql**.
- For **Major engine version**, select **8.0**.

4. Choose **Create**.

The new option group appears on the **Option groups** page in the Amazon RDS console. The following steps show how to add options to the option group.

Add a option to your new option group

Use the following steps to add a option to the new option group that you created in Create a new Amazon RDS option group.

To add an option to your option group

1. Open the Amazon RDS console and choose **Option groups**.
2. For **Option groups**, select the name of the option group that you created.
3. Under **Options**, choose **Add option**.
4. Set the option group details.
   - For **Option name**, choose the option MariaDB Audit Plugin, **MARIADB_AUDIT_PLUGIN**.
   - For **Option settings**, leave all the default options selected.
   - For **Apply immediately**, choose **Yes**.
5. Choose **Create option**.

The new option group should now be available for all associated DB instances. Next, create a MySQL DB instance with the custom parameter and new option group.

Create MySQL DB instance with a custom parameter and a new option group

Finally, create a MySQL DB instance with the custom parameter and new option group that you made in the previous steps. The following steps show how to create the MySQL DB instance with your custom parameter and new option group.

To create a DB instance with a custom parameter and new option group

1. Open the Amazon RDS console and choose **Databases**.
2. Choose **Create database**.
3. For **Choose a database creation method**, choose **Standard Create**.
4. For **Engine options**, choose **MySQL** .
5. For **Availability and durability**, choose **Single DB instance.** This step is necessary to support a custom parameter or new option group.
6. Select **Additional Configuration**.
   - For **Initial database name**, choose a name for your DB instance.
   - Under the **DB parameter group** dropdown, select the name of the custom parameter group you created previously.
   - Under **Option group** dropdown, select the name of new option group you created previously.
7. For this tutorial, you can leave the default settings for any other DB settings or modify them as required.
8. Choose **Create database**.

RDS creates a new MySQL DB instance with a custom parameter groupand new option group. To see more information on this database, see the **Databases** page of the Amazon RDS console.

In this tutorial, you configured a MySQL DB instance with tailored settings using a custom parameter and a new option group. This newly created MySQL DB instance manages the user password lifetime by using the parameter `default_password_lifetime`. This DB instance also disconnects users that connect with an expired password by using the parameter `disconnect_on_expired_password`. You also use the option `MariaDB Audit Plugin` to keep track of server activity. To optimize your database, you can apply additional setting in your custom parameter group and add options.

After you have finished creating your customized DB instance, you should delete your resources to avoid incurring unwanted costs. To delete a DB instance, follow the instructions in Deleting a DB instance.

Managing an Amazon RDS DB instance

Following, you can find instructions for managing and maintaining your Amazon RDS DB instance.

Topics

- Stopping an Amazon RDS DB instance temporarily
- Starting an Amazon RDS DB instance that was previously stopped
- Rebooting a DB instance
- Automatically connecting an EC2 instance and a DB instance
- Automatically connecting a Lambda function and a DB instance
- Modifying an Amazon RDS DB instance
- Maintaining a DB instance
- Upgrading a DB instance engine version
- Renaming a DB instance
- Working with DB instance read replicas
- Tagging Amazon RDS resources
- Amazon Resource Names (ARNs) in Amazon RDS
- Working with storage for Amazon RDS DB instances
- Deleting a DB instance
- Tutorial: Managing a MySQL DB instance environment from development to production

Stopping an Amazon RDS DB instance temporarily

You can stop your DB instance intermittently for temporary testing or for a daily development activity, for a maximum of 7 consecutive days. The most common use case is cost optimization. The time to stop your DB instance depends on a number of factors. For more information, see Use cases and Time considerations.

> **Warning**
>
> Starting a DB instance requires instance recovery and can take from minutes to hours. Therefore, if instance availability is a concern, be cautious about stopping a production instance temporarily. For more information, see Starting an Amazon RDS DB instance that was previously stopped.

To stop and start your DB instance in the same operation, reboot the DB instance. For more information, see Rebooting a DB instance.

Topics

- Use cases for stopping your DB instance
- Time considerations when stopping your DB instance
- Supported DB engines, instance classes, and Regions
- Stopping a DB instance in a Multi-AZ deployment
- How stopping a DB instance works
- Limitations of stopping your DB instance
- Option and parameter group considerations
- Public IP address considerations
- Stopping a DB instance temporarily: basic steps

Use cases for stopping your DB instance

Stopping and starting a DB instance is faster and more efficient than creating a DB snapshot, deleting your DB instance, and then restoring the snapshot when you want to access the instance. Common use cases for stopping an instance include the following:

- **Cost optimization** – For non-production databases, you can stop your Amazon RDS DB instance temporarily to save money. While the instance is stopped, you're not charged for DB instance hours.

  > **Important**
  >
  > While your DB instance is stopped, you are charged for provisioned storage (including Provisioned IOPS). You're also charged for backup storage, including manual snapshots and automated backups within your specified retention window. However, you're not charged for DB instance hours. For more information, see Billing FAQs.

- **Daily development** – If you maintain a DB instance for development purposes, you can start the instance when it's needed and then shut down the instance when it's not needed.
- **Testing** – You might need a temporary DB instance to test backup and recovery procedures, migrations, application upgrades, or related activities. In these use cases, you can stop the DB instance when it's not needed.
- **Training** – If you're conducting training in RDS, you might need to start DB instances during the training session and shut them down afterward.

Time considerations when stopping your DB instance

The time to stop your DB instance varies depending on factors such as the instance class, network state, DB engine type, database state, and workload. The process can take several minutes or even up to an hour. Even if you don't choose to take a final snapshot, the service must perform internal operations to ensure data consistency. These operations might extend the stop time. They can't be skipped. The DB instance status remains in a stopping state until all operations are complete. These operations include the following actions:

- Shut down database engine processes.
- Shut down RDS platform processes.
- Detach the EBS storage volumes associated with your DB instance.
- Terminate the underlying Amazon EC2 instance.

To minimize delays, consider taking the following actions:

- Take a manual backup.

- Stop the DB instance during periods of low activity.
- Maintain recent automated backups.

Supported DB engines, instance classes, and Regions

You can stop and start Amazon RDS DB instances that are running the following DB engines:

- Db2
- MariaDB
- Microsoft SQL Server, including RDS Custom for SQL Server
- MySQL
- Oracle
- PostgreSQL

Stopping and starting a DB instance is supported for all DB instance classes, and in all AWS Regions.

Stopping a DB instance in a Multi-AZ deployment

You can stop and start a DB instance in a Multi-AZ deployment. Note the following limitations:

- You can only create a Multi-AZ deployment if your database engine supports it. For more information about engine support, see Supported Regions and DB engines for Multi-AZ DB clusters in Amazon RDS.
- RDS for SQL Server doesn't support stopping a DB instance in a Multi-AZ deployment. For more information, see Microsoft SQL Server Multi-AZ deployment limitations, notes, and recommendations.
- A long time might be required to stop a DB instance.

How stopping a DB instance works

The stopping operation occurs in the following stages:

1. The DB instance initiates the normal shutdown process.

   The status of the DB instance changes to `stopping`.

2. The instance stops running, up to a maximum of 7 consecutive days.

   The status of the DB instance changes to `stopped`.

Characteristics of a stopped DB instance

When in a stopped state, your DB instance has the following characteristics:

- Your stopped DB instance retains the following:
  - Instance ID
  - Domain Name Server (DNS) endpoint
  - Parameter group
  - Security group
  - Option group
  - Amazon S3 transaction logs (necessary for a point-in-time restore)

  When you restart a DB instance, it has the same configuration as when you stopped it.
- Any storage volumes remain attached to the DB instance, and their data is kept. RDS deletes any data stored in the RAM of the DB instance.

  While your DB instance is stopped, you are charged for provisioned storage (including Provisioned IOPS). You're also charged for backup storage, including manual snapshots and automated backups within your specified retention window.
- RDS removes pending actions, including scheduled maintenance updates, except for pending actions for the option group or DB parameter group of the DB instance.

> **Note**
>
> Occasionally, an RDS for PostgreSQL DB instance doesn't shut down cleanly. If this happens, you see that the instance goes through a recovery process when you restart it later. This is expected behavior of the database engine, intended to protect database integrity. Some memory-based statistics and counters don't retain history and are re-initialized after restart, to capture the operational workload moving forward.

Automatic restart of a stopped DB instance

If you don't manually start your DB instance after it is stopped for seven consecutive days, RDS automatically starts your DB instance for you. This way, your instance doesn't fall behind any required maintenance updates. To learn how to stop and start your instance on a schedule, see How can I use Step Functions to stop an Amazon RDS instance for longer than 7 days?.

Limitations of stopping your DB instance

The following are some limitations of the stopping operation:

- You can't stop a DB instance that has a read replica, or that is a read replica.
- You can't modify a stopped DB instance.
- You can't delete an option group that is associated with a stopped DB instance.
- You can't delete a DB parameter group that is associated with a stopped DB instance.
- In a Multi-AZ deployment, note the following limitations:
  - You can't stop an RDS for SQL Server DB instance.
  - The primary and secondary Availability Zones might be switched after you start the DB instance.

Additional limitations apply to RDS Custom for SQL Server. For more information, see Starting and stopping an RDS Custom for SQL Server DB instance.

Option and parameter group considerations

You can't remove persistent options (including permanent options) from an option group if there are DB instances associated with that option group. This functionality is also true of any DB instance with a state of `stopping`, `stopped`, or `starting`.

You can change the option group or DB parameter group that is associated with a stopped DB instance. However, the change doesn't occur until the next time you start the DB instance. If you chose to apply changes immediately, the change occurs when you start the DB instance. Otherwise the change occurs during the next maintenance window after you start the DB instance.

Public IP address considerations

When you stop a DB instance, it retains its DNS endpoint. If you stop a DB instance that has a public IP address, Amazon RDS releases its public IP address. When the DB instance is restarted, it has a different public IP address.

> **Note**
>
> You should always connect to a DB instance using the DNS endpoint, not the IP address.

Stopping a DB instance temporarily: basic steps

You can stop a DB using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To stop a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to stop.
3. For **Actions**, choose **Stop temporarily**.
4. In the **Stop DB instance temporarily** window, select the acknowledgement that the DB instance will restart automatically after 7 days.
5. (Optional) Select **Save the DB instance in a snapshot** and enter the snapshot name for **Snapshot name**. Choose this option if you want to create a snapshot of the DB instance before stopping it.
6. Choose **Stop temporarily** to stop the DB instance, or choose **Cancel** to cancel the operation.

AWS CLI

To stop a DB instance by using the AWS CLI, call the stop-db-instance command with the following option:

- `--db-instance-identifier` – the name of the DB instance.

Example

```
aws rds stop-db-instance --db-instance-identifier mydbinstance
```

RDS API

To stop a DB instance by using the Amazon RDS API, call the StopDBInstance operation with the following parameter:

- `DBInstanceIdentifier` – the name of the DB instance.

Starting an Amazon RDS DB instance that was previously stopped

This topic explains how to start an Amazon RDS DB instance that was previously stopped, outlining the necessary steps and key considerations for resuming use of the database.

You can stop your Amazon RDS DB instance temporarily to save money. After you stop your DB instance, you can restart it to begin using it again. For more information about stopping a DB instance, see Stopping an Amazon RDS DB instance temporarily.

When you start a DB instance that you previously stopped, the DB instance retains information such as the following:

- Instance ID
- Domain Name Server (DNS) endpoint
- DB parameter group
- VPC security group
- DB option group

Amazon RDS bills in one-second increments for database instances and attached storage. There is a 10-minute minimum charge when an instance is started.

To start the instance, the Amazon RDS service performs actions such as the following:

- Provisioning the underlying Amazon EC2 instance
- Starting the RDS processes
- Starting the database engine processes
- Attaching the EBS storage volumes
- Enabling Performance Insights if it was previously enabled
- Recovering the DB instance (recovery occurs even after a normal shutdown)

The time to start your DB instance varies depending on factors such as the instance class, network state, DB engine type, database size, and the database state when the instance was shut down. The startup process can take minutes to hours. We recommend that you consider the variability in startup time when creating your availability plan.

Console

To start a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to start.
3. For **Actions**, choose **Start**.

AWS CLI

To start a DB instance by using the AWS CLI, call the start-db-instance command with the following option:

- `--db-instance-identifier` – The name of the DB instance.

Example

```
aws rds start-db-instance --db-instance-identifier mydbinstance
```

RDS API

To start a DB instance by using the Amazon RDS API, call the StartDBInstance operation with the following parameter:

- `DBInstanceIdentifier` – The name of the DB instance.

Rebooting a DB instance

You can stop and start the database service on your RDS DB instance in a single operation, called *rebooting*. Rebooting might be necessary to apply configuration changes, address minor issues, or resolve network problems without having to perform a full restart or migration of your database.

> **Note**
>
> This topic applies only to rebooting a Single-AZ or Multi-AZ DB *instance*. For instructions to reboot a Multi-AZ DB cluster, see the section called "Rebooting a Multi-AZ DB cluster".

Topics

- Use cases for rebooting a DB instance
- How rebooting a DB instance works
- How rebooting a DB instance in a Multi-AZ deployment works
- Considerations when rebooting a DB instance
- Prerequisites for rebooting a DB instance
- Rebooting a DB instance: basic steps

Use cases for rebooting a DB instance

Typically, you reboot your DB instance for maintenance reasons so that your changes take effect. The following use cases are common:

- **Associating a new DB parameter group** – When you associate a new DB parameter group with a DB instance, RDS applies the modified static and dynamic parameters only after the DB instance is rebooted. However, if you modify dynamic parameters in the DB parameter group after you have associated it with the DB instance, these changes are applied immediately without a reboot. For more information, see Parameter groups for Amazon RDS.
- **Applying a change to a static parameter in an existing DB parameter group** – When you change a static parameter and save the DB parameter group, the status of DB instances associated with this parameter group in the console changes to **pending-reboot**. The parameter change takes effect only after the associated DB instances are rebooted. When you change a

dynamic parameter in an existing parameter group, the change takes effect immediately by default, without requiring a reboot.

> **Note**
>
> The **pending-reboot** status doesn't result in an automatic reboot during the next maintenance window. To apply the latest parameter changes to your DB instance, reboot the DB instance manually. For more information about parameter groups, see Parameter groups for Amazon RDS.

- **Troubleshooting** – You might encounter performance or other operational issues that necessitate a reboot. For example, your DB instance might be unresponsive.

How rebooting a DB instance works

When Amazon RDS reboots your DB instance it performs the following sequential tasks:

1. Stops the database service on your DB instance
2. Starts the database service on your DB instance

The reboot process leads to a brief outage. During this outage, the DB instance status is *rebooting*. An outage occurs for both a Single-AZ deployment and a Multi-AZ DB instance deployment, even when you reboot with a failover.

How rebooting a DB instance in a Multi-AZ deployment works

If the Amazon RDS DB instance is in a Multi-AZ deployment, you can reboot with a failover. This operation is useful to simulate a failure of a DB instance or restore operations to the original Availability Zone after a failover.

During the reboot with failover, Amazon RDS does the following

- Interrupts the database abruptly. The DB instance and its client sessions might not have time to shut down gracefully.

> **Warning**
>
> To avoid the possibility of data loss, we recommend stopping transactions on your DB instance before rebooting with a failover.

- Performs crash recovery of the database if necessary.
- Switches to a standby replica in another AZ automatically. The AZ change might not be reflected in the AWS Management Console, and in calls to the AWS CLI and RDS API, for several minutes.
- Updates the DNS record for the DB instance to point to the standby DB instance. As a result, you need to clean up and re-establish any existing connections to your DB instance. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.
- Creates an Amazon RDS event after the reboot.

On RDS for Microsoft SQL Server, the failover reboots only the primary DB instance. After the failover, the primary DB instance becomes the new secondary DB instance. Parameters might not be updated for Multi-AZ instances. For reboot without failover, both the primary and secondary DB instances reboot, and parameters are updated after the reboot. If the DB instance is unresponsive, we recommend reboot without failover.

Considerations when rebooting a DB instance

Before you reboot your instance, consider the following:

- For a DB instance with read replicas, you can reboot the source DB instance and its read replicas independently. After a reboot completes, replication resumes automatically.
- The reboot time depends on the crash recovery process, database activity at the time of reboot, and the behavior of your specific DB engine. To improve the reboot time, we recommend that you reduce database activity as much as possible during the reboot. This technique reduces rollback activity for in-transit transactions.

Prerequisites for rebooting a DB instance

Make sure that you meet the following prerequisites:

- Your DB instance must be in the `available` state. Your database can be unavailable for several reasons, such as an in-progress backup, a previously requested modification, or a maintenance window operation.
- If you force a failover to a different AZ, your DB instance must be configured for Multi-AZ.
- If you force a failover to a different AZ, we recommend first stopping transactions on your DB instance to prevent possible data loss.

Rebooting a DB instance: basic steps

You can reboot your DB instance using the AWS Management Console, AWS CLI, or RDS API.

Console

To reboot a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to reboot.
3. For **Actions**, choose **Reboot**.

   The **Reboot DB instance** page appears.
4. (Optional) Choose **Reboot with failover?** to force a failover from one AZ to another.
5. Choose **Reboot** to reboot your DB instance.

   Alternatively, choose **Cancel**.

AWS CLI

To reboot a DB instance by using the AWS CLI, call the `reboot-db-instance` command.

Example Simple reboot

For Linux, macOS, or Unix:

```
aws rds reboot-db-instance \
    --db-instance-identifier mydbinstance
```

For Windows:

```
aws rds reboot-db-instance ^
    --db-instance-identifier mydbinstance
```

Example Reboot with failover

To force a failover from one AZ to the other in a Multi-AZ DB cluster, use the `--force-failover` parameter.

For Linux, macOS, or Unix:

```
aws rds reboot-db-instance \
    --db-instance-identifier mydbinstance \
    --force-failover
```

For Windows:

```
aws rds reboot-db-instance ^
    --db-instance-identifier mydbinstance ^
    --force-failover
```

RDS API

To reboot a DB instance by using the Amazon RDS API, call the `RebootDBInstance` operation.

Automatically connecting an EC2 instance and a DB instance

You can use the Amazon RDS console to simplify setting up a connection between an Amazon Elastic Compute Cloud (Amazon EC2) instance and a DB instance. Often, your DB instance is in a private subnet and your EC2 instance is in a public subnet within a VPC. You can use a SQL client on your EC2 instance to connect to your DB instance. The EC2 instance can also run web servers or applications that access your private DB instance. For instructions on setting up a connection between an EC2 instance and a Multi-AZ DB cluster, see the section called “Connecting an EC2 instance and a Multi-AZ DB cluster”.



If you want to connect to an EC2 instance that isn't in the same VPC as the DB instance, see the scenarios in Scenarios for accessing a DB instance in a VPC.

Topics

- Overview of automatic connectivity with an EC2 instance
- Automatically connecting an EC2 instance and an RDS database
- Viewing connected compute resources

- Connecting to a DB instance that is running a specific DB engine

Overview of automatic connectivity with an EC2 instance

When you set up a connection between an EC2 instance and an RDS database, Amazon RDSautomatically configures the VPC security group for your EC2 instance and for your RDS database.

The following are requirements for connecting an EC2 instance with an RDS database:

- The EC2 instance must exist in the same VPC as the RDS database.

  If no EC2 instances exist in the same VPC, then the console provides a link to create one.
- The user who sets up connectivity must have permissions to perform the following Amazon EC2 operations:
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

If the DB instance and EC2 instance are in different Availability Zones, your account may incur cross-Availability Zone costs.

When you set up a connection to an EC2 instance, Amazon RDS acts according to the current configuration of the security groups associated with the RDS database and EC2 instance, as described in the following table.

Current RDS security group configuration	Current EC2 security group configuration	RDS action
There are one or more security groups associated	There are one or more security groups associated	RDS takes no action.

Current RDS security group configuration	Current EC2 security group configuration	RDS action
with the RDS database with a name that matches the pattern `rds-ec2-`*n* (where *n* is a number). A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	with the EC2 instance with a name that matches the pattern `ec2-rds-`*n* (where *n* is a number). A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the RDS database as the source.	A connection was already configured automatically between the EC2 instance and RDS database. Because a connection already exists between the EC2 instance and the RDS database, the security groups aren't modified.

Current RDS security group configuration	Current EC2 security group configuration	RDS action
Either of the following conditions apply: • There is no security group associated with the RDS database with a name that matches the pattern `rds-ec2-`*n*. • There are one or more security groups associated with the RDS database with a name that matches the pattern `rds-ec2-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the EC2 instance. Amazon RDS can't use a security group that doesn't have one inbound rule with the VPC security group of the EC2 instance as the source. Amazon RDS also can't use a security group that has been modified. Examples of modifications include adding a rule or changing the port of an existing rule.	Either of the following conditions apply: • There is no security group associated with the EC2 instance with a name that matches the pattern `ec2-rds-`*n*. • There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-rds-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the RDS database. Amazon RDS can't use a security group that doesn't have one outbound rule with the VPC security group of the RDS database as the source. Amazon RDS also can't use a security group that has been modified.	RDS action: create new security groups

Current RDS security group configuration	Current EC2 security group configuration	RDS action
There are one or more security groups associated with the RDS database with a name that matches the pattern `rds-ec2-`*n*. A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-rds-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the RDS database. Amazon RDS can't use a security group that doesn't have one outbound rule with the VPC security group of the RDS database as the source. Amazon RDS also can't use a security group that has been modified.	RDS action: create new security groups
There are one or more security groups associated with the RDS database with a name that matches the pattern `rds-ec2-`*n*. A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	A valid EC2 security group for the connection exists, but it is not associated with the EC2 instance. This security group has a name that matches the pattern `ec2-rds-`*n*. It hasn't been modified. It has only one outbound rule with the VPC security group of the RDS database as the source.	RDS action: associate EC2 security group

Current RDS security group configuration	Current EC2 security group configuration	RDS action
Either of the following conditions apply: • There is no security group associated with the RDS database with a name that matches the pattern `rds-ec2-`*n*. • There are one or more security groups associated with the RDS database with a name that matches the pattern `rds-ec2-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the EC2 instance. Amazon RDS can't use a security group that doesn't have one inbound rule with the VPC security group of the EC2 instance as the source. Amazon RDS also can't use security group that has been modified.	There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-rds-`*n*. A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the RDS database as the source.	RDS action: create new security groups

RDS action: create new security groups

Amazon RDS takes the following actions:

- Creates a new security group that matches the pattern `rds-ec2-`*n*. This security group has an inbound rule with the VPC security group of the EC2 instance as the source. This security group is associated with the RDS database and allows the EC2 instance to access the RDS database.

- Creates a new security group that matches the pattern `ec2-rds-`*n*. This security group has an outbound rule with the VPC security group of the RDS database as the target. This security group is associated with the EC2 instance and allows the EC2 instance to send traffic to the RDS database.

RDS action: associate EC2 security group

Amazon RDS associates the valid, existing EC2 security group with the EC2 instance. This security group allows the EC2 instance to send traffic to the RDS database.

Automatically connecting an EC2 instance and an RDS database

Before setting up a connection between an EC2 instance and an RDS database, make sure you meet the requirements described in Overview of automatic connectivity with an EC2 instance.

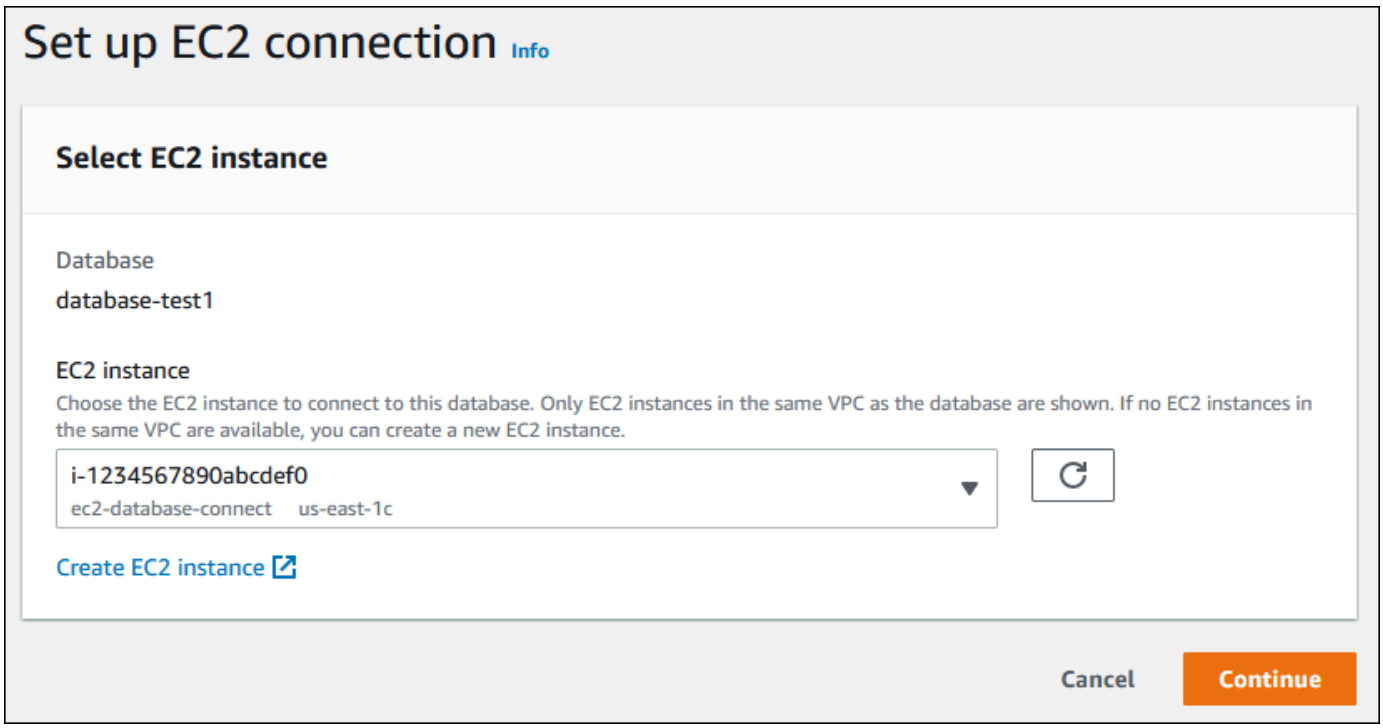
If you make changes to security groups after you configure connectivity, the changes might affect the connection between the EC2 instance and the RDS database.

> **Note**
>
> You can only set up a connection between an EC2 instance and an RDS database automatically by using the AWS Management Console. You can't set up a connection automatically with the AWS CLI or RDS API.

To connect an EC2 instance and an RDS database automatically

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS database.
3. From **Actions**, choose **Set up EC2 connection**.

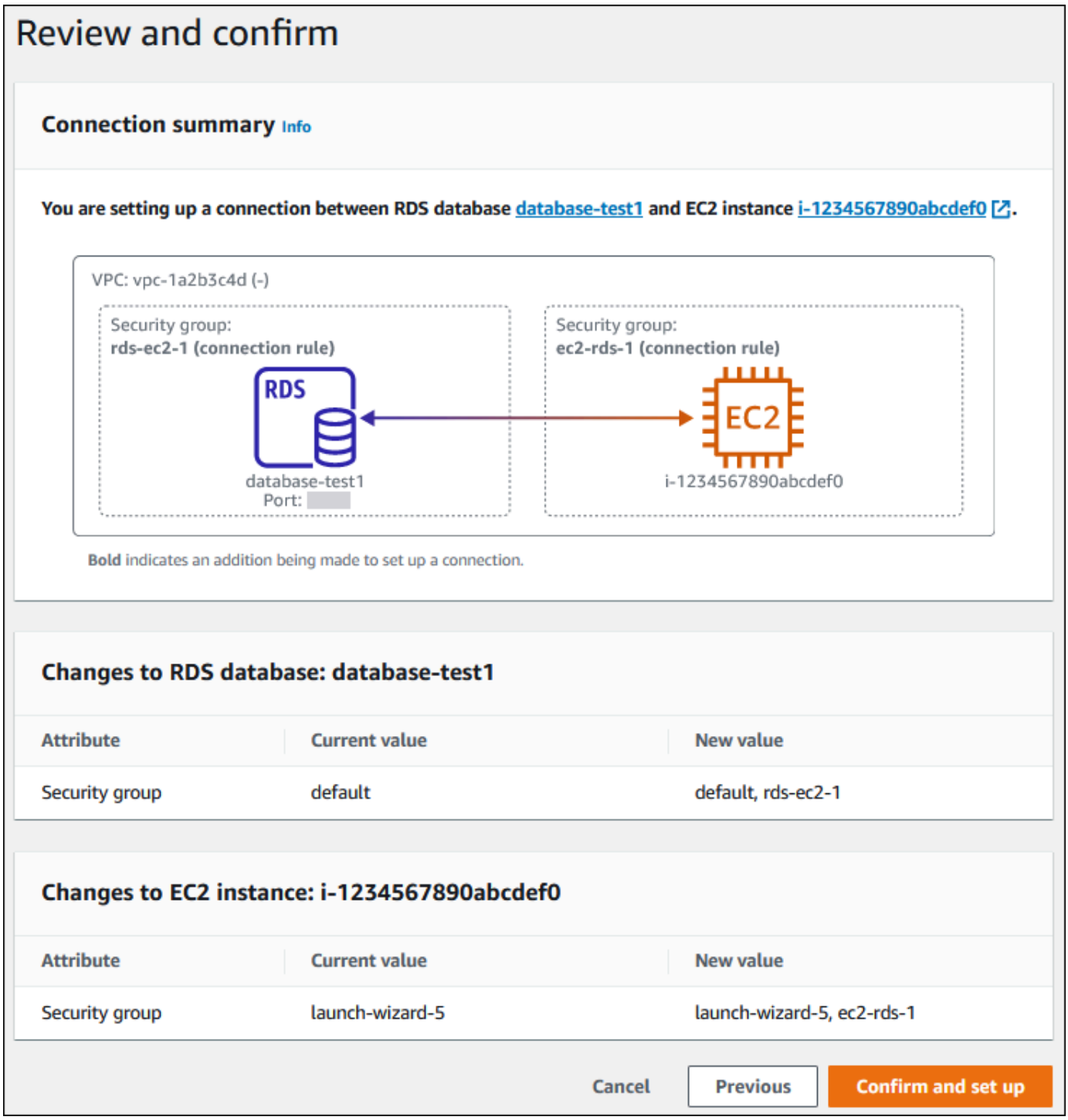
   The **Set up EC2 connection** page appears.
4. On the **Set up EC2 connection** page, choose the EC2 instance.



If no EC2 instances exist in the same VPC, choose **Create EC2 instance** to create one. In this case, make sure the new EC2 instance is in the same VPC as the RDS database.

5. Choose **Continue**.

   The **Review and confirm** page appears.



6. On the **Review and confirm** page, review the changes that RDS will make to set up connectivity with the EC2 instance.

   If the changes are correct, choose **Confirm and set up**.

   If the changes aren't correct, choose **Previous** or **Cancel**.

Viewing connected compute resources

You can use the AWS Management Console to view the compute resources that are connected to an RDS database. The resources shown include compute resource connections that were set up automatically. You can set up connectivity with compute resources automatically in the following ways:

- You can select the compute resource when you create the database.

  For more information, see Creating an Amazon RDS DB instance and Creating a Multi-AZ DB cluster for Amazon RDS.
- You can set up connectivity between an existing database and a compute resource.

  For more information, see Automatically connecting an EC2 instance and an RDS database.

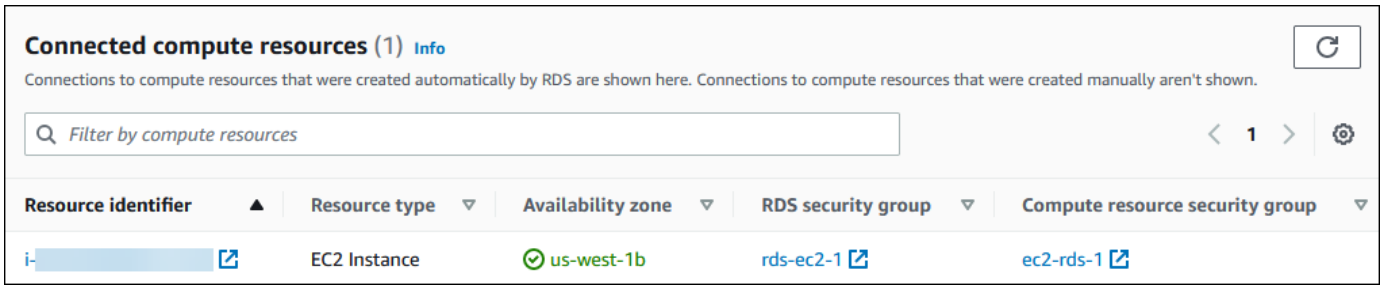
The listed compute resources don't include ones that were connected to the database manually. For example, you can allow a compute resource to access a database manually by adding a rule to the VPC security group associated with the database.

For a compute resource to be listed, the following conditions must apply:

- The name of the security group associated with the compute resource matches the pattern `ec2-rds-`*n* (where *n* is a number).
- The security group associated with the compute resource has an outbound rule with the port range set to the port that the RDS database uses.
- The security group associated with the compute resource has an outbound rule with the source set to a security group associated with the RDS database.
- The name of the security group associated with the RDS database matches the pattern `rds-ec2-`*n* (where *n* is a number).
- The security group associated with the RDS database has an inbound rule with the port range set to the port that the RDS database uses.
- The security group associated with the RDS database has an inbound rule with the source set to a security group associated with the compute resource.

To view compute resources connected to an RDS database

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the name of the RDS database.
3. On the **Connectivity & security** tab, view the compute resources in the **Connected compute resources**.



Connecting to a DB instance that is running a specific DB engine

For information about connecting to a DB instance that is running a specific DB engine, follow the instructions for your DB engine:

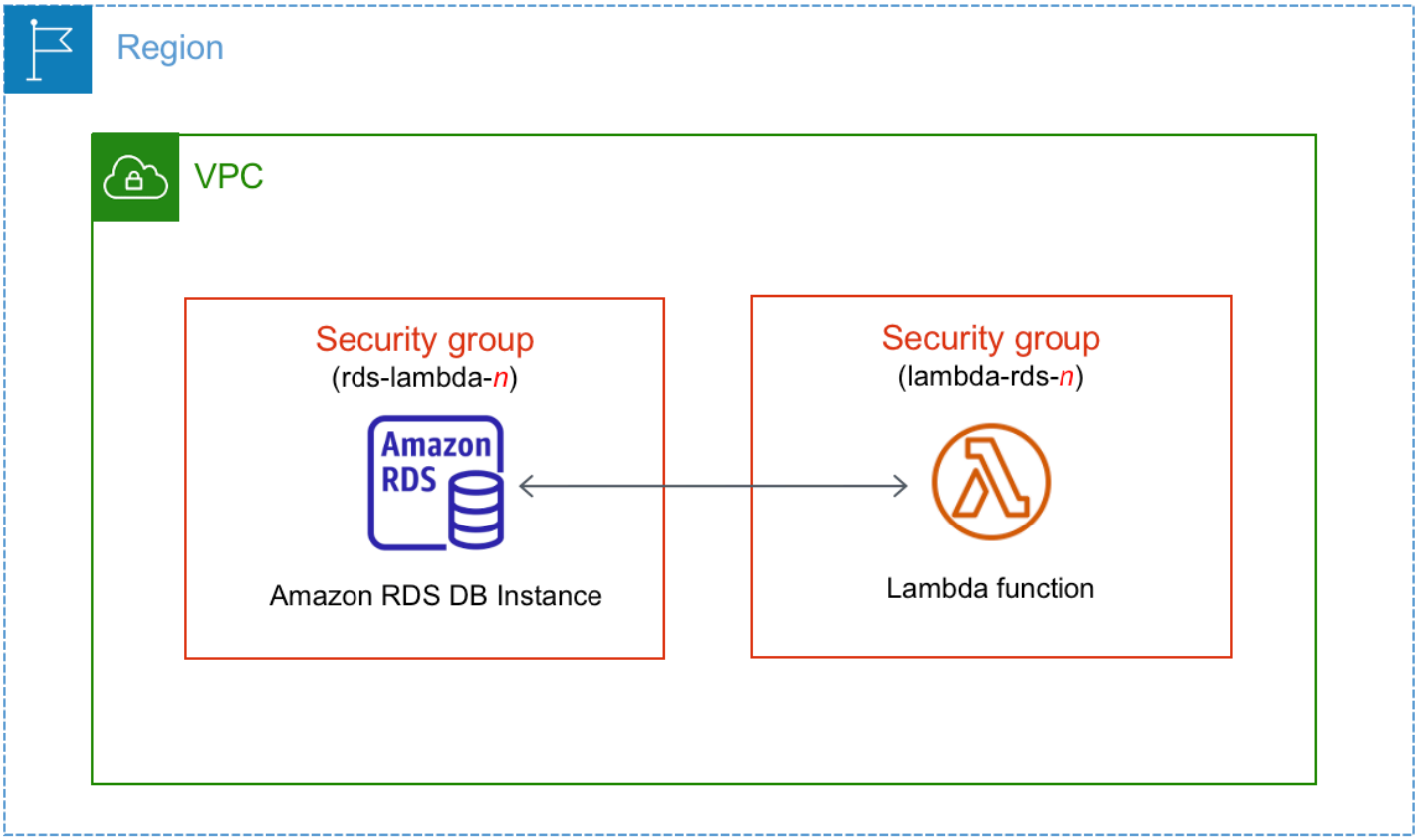
- Connecting to your MariaDB DB instance
- Connecting to your Microsoft SQL Server DB instance
- Connecting to your MySQL DB instance
- Connecting to your Oracle DB instance
- Connecting to a DB instance running the PostgreSQL database engine

Automatically connecting a Lambda function and a DB instance

You can use the Amazon RDS console to simplify setting up a connection between a Lambda function and a DB instance. Often, your DB instance is in a private subnet within a VPC. The Lambda function can be used by applications to access your private DB instance.

For instructions on setting up a connection between a Lambda function and a Multi-AZ DB cluster, see the section called “Connecting a Lambda function and a Multi-AZ DB cluster”.

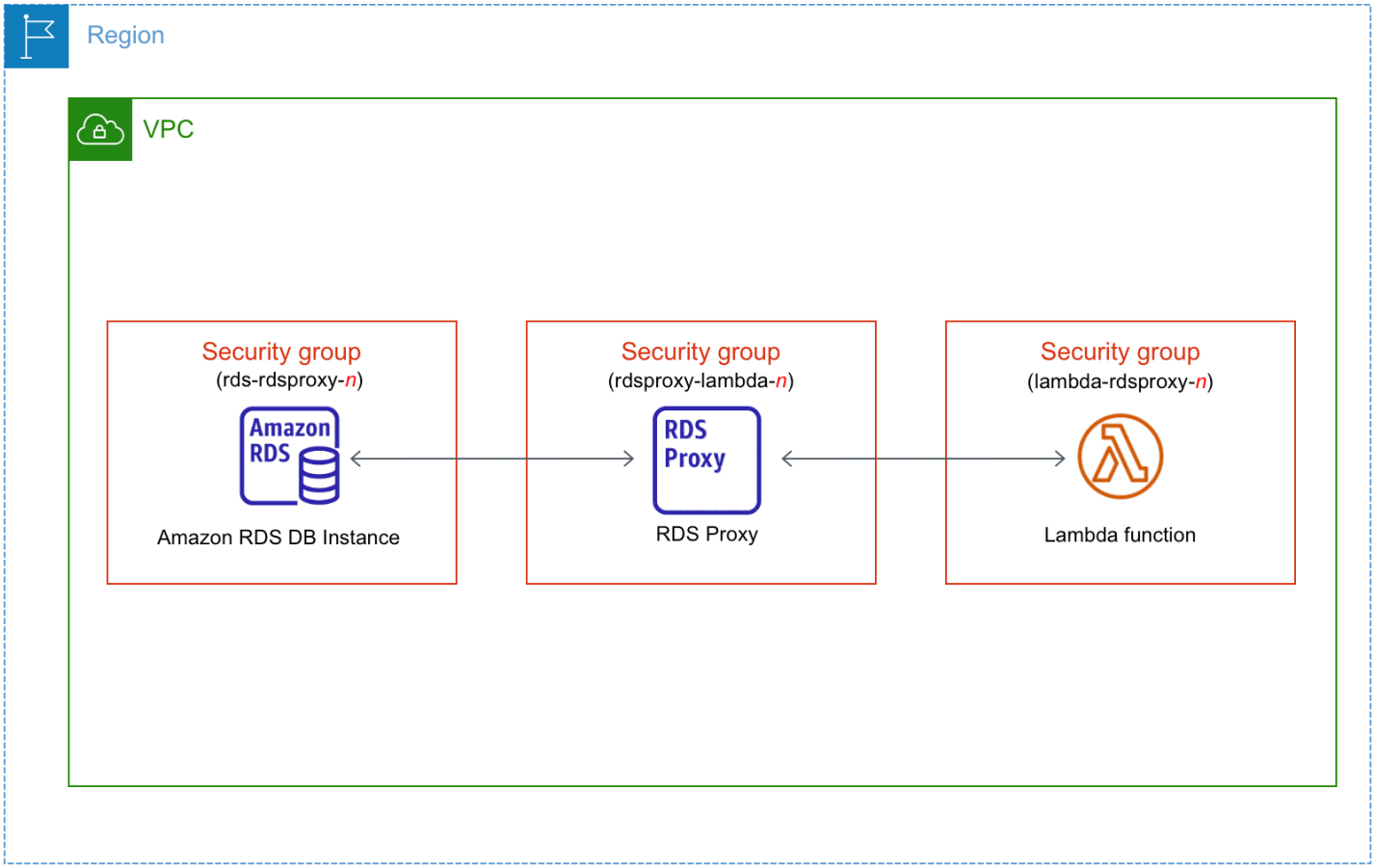
The following image shows a direct connection between your DB instance and your Lambda function.



You can set up the connection between your Lambda function and your DB instance through RDS Proxy to improve your database performance and resiliency. Often, Lambda functions make frequent, short database connections that benefit from connection pooling that RDS Proxy offers. You can take advantage of any AWS Identity and Access Management (IAM) authentication that you already have for Lambda functions, instead of managing database credentials in your Lambda application code. For more information, see Amazon RDS Proxy.

When you use the console to connect with an existing proxy, Amazon RDS updates the proxy security group to allow connections from your DB instance and Lambda function.

You can also create a new proxy from the same console page. When you create a proxy in the console, to access the DB instance, you must input your database credentials or select an AWS Secrets Manager secret.



Tip

To quickly connect a Lambda function to a DB instance, you can also use the in-console guided wizard. To open the wizard, do the following:

1. Open the Functions page of the Lambda console.
2. Select the function you want to connect a database to.
3. On the **Configuration** tab, select **RDS databases**.
4. Choose **Connect to RDS database**.

After you've connected your function to a database, you can create a proxy by choosing **Add proxy**.

Topics

- Overview of automatic connectivity with a Lambda function
- Automatically connecting a Lambda function and an RDS database
- Viewing connected compute resources

Overview of automatic connectivity with a Lambda function

The following are requirements for connecting a Lambda function with an RDS DB instance:

- The Lambda function must exist in the same VPC as the DB instance.
- The user who sets up connectivity must have permissions to perform the following Amazon RDS, Amazon EC2, Lambda, Secrets Manager, and IAM operations:
  - Amazon RDS
    - `rds:CreateDBProxies`
    - `rds:DescribeDBInstances`
    - `rds:DescribeDBProxies`
    - `rds:ModifyDBInstance`
    - `rds:ModifyDBProxy`
    - `rds:RegisterProxyTargets`
  - Amazon EC2
    - `ec2:AuthorizeSecurityGroupEgress`
    - `ec2:AuthorizeSecurityGroupIngress`
    - `ec2:CreateSecurityGroup`
    - `ec2:DeleteSecurityGroup`
    - `ec2:DescribeSecurityGroups`
    - `ec2:RevokeSecurityGroupEgress`
    - `ec2:RevokeSecurityGroupIngress`

- Lambda
  - `lambda:CreateFunctions`
  - `lambda:ListFunctions`
  - `lambda:UpdateFunctionConfiguration`
- Secrets Manager
  - `secretsmanager:CreateSecret`
  - `secretsmanager:DescribeSecret`
- IAM
  - `iam:AttachPolicy`
  - `iam:CreateRole`
  - `iam:CreatePolicy`
- AWS KMS
  - `kms:describeKey`

> **Note**
>
> If the DB instance and Lambda function are in different Availability Zones, your account might incur cross-Availability Zone costs.

When you set up a connection between a Lambda function and an RDS database, Amazon RDS configures the VPC security group for your function and for your DB instance. If you use RDS Proxy, then Amazon RDS also configures the VPC security group for the proxy. Amazon RDS acts according to the current configuration of the security groups associated with the DB instance, Lambda function, and proxy, as described in the following table.

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
There are one or more security groups associated with the DB instance	There are one or more security groups associated with the Lambda function	There are one or more security groups associated with the proxy with a name	Amazon RDS takes no action.

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
with a name that matches the pattern `rds-lambda-` *n* or if a proxy is already connected to your DB instance, RDS checks if the `TargetHealth` of an associated proxy is `AVAILABLE` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the Lambda function or proxy as the source.	with a name that matches the pattern `lambda-rds-` *n* or `lambda-rdsproxy-` *n* (where *n* is a number). A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with either the VPC security group of the DB instance or the proxy as the destination.	that matches the pattern `rdsproxy-lambda-` *n* (where *n* is a number). A security group that matches the pattern hasn't been modified. This security group has inbound and outbound rules with the VPC security groups of the Lambda function and the DB instance.	A connection was already configured automatically between the Lambda function, the proxy (optional), and DB instance. Because a connection already exists between the function, proxy, and the database, the security groups aren't modified.

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
Either of the following conditions apply: • There is no security group associated with the DB instance with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. • There are one or more security groups associated with the DB instance with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. However, none of these security groups can be used for the connectio	Either of the following conditions apply: • There is no security group associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. • There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the DB instance. Amazon RDS can't use a security group	Either of the following conditions apply: • There is no security group associated with the proxy with a name that matches the pattern `rdsproxy-lambda-`*n*. • There are one or more security groups associated with the proxy with a name that matches `rdsproxy-lambda-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the DB instance or Lambda function. Amazon RDS can't use a security group that doesn't have inbound and	RDS action: create new security groups

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
n with the Lambda function. Amazon RDS can't use a security group that doesn't have one inbound rule with the VPC security group of the Lambda function or proxy as the source. Amazon RDS also can't use a security group that has been modified. Examples of modifications include adding a rule or changing the port of an existing rule.	that doesn't have one outbound rule with the VPC security group of the DB instance or proxy as the destination. Amazon RDS also can't use a security group that has been modified.	outbound rules with the VPC security group of the DB instance and the Lambda function. Amazon RDS also can't use a security group that has been modified.	

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
There are one or more security groups associated with the DB instance with a name that matches the pattern `rds-lambda-` *n* or if the `TargetHealth` of an associated proxy is `AVAILABLE` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the Lambda function or proxy as the source.	There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-` *n* or `lambda-rdsproxy-` *n*. However, Amazon RDS can't use any of these security groups for the connection with the DB instance. Amazon RDS can't use a security group that doesn't have one outbound rule with the VPC security group of the DB instance or proxy as the destination. Amazon RDS also can't use a security group that has been modified.	There are one or more security groups associated with the proxy with a name that matches the pattern `rdsproxy-lambda-` *n*. However, Amazon RDS can't use any of these security groups for the connection with the DB instance or Lambda function. Amazon RDS can't use a security group that doesn't have inbound and outbound rules with the VPC security group of the DB instance and the Lambda function. Amazon RDS also can't use a security group that has been modified.	RDS action: create new security groups

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
There are one or more security groups associated with the DB instance with a name that matches the pattern `rds-lambda-` *n* or if the `TargetHealth` of an associated proxy is `AVAILABLE` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the Lambda function or proxy as the source.	A valid Lambda security group for the connection exists, but it isn't associated with the Lambda function. This security group has a name that matches the pattern `lambda-rds-` *n* or `lambda-rdsproxy-` *n*. It hasn't been modified. It has only one outbound rule with the VPC security group of the DB instance or proxy as the destination.	A valid proxy security group for the connection exists, but it isn't associated with the proxy. This security group has a name that matches the pattern `rdsproxy-lambda-` *n*. It hasn't been modified. It has inbound and outbound rules with the VPC security group of the DB instance and the Lambda function.	RDS action: associate Lambda security group

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
Either of the following conditions apply: • There is no security group associated with the DB instance with a name that matches the pattern `rds-lambda-` *n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. • There are one or more security groups associated with the DB instance with a name that matches the pattern `rds-lambda-` *n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. However, Amazon RDS can't use any of these security groups for the connectio	There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-` *n* or `lambda-rdsproxy-` *n*. A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the DB instance or proxy as the destination.	There are one or more security groups associated with the proxy with a name that matches the pattern `rdsproxy-lambda-` *n*. A security group that matches the pattern hasn't been modified. This security group has inbound and outbound rules with the VPC security group of the DB instance and the Lambda function.	RDS action: create new security groups

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
n with the Lambda function or proxy. Amazon RDS can't use a security group that doesn't have one inbound rule with the VPC security group of the Lambda function or proxy as the source. Amazon RDS also can't use a security group that has been modified.			

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
Either of the following conditions apply: • There is no security group associated with the DB instance with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. • There are one or more security groups associated with the DB instance with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. However, Amazon RDS can't use any of these security groups for the connectio	Either of the following conditions apply: • There is no security group associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. • There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the DB instance. Amazon RDS can't use a security group	Either of the following conditions apply: • There is no security group associated with the proxy with a name that matches the pattern `rdsproxy-lambda-`*n*. • There are one or more security groups associated with the proxy with a name that matches `rdsproxy-lambda-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the DB instance or Lambda function. Amazon RDS can't use a security group that doesn't have inbound and	RDS action: create new security groups

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
n with the Lambda function or proxy. Amazon RDS can't use a security group that doesn't have one inbound rule with the VPC security group of the Lambda function or proxy as the source. Amazon RDS also can't use a security group that has been modified.	that doesn't have one outbound rule with the VPC security group of the DB instance or proxy as the source. Amazon RDS also can't use a security group that has been modified.	outbound rules with the VPC security group of the DB instance and the Lambda function. Amazon RDS also can't use a security group that has been modified.	

RDS action: create new security groups

Amazon RDS takes the following actions:

- Creates a new security group that matches the pattern `rds-lambda-`*n* or `rds-rdsproxy-`*n* (if you choose to use RDS Proxy). This security group has an inbound rule with the VPC security group of the Lambda function or proxy as the source. This security group is associated with the DB instance and allows the function or proxy to access the DB instance.
- Creates a new security group that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. This security group has an outbound rule with the VPC security group of the DB instance or proxy as the destination. This security group is associated with the Lambda function and allows the function to send traffic to the DB instance or send traffic through a proxy.
- Creates a new security group that matches the pattern `rdsproxy-lambda-`*n*. This security group has inbound and outbound rules with the VPC security group of the DB instance and the Lambda function.

RDS action: associate Lambda security group

Amazon RDS associates the valid, existing Lambda security group with the Lambda function. This security group allows the function to send traffic to the DB instance or send traffic through a proxy.

Automatically connecting a Lambda function and an RDS database

You can use the Amazon RDS console to automatically connect a Lambda function to your DB instance. This simplifies the process of setting up a connection between these resources.

You can also use RDS Proxy to include a proxy in your connection. Lambda functions make frequent short database connections that benefit from the connection pooling that RDS Proxy offers. You can also use any IAM authentication that you've already set up for your Lambda function, instead of managing database credentials in your Lambda application code.

You can connect an existing DB instance to new and existing Lambda functions using the **Set up Lambda connection** page. The setup process automatically sets up the required security groups for you.

Before setting up a connection between a Lambda function and a DB instance, make sure that:

- Your Lambda function and DB instance are in the same VPC.
- You have the right permissions for your user account. For more information about the requirements, see Overview of automatic connectivity with a Lambda function.

If you change security groups after you configure connectivity, the changes might affect the connection between the Lambda function and the DB instance.

> **Note**
>
> You can automatically set up a connection between a DB instance and a Lambda function only in the AWS Management Console. To connect a Lambda function, the DB instance must be in the **Available** state.

To automatically connect a Lambda function and a DB instance

<result>

After you confirm the setup, Amazon RDS begins the process of connecting your Lambda function, RDS Proxy (if you used a proxy), and DB instance. The console shows the **Connection details** dialog box, which lists the security group changes that allow connections between your resources.

</result>

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to connect to a Lambda function.
3. For **Actions**, choose **Set up Lambda connection**.
4. On the **Set up Lambda connection** page, under **Select Lambda function**, do either of the following:
   - If you have an existing Lambda function in the same VPC as your DB instance, choose **Choose existing function**, and then choose the function.
   - If you don't have a Lambda function in the same VPC, choose **Create new function**, and then enter a **Function name**. The default runtime is set to Nodejs.18. You can modify the settings for your new Lambda function in the Lambda console after you complete the connection setup.
5. (Optional) Under **RDS Proxy**, select **Connect using RDS Proxy**, and then do any of the following:
   - If you have an existing proxy that you want to use, choose **Choose existing proxy**, and then choose the proxy.
   - If you don't have a proxy, and you want Amazon RDS to automatically create one for you, choose **Create new proxy**. Then, for **Database credentials**, do either of the following:

     a. Choose **Database username and password**, and then enter the **Username** and **Password** for your DB instance.

     b. Choose **Secrets Manager secret**. Then, for **Select secret**, choose an AWS Secrets Manager secret. If you don't have a Secrets Manager secret, choose **Create new Secrets Manager secret** to create a new secret. After you create the secret, for **Select secret**, choose the new secret.

     After you create the new proxy, choose **Choose existing proxy**, and then choose the proxy. Note that it might take some time for your proxy to be available for connection.
6. (Optional) Expand **Connection summary** and verify the highlighted updates for your resources.
7. Choose **Set up**.

Viewing connected compute resources

You can use the AWS Management Console to view the Lambda functions that are connected to your DB instance. The resources shown include compute resource connections that Amazon RDS set up automatically.

The listed compute resources don't include those that are manually connected to the DB instance. For example, you can allow a compute resource to access your DB instance manually by adding a rule to your VPC security group associated with the database.

For the console to list a Lambda function, the following conditions must apply:

- The name of the security group associated with the compute resource matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n* (where *n* is a number).
- The security group associated with the compute resource has an outbound rule with the port range set to the port of the DB instance or an associated proxy. The destination for the outbound rule must be set to a security group associated with the DB instance or an associated proxy.
- If the configuration includes a proxy, the name of the security group attached to the proxy associated with your database matches the pattern `rdsproxy-lambda-`*n* (where *n* is a number).
- The security group associated with the function has an outbound rule with the port set to the port that the DB instance or associated proxy uses. The destination must be set to a security group associated with the DB instance or associated proxy.

To view compute resources automatically connected to an DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance.
3. On the **Connectivity & security** tab, view the compute resources under **Connected compute resources**.

Modifying an Amazon RDS DB instance

You can change the settings of a DB instance to accomplish tasks such as adding additional storage or changing the DB instance class. In this topic, you can find out how to modify an Amazon RDS DB instance and learn about the settings for DB instances.

We recommend that you test any changes on a test instance before modifying a production instance. Doing this helps you to fully understand the impact of each change. Testing is especially important when upgrading database versions.

Most modifications to a DB instance you can either apply immediately or defer until the next maintenance window. Some modifications, such as parameter group changes, require that you manually reboot your DB instance for the change to take effect.

> **Important**
>
> Some modifications result in downtime because Amazon RDS must reboot your DB instance for the change to take effect. Review the impact to your database and applications before modifying your DB instance settings.

Console

To modify a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**. The **Modify DB instance** page appears.
4. Change any of the settings that you want. For information about each setting, see Settings for DB instances.
5. When all the changes are as you want them, choose **Continue** and check the summary of modifications.
6. (Optional) Choose **Apply immediately** to apply the changes immediately. Choosing this option can cause downtime in some cases. For more information, see Using the schedule modifications setting.

7. On the confirmation page, review your changes. If they are correct, choose **Modify DB instance** to save your changes.

   Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To modify a DB instance by using the AWS CLI, call the modify-db-instance command. Specify the DB instance identifier and the values for the options that you want to modify. For information about each option, see Settings for DB instances.

Example

The following code modifies `mydbinstance` by setting the backup retention period to 1 week (7 days). The code enables deletion protection by using `--deletion-protection`. To disable deletion protection, use `--no-deletion-protection`. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately. For more information, see Using the schedule modifications setting.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --backup-retention-period 7 \
    --deletion-protection \
    --no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --backup-retention-period 7 ^
    --deletion-protection ^
    --no-apply-immediately
```

Example

The following example adds a storage volume to an RDS for Oracle DB instance. The additional volume uses gp3 storage with 5000 GiB of allocated storage and 12000 IOPS.

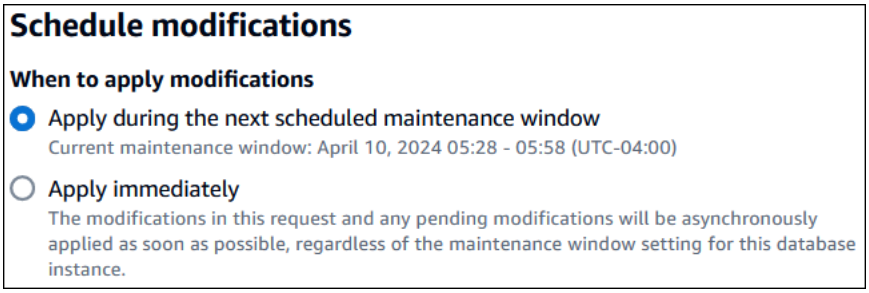
```
aws rds modify-db-instance \
     --db-instance-identifier my-oracle-instance \
     --additional-storage-volumes '[{ \
             "VolumeName": "rdsdbdata2", \
             "StorageType": "gp3", 
             "AllocatedStorage": 5000, \
             "IOPS": 12000 \
         }]'
```

RDS API

To modify a DB instance by using the Amazon RDS API, call the ModifyDBInstance operation. Specify the DB instance identifier, and the parameters for the settings that you want to modify. For information about each parameter, see Settings for DB instances.

Using the schedule modifications setting

When you modify your DB instance, you decide when you want the modifications to occur.



To apply changes immediately rather than in the next maintenance window, choose the **Apply Immediately** option in the AWS Management Console. Or you use the `--apply-immediately` parameter when calling the AWS CLI or set the `ApplyImmediately` parameter to `true` when using the Amazon RDS API.

If you don't choose to apply changes immediately, RDS puts the changes into the pending modifications queue. During the next maintenance window, RDS applies any pending changes in the queue. If you choose to apply changes immediately, your new changes and any changes in the pending modifications queue are applied.

To see the modifications that are pending for the next maintenance window, use the describe-db-instances AWS CLI command and check the `PendingModifiedValues` field.

Important

If any of the pending modifications require the DB instance to be temporarily unavailable (*downtime*), choosing the apply immediately option can cause unexpected downtime. When you choose to apply a change immediately, any pending modifications are also applied immediately, instead of during the next maintenance window.
If you don't want a pending change to be applied in the next maintenance window, you can modify the DB instance to revert the change. You can do this by using the AWS CLI and specifying the `--apply-immediately` option.
When you are adding an additional storage volume using the `modify-db-instance` command, the RDS adds the storage volume immediately regardless of the `--no-apply-immediately` parameter. If you have other modifications in the request, they will be applied baed on the schedule modifications.

Changes to some database settings are applied immediately, even if you choose to defer your changes. To see how the different database settings interact with the apply immediately setting, see Settings for DB instances.

Settings for DB instances

In the following table, you can find details about which settings you can and can't modify. You can also find when changes can be applied and whether the changes cause downtime for your DB instance. By using Amazon RDS features such as Multi-AZ, you can minimize downtime if you later modify the DB instance. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

You can modify a DB instance using the console, the `modify-db-instance` CLI command, or the `ModifyDBInstance` RDS API operation.

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
Allocated storage The storage, in gibibytes, that you want to allocate for your DB	**CLI option:**	If you choose to apply the change	Downtime doesn't occur during this	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

instance. You can only increase the allocated storage. You can't reduce the allocated storage.
 	
You can't modify the storage of some older DB instances, or DB instances restored from older DB snapshots. The **Allocated storage** setting is disabled in the console if your DB instance isn't eligible. You can check whether you can allocate more storage by using the CLI command describe-valid-db-instance-modifications. This command returns the valid storage options for your DB instance.
 	
You can't modify allocated storage if the DB instance status is **storage-optimization**. You also can't modify allocated storage for a DB instance if it's been modified in the last six hours.
 	
The maximum storage allowed depends on your DB engine and the storage type. For more information, see Amazon RDS DB instance storage.
	`--allocated-storage`
	
	RDS API parameter:
	
	`AllocatedStorage`
		immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			change. Performance might be degraded during the change.	

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
Architecture configuration A configuration that allows multiple tenant databases to reside in your DB instance. Currently, only RDS for Oracle container databases (CDBs) support this setting. If your CDB is in the single-tenant configuration, you can modify it to use the **Multi-tenant configuration**. In this configuration, you can use RDS APIs to create 1–30 tenant databases, depending on the database edition and any required option licenses. Application PDBs and proxy PDBs aren't supported. The multi-tenant configuration is permanent, which means that you can't later convert your CDB back to the single-tenant configuration. **Note** The Amazon RDS configuration is called "multi-tenant" rather than "multitenant" because it is a capabilit	**CLI option:** `--multi-tenant` (multi-tenant configuration of the CDB architecture) `--no-multi-tenant` (single-tenant configuration of the CDB architecture) **API parameter:** `MultiTenant`	The change occurs immediately.	Downtime doesn't occur during this change.	Oracle

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
y of Amazon RDS, not just the Oracle DB engine. Similarly, the RDS term "tenant" refers to any tenant in an RDS configuration, not just Oracle PDBs. In the RDS documentation, the unhyphenated term "Oracle multitenant" refers exclusively to the Oracle database CDB architecture, which is compatible with both on-premises and RDS deployments. For more information, see Overview of RDS for Oracle CDBs.				

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
Architecture settings The architecture of the Oracle database: CDB or non-CDB. If you choose **Oracle multitenant architecture**, RDS for Oracle converts your non-CDB into a CDB that uses the single-tenant configuration. This setting is supported only if your database is a non-CDB running Oracle Database 19c with the April 2021 or higher RU. After conversion, your CDB contains one initial pluggable database (PDB). The architecture change is permanent, which means that you can't convert your CDB back to a non-CDB. **Note** To convert a CDB in the single-tenant configuration to the multi-tenant configuration, modify your CDB instance again and choose **Multi-tenant configuration** for your **Architecture configuration**.	**CLI option:** `--engine oracle-ee-cdb` (Oracle multitenant) `--engine oracle-se2-cdb` (Oracle multitenant) **API parameter:** `Engine`	If you choose to apply the change immediately, it occurs immediately. If you don't choose to apply the change immediately, it occurs during the next maintenance window.	Downtime occurs during this change.	Oracle

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
For more information, see Single-tenant configuration of the CDB architecture.				

Auto minor version upgrade
 	
Choose **Enable auto minor version upgrade** to enable your DB instance to receive preferred minor DB engine version upgrades automatically when they become available. This is the default behavior. Amazon RDS performs automatic minor version upgrades in the maintenance window. If you don't choose **Enable auto minor version upgrade**, your DB instance isn't upgraded automatically when new minor versions become available.
 	
For more information, see Automatically upgrading the minor engine version.
	CLI option:
	
	`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`
	
	RDS API parameter:
	
	`AutoMinorVersionUpgrade`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
Backup replication Choose **Enable replication to another AWS Region** to create backups in an additional Region for disaster recovery. Then choose the **Destination Region** for the additional backups.	Not available when modifying a DB instance. For information on enabling cross-Region backups using the AWS CLI or RDS API, see Enabling cross-Region automated backups for Amazon RDS.	The change is applied asynchronously, as soon as possible.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Backup retention period
 	
The number of days that automatic backups are retained. To disable automatic backups, set the backup retention period to 0.
 	
For more information, see Introduction to backups.
 	
Note
If you use AWS Backup to manage your backups, this option doesn't apply. For information about AWS Backup, see the *AWS Backup Developer Guide*.
	CLI option:
	
	`--backup-retention-period`
	
	RDS API parameter:
	
	`BackupRetentionPeriod`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, and you change the setting from a nonzero value to another nonzero value, the change is applied asynchronously, as soon as possible. Otherwise, the change occurs during the next maintenance window.
			Downtime occurs if you change from 0 to a nonzero value, or from a nonzero value to 0.
			
			This applies to both Single-AZ and Multi-AZ DB instances.
				All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Backup window
 	
The time range during which automated backups of your databases occur. The backup window is a start time in Universal Coordinated Time (UTC), and a duration in hours.
 	
For more information, see Introduction to backups.
 	
Note
If you use AWS Backup to manage your backups, this option doesn't appear. For information about AWS Backup, see the *AWS Backup Developer Guide*.
	CLI option:
	
	`--preferred-backup-window`
	
	RDS API parameter:
	
	`PreferredBackupWindow`
		The change is applied asynchronously, as soon as possible.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Certificate authority
 	
The certificate authority (CA) for the server certificate used by the DB instance.
 	
For more information, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
	CLI option:
	
	`--ca-certificate-identifier`
	
	RDS API parameter:
	
	`CACertificateIdentifier`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime only occurs if the DB engine doesn't support rotation without restart. You can use the describe-db-engine-versions AWS CLI command to determine whether the DB engine supports rotation without restart.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Copy tags to snapshots
 	
If you have any DB instance tags, enable this option to copy them when you create a DB snapshot.
 	
For more information, see Tagging Amazon RDS resources.
	CLI option:
	
	`--copy-tags-to-snapshot` or `--no-copy-tags-to-snapshot`
	
	RDS API parameter:
	
	`CopyTagsToSnapshot`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Database port
 	
The port that you want to use to access the DB instance.
 	
The port value must not match any of the port values specified for options in the option group that is associated with the DB instance.
 	
For more information, see Connecting to an Amazon RDS DB instance.
	CLI option:
	
	`--db-port-number`
	
	RDS API parameter:
	
	`DBPortNumber`
		The change occurs immediately. This setting ignores the apply immediately setting.	The DB instance is rebooted immediately.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

DB engine version
 	
The version of the DB engine that you want to use. Before you upgrade your production DB instance, we recommend that you test the upgrade process on a test DB instance. Doing this helps verify its duration and validate your applications.
 	
For more information, see Upgrading a DB instance engine version.
	CLI option:
	
	`--engine-version`
	
	RDS API parameter:
	
	`EngineVersion`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime occurs during this change.	All DB engines

DB instance class
 	
The DB instance class that you want to use.
 	
For more information, see DB instance classes.
	CLI option:
	
	`--db-instance-class`
	
	RDS API parameter:
	
	`DBInstanceClass`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime occurs during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

DB instance identifier
 	
The new DB instance identifier. This value is stored as a lowercase string.
 	
For more information about the effects of renaming a DB instance, see Renaming a DB instance.
	CLI option:
	
	`--new-db-instance-identifier`
	
	RDS API parameter:
	
	`NewDBInstanceIdentifier`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime occurs during this change unless your DB engine version supports dynamic SSL upload. To determine whether your version requires a restart, run the following AWS CLI command:
			
			`aws rds describe-db-engine-versions \ --default-only \ --engine your-db-engine \ --query 'DBEngineVersions[*].SupportsCertificateRotationWithoutRestart'`
				All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

DB parameter group
 	
The DB parameter group that you want associated with the DB instance.
 	
For more information, see Parameter groups for Amazon RDS.
	CLI option:
	
	`--db-parameter-group-name`
	
	RDS API parameter:
	
	`DBParameterGroupName`
		The association of the new DB parameter group with the DB instance occurs immediately.	Downtime doesn't occur when you associate a new DB parameter group with your DB instance.
			
			The association of a DB parameter group is different from the application of parameter changes within a parameter group. RDS applies modified static and dynamic parameter settings in the newly associated group only after you manually reboot the DB instance. However, if
				All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
			you modify dynamic parameters in the DB parameter group after you associate it with the DB instance, these parameter settings are applied immediately without requiring a reboot. For more information, see Parameter groups for Amazon RDS and Rebooting a DB instance.	

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Dedicated Log Volume
 	
Use a dedicated log volume (DLV) to store database transaction logs on a storage volume that's separate from the volume containing the database tables.
 	
For more information, see Using a dedicated log volume (DLV).
	CLI option:
	
	`-dedicated-log-volume`
	
	RDS API parameter:
	
	`DedicatedLogVolume`
		The change is applied when the DB instance is rebooted.	Downtime occurs while the DB instance is rebooted.	MariaDB, MySQL, PostgreSQL

Deletion protection
 	
Enable deletion protection to prevent your DB instance from being deleted.
 	
For more information, see Deleting a DB instance.
	CLI option:
	
	`--deletion-protection|--no-deletion-protection`
	
	RDS API parameter:
	
	`DeletionProtection`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Enhanced Monitoring
 	
Enable Enhanced Monitoring to enable gathering metrics in real time for the operating system that your DB instance runs on.
 	
For more information, see Monitoring OS metrics with Enhanced Monitoring.
	CLI option:
	
	`--monitoring-interval` and `--monitoring-role-arn`
	
	RDS API parameter:
	
	`MonitoringInterval` and `MonitoringRoleArn`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

IAM DB authentication
 	
Enable IAM DB authentication to authenticate database users through users and roles.
 	
For more information, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.
	CLI option:
	
	`--enable-iam-database-authentication|--no-enable-iam-database-authentication`
	
	RDS API parameter:
	
	`EnableIAMDatabaseAuthentication`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime doesn't occur during this change.	Only MariaDB, MySQL, and PostgreSQL

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Kerberos authentication
 	
Choose the Active Directory to move the DB instance to. The directory must exist prior to this operation. If a directory is already selected, you can specify **None** to remove the DB instance from its current directory.
 	
For more information, see Kerberos authentication.
	CLI option:
	
	`--domain` and `--domain-iam-role-name`
	
	RDS API parameter:
	
	`Domain` and `DomainIAMRoleName`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			A brief downtime occurs during this change.	Only Microsoft SQL Server, MySQL, Oracle, and PostgreSQL

License model
 	
Choose **bring-your-own-license** to use your license for Db2 and Oracle.
 	
Choose **license-included** to use the general license agreement for Microsoft SQL Server or Oracle.
 	
For more information, see Amazon RDS for Db2 licensing options, Licensing Microsoft SQL Server on Amazon RDS, and RDS for Oracle licensing options.
	CLI option:
	
	`--license-model`
	
	RDS API parameter:
	
	`LicenseModel`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime occurs during this change.	Only Microsoft SQL Server and Oracle

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Log exports
 	
The types of database log files to publish to Amazon CloudWatch Logs.
 	
For more information, see Publishing database logs to Amazon CloudWatch Logs.
	CLI option:
	
	`--cloudwatch-logs-export-configuration`
	
	RDS API parameter:
	
	`CloudwatchLogsExportConfiguration`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Maintenance window
 	
The time range during which system maintenance occurs. System maintenance includes upgrades, if applicable. The maintenance window is a start time in Universal Coordinated Time (UTC), and a duration in hours.
 	
If you set the window to the current time, there must be at least 30 minutes between the current time and the end of the window. This timing helps ensure that any pending changes are applied.
 	
For more information, see Amazon RDS maintenance window.
	CLI option:
	
	`--preferred-maintenance-window`
	
	RDS API parameter:
	
	`PreferredMaintenanceWindow`
		The change occurs immediately. This setting ignores the apply immediately setting.	If there are one or more pending actions that cause downtime, and the maintenance window is changed to include the current time, those pending actions are applied immediately and downtime occurs.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Manage master credentials in AWS Secrets Manager
 	
Select **Manage master credentials in AWS Secrets Manager** to manage the master user password in a secret in Secrets Manager.
 	
Optionally, choose a KMS key to use to protect the secret. Choose from the KMS keys in your account, or enter the key from a different account.
 	
If RDS is already managing the master user password for the DB instance, you can rotate the master user password by choosing **Rotate secret immediately**.
 	
For more information, see Password management with Amazon RDS and AWS Secrets Manager.
	CLI option:
	
	`--manage-master-user-password | --no-manage-master-user-password`
	
	`--master-user-secret-kms-key-id`
	
	`--rotate-master-user-password | --no-rotate-master-user-password`
	
	RDS API parameter:
	
	`ManageMasterUserPassword`
		If you are turning on or turning off automatic master user password management, the change occurs immediately. This change ignores the apply immediately setting.
		
		If you are rotating the master user password, you must specify that the change is applied immediately.
			Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

	`MasterUserSecretKmsKeyId`
	
	`RotateMasterUserPassword`			

Multi-AZ deployment
 	
Yes to deploy your DB instance in multiple Availability Zones. Otherwise, **No**.
 	
For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.
	CLI option:
	
	`--multi-az|--no-multi-az`
	
	RDS API parameter:
	
	`MultiAZ`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime doesn't occur during this change. However, there is a possible performance impact. For more information, see Converting a DB instance to a Multi-AZ deployment for Amazon RDS.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Network type
 	
The IP addressing protocols supported by the DB instance.
 	
IPv4 to specify that resources can communicate with the DB instance only over the Internet Protocol version 4 (IPv4) addressing protocol.
 	
Dual-stack mode to specify that resources can communicate with the DB instance over IPv4, Internet Protocol version 6 (IPv6), or both. Use dual-stack mode if you have any resources that must communicate with your DB instance over the IPv6 addressing protocol. Also, make sure that you associate an IPv6 CIDR block with all subnets in the DB subnet group that you specify.
 	
For more information, see Amazon RDS IP addressing.
	CLI option:
	
	`--network-type`
	
	RDS API parameter:
	
	`NetworkType`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime is possible during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

New master password
 	
The password for your master user. The password must contain 8–41 alphanumeric characters.
	CLI option:
	
	`--master-user-password`
	
	RDS API parameter:
	
	`MasterUserPassword`
		The change is applied asynchronously, as soon as possible. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Option group
 	
The option group that you want associated with the DB instance.
 	
For more information, see Working with option groups.
	CLI option:
	
	`--option-group-name`
	
	RDS API parameter:
	
	`OptionGroupName`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime doesn't occur during this change. One exception is adding the MariaDB Audit Plugin to an RDS for MariaDB or RDS for MySQL DB instance, which might cause an outage.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Performance Insights
 	
Enable Performance Insights to monitor your DB instance load so that you can analyze and troubleshoot your database performance.
 	
Performance Insights isn't available for some DB engine versions and DB instance classes. The **Performance Insights** section doesn't appear in the console if it isn't available for your DB instance.
 	
For more information, see Monitoring DB load with Performance Insights on Amazon RDS and Amazon RDS DB engine, Region, and instance class support for Performance Insights.
	CLI option:
	
	`--enable-performance-insights| --no-enable-performance-insights`
	
	RDS API parameter:
	
	`EnablePerformanceInsights`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All except Db2

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Performance Insights AWS KMS key
 	
The AWS KMS key identifier for the AWS KMS key for encryption of Performance Insights data. The key identifier is the Amazon Resource Name (ARN), AWS KMS key identifier, or the key alias for the KMS key.
 	
For more information, see Turning Performance Insights on and off for Amazon RDS.
	CLI option:
	
	`--performance-insights-kms-key-id`
	
	RDS API parameter:
	
	`PerformanceInsightsKMSKeyId`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All except Db2

Performance Insights Retention period
 	
The amount of time, in days, to retain Performance Insights data. The retention setting is **Default (7 days)**. To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see Pricing and data retention for Performance Insights.
 	
For more information, see Turning Performance Insights on and off for Amazon RDS.
	CLI option:
	
	`--performance-insights-retention-period`
	
	RDS API parameter:
	
	`PerformanceInsightsRetentionPeriod`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All except Db2

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Processor features
 	
The number of CPU cores and the number of threads per core for the DB instance class of the DB instance.
 	
For more information, see Configuring the processor for a DB instance class in RDS for Oracle.
	CLI option:
	
	`--processor-features` and `--use-default-processor-features` | `--no-use-default-processor-features`
	
	RDS API parameter:
	
	`ProcessorFeatures` and `UseDefaultProcessorFeatures`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime occurs during this change.	Only Oracle

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Provisioned IOPS
 	
The Provisioned IOPS (I/O operations per second) value for the DB instance. This setting is available only if you choose one of the following for **Storage type**:
 	
• **General purpose SSD (gp3)**
• **Provisioned IOPS SSD (io1)**
• **Provisioned IOPS SSD (io2)**
 	
For more information, see the section called “Provisioned IOPS storage” and the section called “gp3 storage (recommended)”.
	CLI option:
	
	`--iops`
	
	RDS API parameter:
	
	`Iops`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Public access
 	
Publicly accessible to give the DB instance a public IP address, meaning that it's accessible outside the VPC. To be publicly accessible, the DB instance also has to be in a public subnet in the VPC.
 	
Not publicly accessible to make the DB instance accessible only from inside the VPC.
 	
For more information, see Hiding a DB instance in a VPC from the internet.
 	
To connect to a DB instance from outside its VPC, the DB instance must be publicly accessible. Also, access must be granted using the inbound rules of the DB instance's security group. In addition, other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance.
 	
If your DB instance isn't publicly accessible, you can also use an AWS Site-to-Site VPN connection or an Direct Connect connection to access it from a private
	CLI option:
	
	`--publicly-accessible | --no-publicly-accessible`
	
	RDS API parameter:
	
	`PubliclyAccessible`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
network. For more information, see Internetwork traffic privacy.				

Security group
 	
The VPC security group that you want associated with the DB instance.
 	
For more information, see Controlling access with security groups.
	CLI option:
	
	`--vpc-security-group-ids`
	
	RDS API parameter:
	
	`VpcSecurityGroupIds`
		The change is applied asynchronously, as soon as possible. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Storage autoscaling
 	
Enable storage autoscaling to enable Amazon RDS to automatically increase storage when needed to avoid having your DB instance run out of storage space.
 	
Use **Maximum storage threshold** to set the upper limit for Amazon RDS to automatic ally increase storage for your DB instance. The default is 1,000 GiB.
 	
For more information, see Managing capacity automatic ally with Amazon RDS storage autoscaling.
	CLI option:
	
	`--max-all ocated-st orage`
	
	RDS API parameter:
	
	`MaxAlloca tedStorag e`
		The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Storage throughput
 	
The new storage throughput value for the DB instance. This setting is available only if you choose **General purpose SSD (gp3)** for **Storage type**.
 	
For more information, see the section called “gp3 storage (recommended)”.
	CLI option:
	
	`--storage-throughput`
	
	RDS API parameter:
	
	`StorageThroughput`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime doesn't occur during this change.	All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Storage type
 	
The storage type that you want to use.
 	
If you choose **General Purpose SSD (gp3)**, you can provision additional **Provisioned IOPS** and **Storage throughput** under **Advanced settings**.
 	
If you choose **Provisioned IOPS SSD (io1)** or **Provisioned IOPS SSD (io2)**, enter the **Provisioned IOPS** value.
 	
After Amazon RDS begins to modify your DB instance to change the storage size or type, you can't submit another request to change the storage size, performance, or type for six hours.
 	
For more information, see Amazon RDS storage types.
	CLI option:
	
	`--storage-type`
	
	RDS API parameter:
	
	`StorageType`
		If you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			The following changes all result in a brief downtime while the process starts. After that, you can use your database normally while the change takes place.
			
			• From **General Purpose (SSD)** or **Provisioned IOPS (SSD)** to **Magnetic**.
			• From **Magnetic** to **General Purpose (SSD)** or **Provisioned IOPS (SSD)**.
				All DB engines

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines

Additional storage volumes	**CLI option:**
	
	`--additional-storage-volumes`
	
	RDS API parameter:
	
	`AdditionalStorageVolumes`
		• When adding a storage volume, the change occurs immediately. This action ignores the apply immediately setting.
		• When modifying an additional storage volume, if you choose to apply the change immediately, it occurs immediately.
		
		If you don't choose to apply the change immediately, it occurs during the next
			You can add up to three additional storage volumes to your RDS for Oracle or RDS for SQL Server DB instance. You can configure each additional volume to use gp3 or io2 storage types. You can also specify different allocated storage, IOPS, and throughput settings to optimize for your workload requirements.
			
			Make sure you specify the volume names as `rdsdbdata2` , `rdsdbdata3` ,
				RDS for Oracle and RDS for SQL Server

Console setting and description	CLI option and RDS API parameter	When the change occurs	Downtime notes	Supported DB engines
		maintenance window. • When deleting an additional storage volume, the change occurs immediately. This action ignores the apply immediately setting.	or `rdsdbdata4` . The combined storage across the primary and additional volumes cannot exceed 256 TiB.	

DB subnet group
The DB subnet group for the DB instance. You can use this setting to move your DB instance to a different VPC.
For more information, see Amazon VPC and Amazon RDS.
	CLI option:
	`--db-subnet-group-name`
	RDS API parameter:
	`DBSubnetGroupName`
		If you choose to apply the change immediately, it occurs immediately.
		If you don't choose to apply the change immediately, it occurs during the next maintenance window.
			Downtime occurs during this change.	All DB engines

Maintaining a DB instance

Periodically, Amazon RDS performs maintenance on Amazon RDS resources. The following topics describe these maintenance actions and how to apply them.

Overview of DB instance maintenance updates

Maintenance most often involves updates to the following resources in your DB instance:

- Underlying hardware
- Underlying operating system (OS)
- Database engine version

Updates to the operating system most often occur for security issues. We recommend that you do them as soon as possible. For more information about operating system updates, see the section called “Applying updates”.

Topics

- Offline resources during maintenance updates
- Deferred DB instance modifications
- Eventual consistency for the DescribePendingMaintenanceActions API

Offline resources during maintenance updates

Some maintenance items require that Amazon RDS take your DB instance offline for a short time. Maintenance items that require a resource to be offline include required operating system or database patching. Required patching is automatically scheduled only for patches that are related to security and instance reliability. Such patching occurs infrequently, typically once every few months. It seldom requires more than a fraction of your maintenance window.

Deferred DB instance modifications

Deferred DB instance modifications that you have chosen not to apply immediately are applied during the maintenance window. For example, you might choose to change the DB instance class or parameter group during the maintenance window. Such modifications that you specify using the **pending reboot** setting don't show up in the **Pending maintenance** list. For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

To see the modifications that are pending for the next maintenance window, use the describe-db-instances AWS CLI command and check the `PendingModifiedValues` field.

Eventual consistency for the DescribePendingMaintenanceActions API

The Amazon RDS `DescribePendingMaintenanceActions` API follows an eventual consistency model. This means that the result of the `DescribePendingMaintenanceActions` command might not be immediately visible to all subsequent RDS commands. Keep this in mind when you use `DescribePendingMaintenanceActions` immediately after using a previous API command.

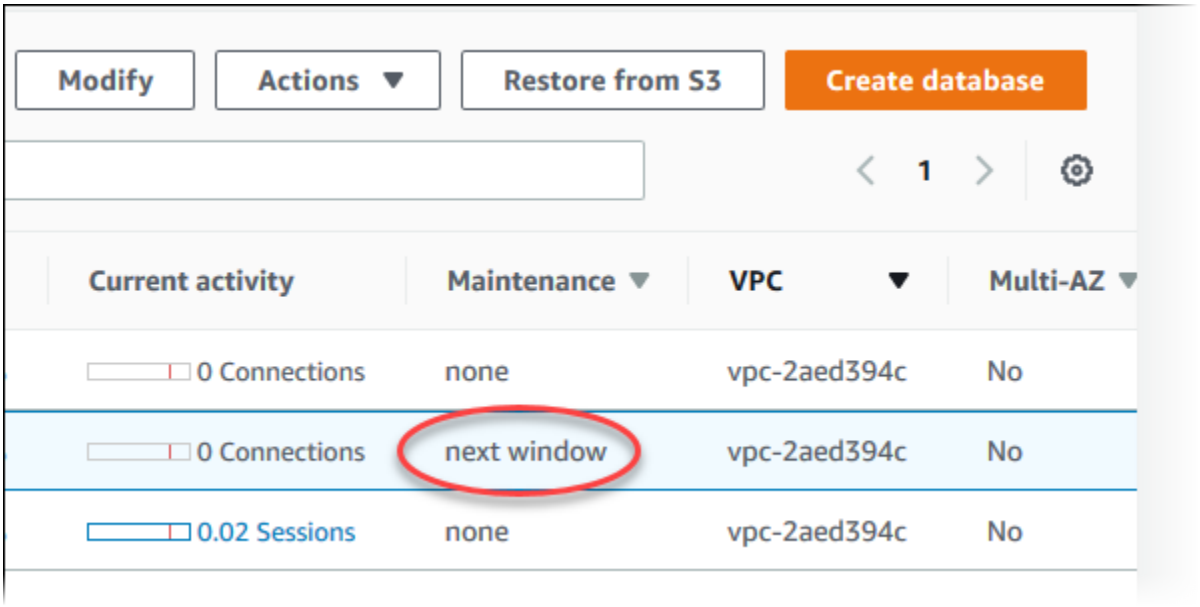
Eventual consistency can affect the way you managed your maintenance updates. For example, if you run the `ApplyPendingMaintenanceActions` command to update the database engine version for a DB instance, it will eventually be visible to `DescribePendingMaintenanceActions`. In this scenario, `DescribePendingMaintenanceActions` might show that the maintenance action wasn't applied even though it was.

To manage eventual consistency, you can do the following:

- Confirm the state of your DB instance before you run a command to modify it. Run the appropriate `DescribePendingMaintenanceActions` command using an exponential backoff algorithm to ensure that you allow enough time for the previous command to propagate through the system. To do this, run the `DescribePendingMaintenanceActions` command repeatedly, starting with a couple of seconds of wait time, and increasing gradually up to five minutes of wait time.
- Add wait time between subsequent commands, even if a `DescribePendingMaintenanceActions` command returns an accurate response. Apply an exponential backoff algorithm starting with a couple of seconds of wait time, and increase gradually up to about five minutes of wait time.

Viewing pending maintenance updates

View whether a maintenance update is available for your DB instance by using the RDS console, the AWS CLI, or the RDS API. If an update is available, it is indicated in the **Maintenance** column for the DB instance on the Amazon RDS console, as shown in this figure.



If no maintenance update is available for a DB instance, the column value is **none** for it.

If a maintenance update is available for a DB instance, the following column values are possible:

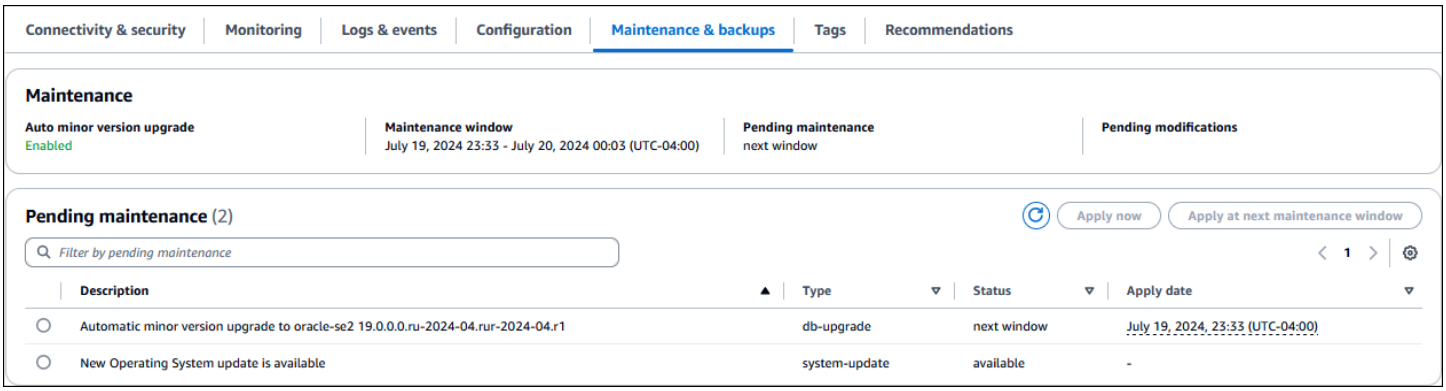
- **required** – The maintenance action will be applied to the resource and can't be deferred indefinitely.
- **available** – The maintenance action is available, but it will not be applied to the resource automatically. You can apply it manually.
- **next window** – The maintenance action will be applied to the resource during the next maintenance window.
- **In progress** – The maintenance action is being applied to the resource.

If an update is available, you can do one of the following:

- If the maintenance value is **next window**, defer the maintenance actions by choosing **Defer upgrade** from **Actions**. You can't defer a maintenance action that has already started.
- Apply the maintenance actions immediately.
- Apply the maintenance actions during your next maintenance window.
- Take no action.

To take an action by using the AWS Management Console

1. Choose the DB instance to show its details.
2. Choose **Maintenance & backups**. The pending maintenance actions appear.
3. Choose the action to take, then choose when to apply it.



The maintenance window determines when pending operations start, but doesn't limit the total run time of these operations. Maintenance operations aren't guaranteed to finish before the maintenance window ends, and can continue beyond the specified end time. For more information, see Amazon RDS maintenance window.

You can also view whether a maintenance update is available for your DB instance by running the describe-pending-maintenance-actions AWS CLI command.

For information about applying maintenance updates, see Applying updates to a DB instance .

Maintenance actions for Amazon RDS

The following maintenance actions apply to RDS DB instances:

- `server-certificate-rotation` – Rotate the Amazon RDS server certificate for the DB instance.

  > **Note**
  >
  > Engines that support rotation without restart don't receive this notification.

- `db-upgrade` – Upgrade the DB engine version for the DB instance.

- `hardware-maintenance` – Perform maintenance on the underlying hardware for the DB instance.
- `system-update` – Update the operating system for the DB instance.

Maintenance for Multi-AZ deployments

Running a DB instance as a Multi-AZ deployment can further reduce the impact of a maintenance event. This result is because Amazon RDS applies operating system updates by following these steps:

1. Perform maintenance on the standby.
2. Promote the standby to primary.
3. Perform maintenance on the old primary, which becomes the new standby.

If you upgrade the database engine for your DB instance in a Multi-AZ deployment, Amazon RDS modifies both primary and secondary DB instances at the same time. In this case, both the primary and secondary DB instances in the Multi-AZ deployment are unavailable during the upgrade. This operation causes downtime until the upgrade is complete. The duration of the downtime varies based on the size of your DB instance.

If there are underlying operating system patches that need to be applied, a short Multi-AZ failover is required to apply the patches to the primary DB instance. This failover typically lasts less than a minute.

If your DB instance runs RDS for MySQL, RDS for PostgreSQL, or RDS for MariaDB, you can minimize the downtime required for an upgrade by using a blue/green deployment. For more information, see Using Amazon RDS Blue/Green Deployments for database updates. If you upgrade an RDS for SQL Server or RDS Custom for SQL Server DB instance in a Multi-AZ deployment, then Amazon RDS performs rolling upgrades, so you have an outage only for the duration of a failover. For more information, see Multi-AZ considerations.

For more information about Multi-AZ deployments, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

Amazon RDS maintenance window

The *maintenance window* is a weekly time interval during which any system changes are applied. Every DB instance has a weekly maintenance window. The maintenance window is an opportunity

to control when modifications and software patching occur. For more information about adjusting the maintenance window, see Adjusting the preferred DB instance maintenance window.

RDS consumes some of the resources on your DB instance while maintenance is being applied. You might observe a minimal effect on performance. For a DB instance, on rare occasions, a Multi-AZ failover might be required for a maintenance update to complete.

If a maintenance event is scheduled for a given week, it's initiated during the 30-minute maintenance window you identify. Most maintenance events also complete during the 30-minute maintenance window, although larger maintenance events may take more than 30 minutes to complete. The maintenance window is paused when the DB instance is stopped.

The 30-minute maintenance window is selected at random from an 8-hour block of time per region. If you don't specify a maintenance window when you create the DB instance, RDS assigns a 30-minute maintenance window on a randomly selected day of the week.

The following table shows the time blocks for each AWS Region from which default maintenance windows are assigned.

Region Name	Region	Time Block
US East (N. Virginia)	us-east-1	03:00–11:00 UTC
US East (Ohio)	us-east-2	03:00–11:00 UTC
US West (N. California)	us-west-1	06:00–14:00 UTC
US West (Oregon)	us-west-2	06:00–14:00 UTC
Africa (Cape Town)	af-south-1	03:00–11:00 UTC
Asia Pacific (Hong Kong)	ap-east-1	06:00–14:00 UTC
Asia Pacific (Hyderabad)	ap-south-2	06:30–14:30 UTC
Asia Pacific (Jakarta)	ap-southeast-3	08:00–16:00 UTC

Region Name	Region	Time Block
Asia Pacific (Malaysia)	ap-southeast-5	09:00–17:00 UTC
Asia Pacific (Melbourne)	ap-southeast-4	11:00–19:00 UTC
Asia Pacific (Mumbai)	ap-south-1	06:00–14:00 UTC
Asia Pacific (New Zealand)	ap-southeast-6	13:00–21:00 UTC
Asia Pacific (Osaka)	ap-northeast-3	22:00–23:59 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00–21:00 UTC
Asia Pacific (Singapore)	ap-southeast-1	14:00–22:00 UTC
Asia Pacific (Sydney)	ap-southeast-2	12:00–20:00 UTC
Asia Pacific (Taipei)	ap-east-2	9:00–17:00 UTC
Asia Pacific (Thailand)	ap-southeast-7	8:00–16:00 UTC
Asia Pacific (Tokyo)	ap-northeast-1	13:00–21:00 UTC
Canada (Central)	ca-central-1	03:00–11:00 UTC
Canada West (Calgary)	ca-west-1	18:00–02:00 UTC
China (Beijing)	cn-north-1	06:00–14:00 UTC
China (Ningxia)	cn-northwest-1	06:00–14:00 UTC
Europe (Frankfurt)	eu-central-1	21:00–05:00 UTC
Europe (Ireland)	eu-west-1	22:00–06:00 UTC

Region Name	Region	Time Block
Europe (London)	eu-west-2	22:00–06:00 UTC
Europe (Milan)	eu-south-1	02:00–10:00 UTC
Europe (Paris)	eu-west-3	23:59–07:29 UTC
Europe (Spain)	eu-south-2	02:00–10:00 UTC
Europe (Stockholm)	eu-north-1	23:00–07:00 UTC
Europe (Zurich)	eu-central-2	02:00–10:00 UTC
Israel (Tel Aviv)	il-central-1	03:00–11:00 UTC
Mexico (Central)	mx-central-1	19:00–3:00 UTC
Middle East (Bahrain)	me-south-1	06:00–14:00 UTC
Middle East (UAE)	me-central-1	05:00–13:00 UTC
South America (São Paulo)	sa-east-1	00:00–08:00 UTC
AWS GovCloud (US-East)	us-gov-east-1	17:00–01:00 UTC
AWS GovCloud (US-West)	us-gov-west-1	06:00–14:00 UTC

Topics

- Adjusting the preferred DB instance maintenance window

Adjusting the preferred DB instance maintenance window

The maintenance window should fall at the time of lowest usage and thus might need modification from time to time. Your DB instance is unavailable during this time only if the system changes,

such as a change in DB instance class, are being applied and require an outage. Your DB instance is unavailable only for the minimum amount of time required to make the necessary changes.

In the following example, you adjust the preferred maintenance window for a DB instance.

For this example, we assume that a DB instance named *mydbinstance* exists and has a preferred maintenance window of "Sun:05:00-Sun:06:00" UTC.

Console

To adjust the preferred maintenance window

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then select the DB instance that you want to modify.
3. Choose **Modify**. The **Modify DB instance** page appears.
4. In the **Maintenance** section, update the maintenance window.

   > **Note**
   >
   > The maintenance window and the backup window for the DB instance cannot overlap. If you enter a value for the maintenance window that overlaps the backup window, an error message appears.

5. Choose **Continue**.

   On the confirmation page, review your changes.
6. To apply the changes to the maintenance window immediately, select **Apply immediately**.
7. Choose **Modify DB instance** to save your changes.

   Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

AWS CLI

To adjust the preferred maintenance window, use the AWS CLI `modify-db-instance` command with the following parameters:

- `--db-instance-identifier`

- `--preferred-maintenance-window`

Example

The following code example sets the maintenance window to Tuesdays from 4:00-4:30AM UTC.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
--db-instance-identifier mydbinstance \
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier mydbinstance ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

RDS API

To adjust the preferred maintenance window, use the Amazon RDS API `ModifyDBInstance` operation with the following parameters:

- `DBInstanceIdentifier`
- `PreferredMaintenanceWindow`

Applying updates to a DB instance

With Amazon RDS, you can choose when to apply maintenance operations. You can decide when Amazon RDS applies updates by using the AWS Management Console, AWS CLI, or RDS API.

Console

To manage an update for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that has a required update.
4. For **Actions**, choose one of the following:

   - **Patch now**
   - **Patch at next window**

   > **Note**
   >
   > If you choose **Patch at next window** and later want to delay the update, you can choose **Defer upgrade**. You can't defer a maintenance action if it has already started. To cancel a maintenance action, modify the DB instance and disable **Auto minor version upgrade**.

AWS CLI

To apply a pending update to a DB instance, use the apply-pending-maintenance-action AWS CLI command.

Example

For Linux, macOS, or Unix:

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
    --apply-action system-update \
    --opt-in-type immediate
```

For Windows:

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

> **Note**
>
> To defer a maintenance action, specify `undo-opt-in` for `--opt-in-type`. You can't specify `undo-opt-in` for `--opt-in-type` if the maintenance action has already started. To cancel a maintenance action, run the modify-db-instance AWS CLI command and specify `--no-auto-minor-version-upgrade`.

To return a list of resources that have at least one pending update, use the describe-pending-maintenance-actions AWS CLI command.

Example

For Linux, macOS, or Unix:

```
aws rds describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

For Windows:

```
aws rds describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

You can also return a list of resources for a DB instance by specifying the `--filters` parameter of the `describe-pending-maintenance-actions` AWS CLI command. The format for the `--filters` command is `Name=filter-name,Value=resource-id,...`.

The following are the accepted values for the `Name` parameter of a filter:

- `db-instance-id` – Accepts a list of DB instance identifiers or Amazon Resource Names (ARNs). The returned list only includes pending maintenance actions for the DB instances identified by these identifiers or ARNs.

- `db-cluster-id` – Accepts a list of DB cluster identifiers or ARNs for Amazon Aurora. The returned list only includes pending maintenance actions for the DB clusters identified by these identifiers or ARNs.

For example, the following example returns the pending maintenance actions for the `sample-instance1` and `sample-instance2` DB instances.

Example

For Linux, macOS, or Unix:

```
aws rds describe-pending-maintenance-actions \
 --filters Name=db-instance-id,Values=sample-instance1,sample-instance2
```

For Windows:

```
aws rds describe-pending-maintenance-actions ^
 --filters Name=db-instance-id,Values=sample-instance1,sample-instance2
```

RDS API

To apply an update to a DB instance, call the Amazon RDS API `ApplyPendingMaintenanceAction` operation.

To return a list of resources that have at least one pending update, call the Amazon RDS API `DescribePendingMaintenanceActions` operation.

Operating system updates for RDS DB instances

RDS for Db2, RDS for MariaDB, RDS for MySQL, RDS for PostgreSQL, RDS for SQL Server, RDS for Oracle and RDS Custom for SQL Server DB instances occasionally require operating system updates. Amazon RDS upgrades the operating system to a newer version to improve database performance and customers' overall security posture. Typically, the updates take about 10 minutes. Operating system updates don't change the DB engine version or DB instance class of a DB instance.

Operating system updates can be either optional or mandatory:

- An **optional update** can be applied at any time. While these updates are optional, we recommend that you apply them periodically to keep your RDS fleet up to date. RDS *does not* apply these updates automatically.

  To be notified when a new, optional operating system patch becomes available, you can subscribe to RDS-EVENT-0230 in the security patching event category. For information about subscribing to RDS events, see Subscribing to Amazon RDS event notification.

  > **Note**
  >
  > `RDS-EVENT-0230` doesn't apply to operating system distribution upgrades.

- A **mandatory update** is required and has an apply date. Plan to schedule your update before this apply date. After the specified apply date, Amazon RDS automatically upgrades the operating system for your DB instance to the latest version during one of your assigned maintenance windows.

> **Note**
>
> Staying current on all optional and mandatory updates might be required to meet various compliance obligations. We recommend that you apply all updates made available by RDS routinely during your maintenance windows.

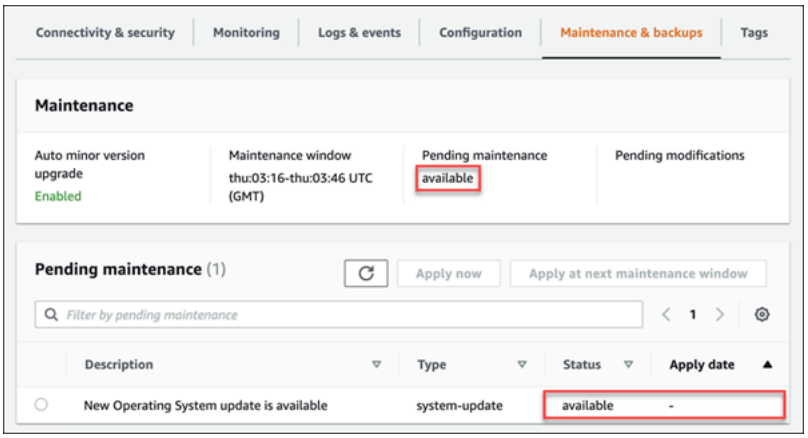
You can use the AWS Management Console or the AWS CLI to get information about the type of operating system upgrade.

Console

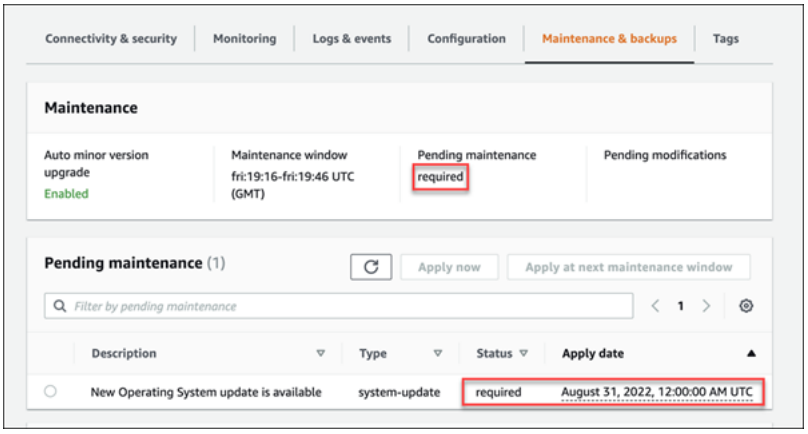
To get update information using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then select the DB instance.
3. Choose **Maintenance & backups**.
4. In the **Pending maintenance** section, find the operating system update, and check the **Status** value.

In the AWS Management Console, an optional update has its maintenance **Status** set to **available** and doesn't have an **Apply date**, as shown in the following image.



A mandatory update has its maintenance **Status** set to **required** and has an **Apply date**, as shown in the following image.



AWS CLI

To get update information from the AWS CLI, use the describe-pending-maintenance-actions command.

```
aws rds describe-pending-maintenance-actions
```

A mandatory operating system update includes an `AutoAppliedAfterDate` value and a `CurrentApplyDate` value. An optional operating system update doesn't include these values.

The following output shows a mandatory operating system update.

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb1",
```

```
  "PendingMaintenanceActionDetails": [
    {
      "Action": "system-update",
      "AutoAppliedAfterDate": "2022-08-31T00:00:00+00:00",
      "CurrentApplyDate": "2022-08-31T00:00:00+00:00",
      "Description": "New Operating System update is available"
    }
  ]
}
```

The following output shows an optional operating system update.

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb2",
  "PendingMaintenanceActionDetails": [
    {
      "Action": "system-update",
      "Description": "New Operating System update is available"
    }
  ]
}
```

Availability of operating system updates

Operating system updates are specific to DB engine version and DB instance class. Therefore, DB instances receive or require updates at different times. When an operating system update is available for your DB instance based on its engine version and instance class, the update appears in the console. It can also be viewed by running the describe-pending-maintenance-actions AWS CLI command or by calling the DescribePendingMaintenanceActions RDS API operation. If an update is available for your instance, you can update your operating system by following the instructions in Applying updates to a DB instance .

Using AWS Organizations upgrade rollout policy for automatic minor version upgrades

Amazon RDS supports AWS Organizations upgrade rollout policy to manage automatic minor version upgrades across multiple database resources and AWS accounts. This policy helps you implement a controlled upgrade strategy for your instances by:

How upgrade rollout policy works

When a new minor engine version becomes eligible for automatic upgrade, the policy controls the upgrade sequence based on defined orders:

- Resources marked as [first] (typically development environments) become eligible for upgrades during their maintenance windows.
- After a designated bake time, resources marked as [second] become eligible.
- After another designated bake time, resources marked as [last] (typically production environments) become eligible.
- Monitoring upgrade progress through AWS Health notifications.

You can define your upgrade orders by:

- Account-level policies – Apply to all eligible resources in specified accounts.
- Resource tags – Apply to specific resources based on tags.

> **Note**
>
> Resources not configured with an upgrade policy or excluded from the policy automatically receive an upgrade order of [second].

Prerequisites

- Your AWS account must be part of an organization in Organizations with upgrade rollout policy enabled
- Enable automatic minor version upgrades for your instances
- Tags are not strictly required for upgrade rollout policy. If you want to define specific upgrade orders for different environments (for example, development, test, QA, production), you can use tags. If you don't include tag settings in your policy, all resources under that policy follow the default upgrade order.

Prerequisites

- Your AWS account must be part of an organization in Organizations with upgrade rollout policy enabled
- Enable automatic minor version upgrades for your instances

- Tag your resources to identify their environment (for example, development, test, production)

To tag your resources

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the instance you want to tag.
4. Choose **Actions**, then choose **Manage tags**.
5. Choose **Add tag**.
6. Enter your tag key (for example, 'Environment') and value (for example, 'Development')
7. Choose **Add tag**, then **Save**.

You can also add tags using the AWS CLI:

```
aws rds add-tags-to-resource \
    --resource-name arn:aws:rds:region:account-number:db:instance-name \
    --tags Key=Environment,Value=Development
```

Upgrade order and phases

The upgrade rollout policy supports three upgrade orders:

- [first] - Typically used for development or testing environments
- [second] - Typically used for QA environments. Default order for resources not specifically configured
- [last] - Usually reserved for production environments

When a new minor engine version becomes eligible for automatic upgrade:

- Resources with upgrade order [first] become eligible for upgrades during their configured maintenance windows.
- After a designated bake time, resources with upgrade order [second] become eligible for upgrades during their maintenance windows.

- After another designated bake time, resources with upgrade order [last] become eligible for upgrades during their maintenance windows.
- The automatic minor version upgrade campaign closes after all eligible resources with upgrade orders [first], [second], and [last] have been upgraded, or when the campaign reaches its scheduled end date, whichever comes first.

> **Note**
>
> All automatic minor version upgrades are performed during each instance's configured maintenance window to minimize potential impact to your applications.

Observability

AWS Health and monitoring

You receive AWS health notifications:

- Before the start of an automatic minor version upgrade campaign
- Between each phase transition to help track and monitor upgrade progress
- Progress updates showing the number of resources upgraded across your fleet in the AWS Health console

Amazon RDS event notifications:

- Notifications for resources enabled for automatic minor version upgrades, including:
  - When your resource becomes eligible for upgrade based on its upgrade order ([first], [second], or [last])
  - Scheduled upgrade timeline during the maintenance window
  - Individual database upgrade start and completion status
- Subscribe to these events through Amazon EventBridge0 for automated monitoring

Considerations

Some considerations to keep in mind:

- The policy applies to all future automatic minor version upgrade campaigns, including policy changes made during active campaigns.
- If you join an ongoing upgrade campaign, your resources follow the current running upgrade order and do not wait for a configured policy.
- Resources not configured with an upgrade policy automatically receive an upgrade order of [second].
- The policy provides validation periods between upgrade phases before proceeding to the next phase.
- Changes to either the policy or resource tags require time to propagate before the new upgrade order is applied.
- The policy applies only to Amazon RDS resources with automatic minor version upgrades enabled.
- If you detect an issue within an environment, you can turn off automatic minor version upgrades for subsequent environments or use the validation period to resolve issues before upgrades proceed to the next upgrade order.

> **Note**
>
> This feature supports automatic minor version upgrades for Oracle Database engine versions released after January 2026.

For more information about tagging RDS resources, see Tagging Amazon RDS resources. For detailed instructions on setting up and using upgrade rollout policy, see Getting started with AWS Organizations in the *AWS Organizations User Guide*.

Upgrading a DB instance engine version

Amazon RDS provides newer versions of each supported database engine so you can keep your DB instance up-to-date. Newer versions can include bug fixes, security enhancements, and other improvements for the database engine. When Amazon RDS supports a new version of a database engine, you can choose how and when to upgrade your database DB instances.

There are two kinds of upgrades: major version upgrades and minor version upgrades. In general, a *major engine version upgrade* can introduce changes that are not compatible with existing applications. In contrast, a *minor version upgrade* includes only changes that are backward-compatible with existing applications.

For Multi-AZ DB clusters, major version upgrades are only supported for RDS for PostgreSQL. Minor version upgrades are supported for all engines that support Multi-AZ DB clusters. For more information, see the section called “Upgrading a Multi-AZ DB cluster”.

The version numbering sequence is specific to each database engine. For example, RDS for MySQL 5.7 and 8.0 are major engine versions and upgrading from any 5.7 version to any 8.0 version is a major version upgrade. RDS for MySQL version 5.7.22 and 5.7.23 are minor versions and upgrading from 5.7.22 to 5.7.23 is a minor version upgrade.

> **Important**
>
> You can't modify a DB instance when it is being upgraded. During an upgrade, the DB instance status is `upgrading`.

For more information about major and minor version upgrades for a specific DB engine, see the following documentation for your DB engine:

- Upgrades of the MariaDB DB engine
- Upgrades of the Microsoft SQL Server DB engine
- Upgrades of the RDS for MySQL DB engine
- Upgrading the RDS for Oracle DB engine
- Upgrades of the RDS for PostgreSQL DB engine

For major version upgrades, you must manually modify the DB engine version through the AWS Management Console, AWS CLI, or RDS API. For minor version upgrades, you can manually modify the engine version, or you can choose to enable the **Auto minor version upgrade** option.

> **Note**
>
> Database engine upgrades require downtime. You can minimize the downtime required for DB instance upgrade by using a blue/green deployment. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

Topics

- Manually upgrading the engine version
- Automatically upgrading the minor engine version

Manually upgrading the engine version

To manually upgrade the engine version of a DB instance, you can use the AWS Management Console, the AWS CLI, or the RDS API.

Console

To upgrade the engine version of a DB instance by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to upgrade.
3. Choose **Modify**. The **Modify DB instance** page appears.
4. For **DB engine version**, choose the new version.
5. Choose **Continue** and check the summary of modifications.
6. Decide when to schedule your upgrade:
   - To put the changes into the pending modifications queue, choose **Apply during the next scheduled maintenance window**. During the next maintenance window, RDS applies any pending changes in the queue.

- To apply the changes immediately, choose **Apply immediately**. Choosing this option can cause an outage in some cases. For more information, see Using the schedule modifications setting.

7. On the confirmation page, review your changes. If they are correct, choose **Modify DB instance** to save your changes.

   Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

AWS CLI

To upgrade the engine version of a DB instance, use the CLI modify-db-instance command. Specify the following parameters:

- `--db-instance-identifier` – the name of the DB instance.
- `--engine-version` – the version number of the database engine to upgrade to.

  For information about valid engine versions, use the AWS CLI describe-db-engine-versions command.
- `--allow-major-version-upgrade` – to upgrade the major version.
- `--no-apply-immediately` – to apply changes during the next maintenance window. To apply changes immediately, use `--apply-immediately`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --engine-version new_version \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --engine-version new_version ^
    --allow-major-version-upgrade ^
```

```
    --no-apply-immediately
```

RDS API

To upgrade the engine version of a DB instance, use the ModifyDBInstance action. Specify the following parameters:

- `DBInstanceIdentifier` – the name of the DB instance, for example *`mydbinstance`*.
- `EngineVersion` – the version number of the database engine to upgrade to. For information about valid engine versions, use the DescribeDBEngineVersions operation.
- `AllowMajorVersionUpgrade` – whether to allow a major version upgrade. To do so, set the value to `true`.
- `ApplyImmediately` – whether to apply changes immediately or during the next maintenance window. To apply changes immediately, set the value to `true`. To apply changes during the next maintenance window, set the value to `false`.

Automatically upgrading the minor engine version

Automatic minor version upgrades periodically update your database to recent database engine versions. However, the upgrade might not always include the latest database engine version. If you need to keep your databases on specific versions at particular times, we recommend that you manually upgrade to the database versions that you need according to your required schedule. In cases of critical security issues or when a version reaches its end-of-support date, Amazon RDS might apply a minor version upgrade even if you haven't enabled the **Auto minor version upgrade** option. For more information, see the upgrade documentation for your specific database engine.

- Automatic minor version upgrades for RDS for PostgreSQL
- Automatic minor version upgrades for RDS for MySQL
- Automatic minor version upgrades for RDS for MariaDB
- Oracle minor version upgrades
- Upgrades of the Microsoft SQL Server DB engine
- Db2 on Amazon RDS versions

Topics

- How automatic minor version upgrades work

- Turning on automatic minor version upgrades
- Determining the availability of maintenance updates
- Finding automatic minor version upgrade targets

How automatic minor version upgrades work

The *upgrade target* is the DB engine version to which Amazon RDS upgrades your database. A minor engine version is designated as the upgrade target when the following conditions are met:

- The database is running a minor version of the DB engine that is lower than the target minor engine version.

  You can find the current engine version for your DB instance by looking on the **Configuration** tab of the database details page or running the CLI command `describe-db-instances`.
- The database has automatic minor version upgrade enabled.

RDS schedules the upgrade to run automatically in the maintenance window. During the upgrade, RDS does the following:

1. Runs a system precheck to make sure the database is healthy and ready to be upgraded
2. Upgrades the DB engine to the target minor engine version
3. Runs post-upgrade checks
4. Marks the database upgrade as complete

Automatic upgrades incur downtime. The length of the downtime depends on various factors, including the DB engine type and the size of the database.

Turning on automatic minor version upgrades

You can control whether auto minor version upgrade is enabled for a DB instance when you perform the following tasks:

- Creating a DB instance
- Modifying a DB instance
- Creating a read replica
- Restoring a DB instance from a snapshot

- Restoring a DB instance to a specific time
- Importing a DB instance from Amazon S3 (for a MySQL backup on Amazon S3)

When you perform these tasks, you can control whether auto minor version upgrade is enabled for the DB instance in the following ways:

- Using the console, set the **Auto minor version upgrade** option.
- Using the AWS CLI, set the `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` option.
- Using the RDS API, set the `AutoMinorVersionUpgrade` parameter.

Determining the availability of maintenance updates

To determine whether a maintenance update, such as a DB engine version upgrade, is available for your DB instance, you can use the console, AWS CLI, or RDS API. You can also upgrade the DB engine version manually and adjust the maintenance window. For more information, see Maintaining a DB instance.

Finding automatic minor version upgrade targets

You can use the following AWS CLI command to determine the current automatic minor upgrade target version for a specified minor DB engine version in a specific AWS Region. You can find the possible `--engine` values for this command in the description for the `Engine` parameter in CreateDBInstance.

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine engine \
--engine-version minor-version \
--region region \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
```

```
--engine engine ^
--engine-version minor-version ^
--region region ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output text
```

For example, the following AWS CLI command determines the automatic minor upgrade target for MySQL minor version 8.0.11 in the US East (Ohio) AWS Region (us-east-2).

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mysql \
--engine-version 8.0.11 \
--region us-east-2 \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output table
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mysql ^
--engine-version 8.0.11 ^
--region us-east-2 ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output table
```

Your output is similar to the following.

```
---------------------------------
|    DescribeDBEngineVersions    |
+--------------+-----------------+
|  AutoUpgrade |  EngineVersion  |
+--------------+-----------------+
|  False       |  8.0.15         |
|  False       |  8.0.16         |
|  False       |  8.0.17         |
|  False       |  8.0.19         |
|  False       |  8.0.20         |
```

```
|  False       |  8.0.21         |
|  True        |  8.0.23         |
|  False       |  8.0.25         |
+--------------+-----------------+
```

In this example, the `AutoUpgrade` value is `True` for MySQL version 8.0.23. So, the automatic minor upgrade target is MySQL version 8.0.23, which is highlighted in the output.

Important

If you plan to migrate an RDS for PostgreSQL DB instance to an Aurora PostgreSQL DB cluster soon, we strongly recommend that you turn off auto minor version upgrades for the DB instance early during planning. Migration to Aurora PostgreSQL might be delayed if the RDS for PostgreSQL version isn't yet supported by Aurora PostgreSQL. For information about Aurora PostgreSQL versions, see Engine versions for Amazon Aurora PostgreSQL.

Renaming a DB instance

You can rename a DB instance by using the AWS Management Console, the AWS CLI `modify-db-instance` command, or the Amazon RDS API `ModifyDBInstance` action. Renaming a DB instance can have far-reaching effects. The following is a list of considerations before you rename a DB instance.

- When you rename a DB instance, the endpoint for the DB instance changes, because the URL includes the name you assigned to the DB instance. You should always redirect traffic from the old URL to the new one.
- When you rename a DB instance, the old DNS name that was used by the DB instance is immediately deleted, although it could remain cached for a few minutes. The new DNS name for the renamed DB instance becomes effective in about 10 minutes. The renamed DB instance is not available until the new name becomes effective.
- You cannot use an existing DB instance name when renaming an instance.
- All read replicas associated with a DB instance remain associated with that instance after it is renamed. For example, suppose you have a DB instance that serves your production database and the instance has several associated read replicas. If you rename the DB instance and then replace it in the production environment with a DB snapshot, the DB instance that you renamed will still have the read replicas associated with it.
- Metrics and events associated with the name of a DB instance are maintained if you reuse a DB instance name. For example, if you promote a read replica and rename it to be the name of the previous primary DB instance, the events and metrics associated with the primary DB instance are associated with the renamed instance.
- DB instance tags remain with the DB instance, regardless of renaming.
- DB snapshots are retained for a renamed DB instance.

> **Note**
>
> A DB instance is an isolated database environment running in the cloud. A DB instance can host multiple databases, or a single Oracle database with multiple schemas. For information about changing a database name, see the documentation for your DB engine.

Renaming to replace an existing DB instance

The most common reasons for renaming a DB instance are that you are promoting a read replica or you are restoring data from a DB snapshot or point-in-time recovery (PITR). By renaming the database, you can replace the DB instance without having to change any application code that references the DB instance. In these cases, you would do the following:

1. Stop all traffic going to the primary DB instance. This can involve redirecting traffic from accessing the databases on the DB instance or some other way you want to use to prevent traffic from accessing your databases on the DB instance.
2. Rename the primary DB instance to a name that indicates it is no longer the primary DB instance as described later in this topic.
3. Create a new primary DB instance by restoring from a DB snapshot or by promoting a read replica, and then give the new instance the name of the previous primary DB instance.
4. Associate any read replicas with the new primary DB instance.

If you delete the old primary DB instance, you are responsible for deleting any unwanted DB snapshots of the old primary DB instance.

For information about promoting a read replica, see Promoting a read replica to be a standalone DB instance.

> **Important**
>
> The DB instance is rebooted when it is renamed. For RDS for SQL Server Multi-AZ instances with AlwaysOn or Mirroring option enabled, a failover is expected when instance is rebooted after the rename operation.

Console

To rename a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to rename.

4. Choose **Modify**.
5. In **Settings**, enter a new name for **DB instance identifier**.
6. Choose **Continue**.
7. To apply the changes immediately, choose **Apply immediately**. Choosing this option can cause an outage in some cases. For more information, see Modifying an Amazon RDS DB instance.
8. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** to save your changes.

   Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

AWS CLI

To rename a DB instance, use the AWS CLI command `modify-db-instance`. Provide the current `--db-instance-identifier` value and `--new-db-instance-identifier` parameter with the new name of the DB instance.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier DBInstanceIdentifier \
    --new-db-instance-identifier NewDBInstanceIdentifier
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier DBInstanceIdentifier ^
    --new-db-instance-identifier NewDBInstanceIdentifier
```

RDS API

To rename a DB instance, call Amazon RDS API operation `ModifyDBInstance` with the following parameters:

- `DBInstanceIdentifier` — existing name for the instance
- `NewDBInstanceIdentifier` — new name for the instance

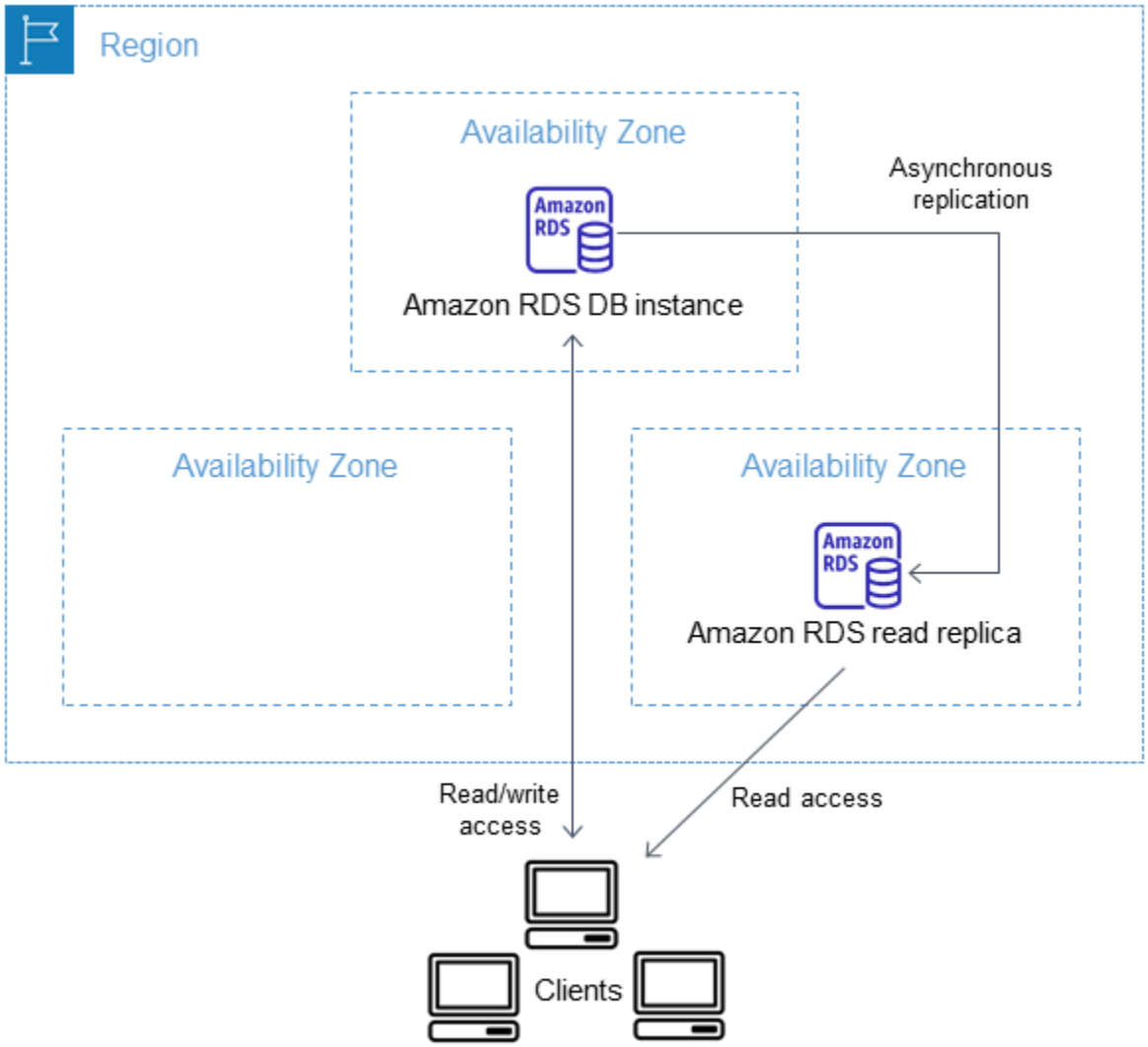
Working with DB instance read replicas

A *read replica* is a read-only copy of a DB instance. You can reduce the load on your primary DB instance by routing queries from your applications to the read replica. In this way, you can elastically scale out beyond the capacity constraints of a single DB instance for read-heavy database workloads.

To create a read replica from a source DB instance, Amazon RDS uses the built-in replication features of the DB engine. For information about using read replicas with a specific engine, see the following sections:

- Working with replicas for Amazon RDS for Db2
- Working with MariaDB read replicas
- Working with read replicas for Microsoft SQL Server in Amazon RDS
- Working with MySQL read replicas
- Working with read replicas for Amazon RDS for Oracle
- Working with read replicas for Amazon RDS for PostgreSQL

After you create a read replica from a source DB instance, the source becomes the primary DB instance. When you make updates to the primary DB instance, Amazon RDS copies them asynchronously to the read replica. The following diagram shows a source DB instance replicating to a read replica in a different Availability Zone (AZ). Clients have read/write access to the primary DB instance and read-only access to the replica.



Read replicas are billed as standard DB instances at the same rates as the DB instance class used for the replica. You aren't charged for the data transfer incurred in replicating data between the source DB instance and a read replica within the same AWS Region. For more information, see Cross-Region replication costs and DB instance billing for Amazon RDS .

Topics

- Overview of Amazon RDS read replicas
- Differences between read replicas for DB engines
- Creating a read replica
- Promoting a read replica to be a standalone DB instance
- Monitoring read replication
- Creating a read replica in a different AWS Region

Overview of Amazon RDS read replicas

The following sections discuss DB *instance* read replicas. For information about Multi-AZ DB *cluster* read replicas, see the section called “Working with Multi-AZ DB cluster read replicas”.

Topics

- Use cases for read replicas
- How read replicas work
- Read replicas in a Multi-AZ deployment
- Cross-Region read replicas
- Read replica storage types
- Restrictions for creating a replica from a replica
- Considerations when deleting replicas

Use cases for read replicas

Deploying one or more read replicas for a given source DB instance might make sense in a variety of scenarios, including the following:

- Scaling beyond the compute or I/O capacity of a single DB instance for read-heavy database workloads. You can direct this excess read traffic to one or more read replicas.
- Serving read traffic while the source DB instance is unavailable. In some cases, your source DB instance might not be able to take I/O requests, for example due to I/O suspension for backups or scheduled maintenance. In these cases, you can direct read traffic to your read replicas. For this use case, keep in mind that the data on the read replica might be "stale" because the source DB instance is unavailable.
- Business reporting or data warehousing scenarios where you might want business reporting queries to run against a read replica, rather than your production DB instance.
- Implementing disaster recovery. You can promote a read replica to a standalone instance as a disaster recovery solution if the primary DB instance fails.

How read replicas work

When you create a read replica, you specify an existing DB instance as the source. Then Amazon RDS takes a snapshot of the source instance and creates a read-only instance from the snapshot.

Amazon RDS then uses the asynchronous replication method for the DB engine to update the read replica whenever there is a change to the primary DB instance.

The read replica operates as a DB instance that allows only read-only connections. Exceptions are the RDS for Db2 and the RDS for Oracle DB engines, which support replica databases in standby mode and mounted mode, respectively. A standby replica and mounted replica don't accept user connections and so can't serve read-only workloads. The primary use for standby replicas and mounted replicas is cross-Region disaster recovery. For more information, see Working with replicas for Amazon RDS for Db2 and Working with read replicas for Amazon RDS for Oracle.

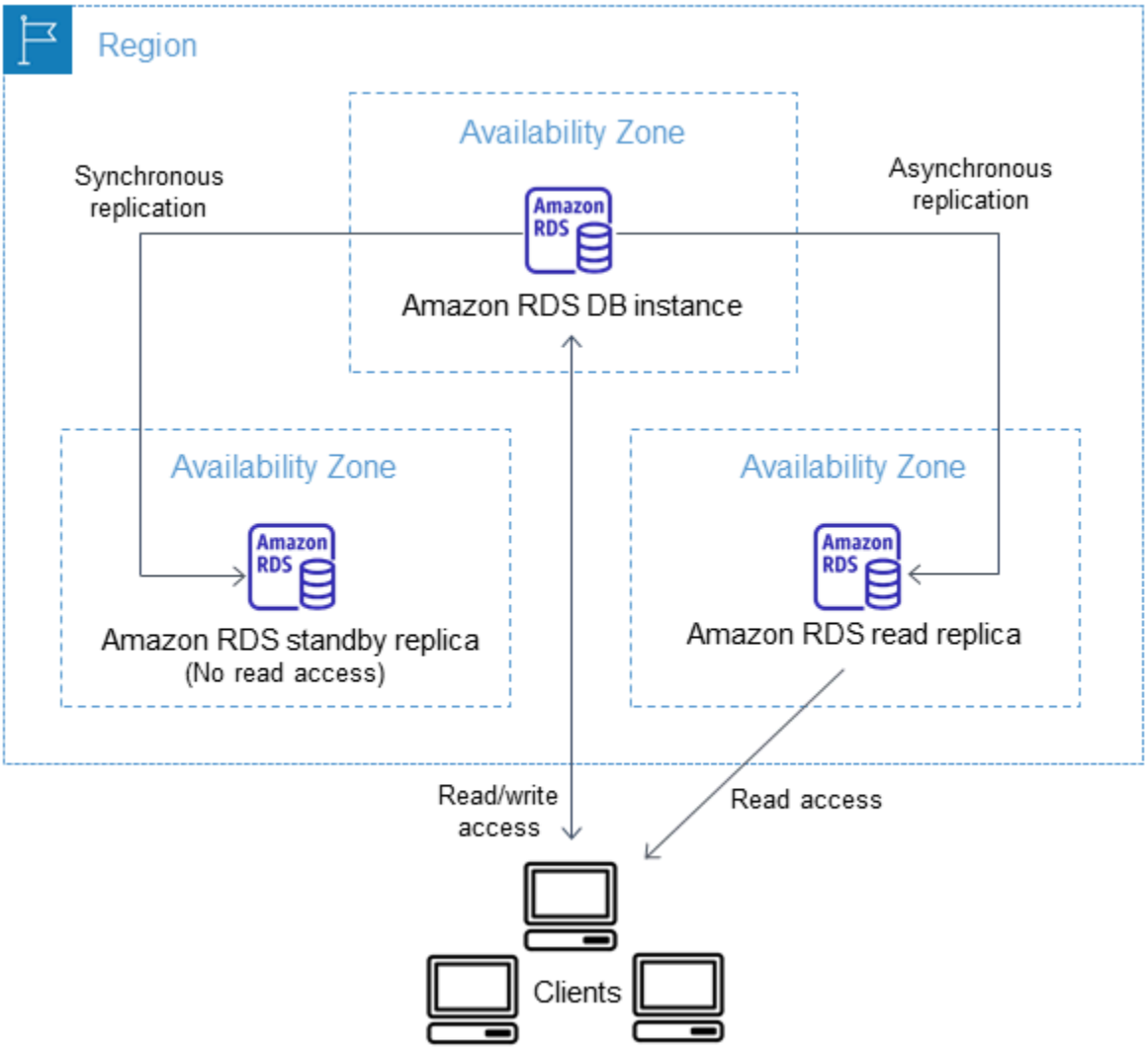
Applications connect to a read replica just as they do to any DB instance. Amazon RDS replicates all databases from the source DB instance.

You must manually create read replicas. RDS doesn't support autoscaling of read replicas, which is the automatic add or removing of read replicas as read demand changes.

Read replicas in a Multi-AZ deployment

You can configure a read replica for a DB instance that also has a standby replica configured for high availability in a Multi-AZ deployment. Replication with the standby replica is synchronous. Unlike a read replica, a standby replica can't serve read traffic.

In the following scenario, clients have read/write access to a primary DB instance in one AZ. The primary instance copies updates asynchronously to a read replica in a second AZ and also copies them synchronously to a standby replica in a third AZ. Clients have read access only to the read replica.



For more information about high availability and standby replicas, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

Cross-Region read replicas

In some cases, a read replica resides in a different AWS Region from its primary DB instance. In these cases, Amazon RDS sets up a secure communications channel between the primary DB instance and the read replica. Amazon RDS establishes any AWS security configurations needed to enable the secure channel, such as adding security group entries. For more information about cross-Region read replicas, see Creating a read replica in a different AWS Region.

The information in this chapter applies to creating Amazon RDS read replicas either in the same AWS Region as the source DB instance, or in a separate AWS Region. The following information

doesn't apply to setting up replication with an instance that is running on an Amazon EC2 instance or that is on-premises.

Read replica storage types

By default, a read replica is created with the same storage type as the source DB instance. However, you can create a read replica that has a different storage type from the source DB instance based on the options listed in the following table.

Source DB instance storage type	Source DB instance storage allocation	Read replica storage type options
Provisioned IOPS	100 GiB–64 TiB	Provisioned IOPS, General Purpose, Magnetic
General Purpose	100 GiB–64 TiB	Provisioned IOPS, General Purpose, Magnetic
General Purpose	<100 GiB	General Purpose, Magnetic
Magnetic	100 GiB–6 TiB	Provisioned IOPS, General Purpose, Magnetic
Magnetic	<100 GiB	General Purpose, Magnetic

Note

When you increase the allocated storage of a read replica, it must be by at least 10 percent. If you try to increase the value by less than 10 percent, you get an error.

Restrictions for creating a replica from a replica

Amazon RDS doesn't support circular replication. You can't configure a DB instance to serve as a replication source for an existing DB instance. You can only create a new read replica from an existing DB instance. For example, if **`MySourceDBInstance`** replicates to **`ReadReplica1`**, you can't configure **`ReadReplica1`** to replicate back to **`MySourceDBInstance`**.

For RDS for MariaDB and RDS for MySQL, and for certain versions of RDS for PostgreSQL, you can create a read replica from an existing read replica. For example, you can create new read replica **`ReadReplica2`** from existing replica **`ReadReplica1`**. For RDS for Db2, RDS for Oracle, and RDS for SQL Server, you can't create a read replica from an existing read replica.

Considerations when deleting replicas

RDS doesn't support autoscaling of read replicas. Thus, RDS won't increase the number of replicas when demand increases or decrease the number of replicas when demand decreases. If you no longer need read replicas, manually delete them using the same mechanisms for deleting a DB instance. If you delete a source DB instance without deleting its read replicas in the same AWS Region, each replica is promoted to a standalone DB instance.

For information about deleting a DB instance, see Deleting a DB instance. For information about read replica promotion, see Promoting a read replica to be a standalone DB instance. For information related to deleting the source DB instance for a cross-Region read replica, see Cross-Region replication considerations.

Differences between read replicas for DB engines

Because Amazon RDS DB engines implement replication differently, there are several significant differences that you should know about.

Db2

Replicas for RDS for Db2 have the following features and behaviors:

- **Replication method** – Physical replication.
- **Transaction logs purge** – RDS for Db2 purges the logs from the primary DB instance when the following conditions have been met:
  - The logs are at least two hours old.
  - The archive log retention hours setting has passed.
  - RDS for Db2 successfully replicated the logs to all the replica DB instances.

  This applies to both same AWS Region DB instances and cross-Region DB instances. For information about setting archive log retention hours, see rdsadmin.set_archive_log_retention.
- **Writable replicas** – A Db2 replica is a physical copy, and Db2 doesn't allow for writes in a replica. You can promote the replica to make it writable. The promoted replica has the replicated data to the point when the request was made to promote it.

- **Backups** – Automatic backups and manual snapshots are supported on RDS for Db2 replicas.
- **Parallel replication** – Archive log data is always transmitted in parallel from the primary database to all of its replicas.
- **Standby state** – The primary use for standby replicas is cross-Region disaster recovery. For information, see Working with replicas for Amazon RDS for Db2.

MariaDB and MySQL

Read replicas for RDS for MariaDB and RDS for MySQL have the following features and behaviors:

- **Replication method** – Logical replication.
- **Transaction logs purge** – RDS for MariaDB and RDS for MySQL keep any binary logs that haven't been applied.
- **Writable replicas** – You can enable the MariaDB or MySQL read replica to be writable.
- **Backups** – Automatic backups and manual snapshots are supported on RDS for MariaDB or RDS for MySQL read replicas.
- **Parallel replication** – All supported MariaDB and MySQL versions allow for parallel replication threads.
- **Mounted state** – Not supported.

Oracle

Read replicas for RDS for Oracle have the following features and behaviors:

- **Replication method** – Physical replication.
- **Transaction logs purge** – If a primary DB instance has no cross-Region read replicas, Amazon RDS for Oracle keeps a minimum of two hours of transaction logs on the source DB instance. Logs are purged from the source DB instance after two hours or after the archive log retention hours setting has passed, whichever is longer. Logs are purged from the read replica after the archive log retention hours setting has passed only if they have been successfully applied to the database.

  In some cases, a primary DB instance might have one or more cross-Region read replicas. If so, Amazon RDS for Oracle keeps the transaction logs on the source DB instance until they have been transmitted and applied to all cross-Region read replicas.

For information about setting archive log retention hours, see Retaining archived redo logs.

- **Writable replicas** – An Oracle read replica is a physical copy, and Oracle doesn't allow for writes in a read replica. You can promote the read replica to make it writable. The promoted read replica has the replicated data to the point when the request was made to promote it.
- **Backups** – Automatic backups and manual snapshots are supported on RDS for Oracle read replicas.
- **Parallel replication** – Redo log data is always transmitted in parallel from the primary database to all of its read replicas.
- **Mounted state** – The primary use for mounted replicas is cross-Region disaster recovery. An Active Data Guard license isn't required for mounted replicas. For more information, see Working with read replicas for Amazon RDS for Oracle.

PostgreSQL

Read replicas for RDS for PostgreSQL have the following features and behaviors:

- **Replication method** – Physical replication.
- **Transaction logs purge** – PostgreSQL has the parameter `wal_keep_segments` that dictates how many write ahead log (WAL) files are kept to provide data to the read replicas. The parameter value specifies the number of logs to keep.
- **Writable replicas** – A PostgreSQL read replica is a physical copy, and PostgreSQL doesn't allow for a read replica to be made writable.
- **Backups** – Manual snapshots are supported for RDS for PostgreSQL read replicas. Automated backups for read replicas are supported for RDS for PostgreSQL 14.1 and higher versions only. You can't turn on automated backups for PostgreSQL read replicas for RDS for PostgreSQL versions earlier than 14.1. For RDS for PostgreSQL 13 and earlier versions, create a snapshot from a read replica if you want a backup of it.
- **Parallel replication** – PostgreSQL has a single process handling replication.
- **Mounted state** – Not supported.

SQL Server

Read replicas for RDS for SQL Server have the following features and behaviors:

- **Replication method** – Physical replication.

- **Transaction logs purge** – The Virtual Log File (VLF) of the transaction log file on the primary replica can be truncated after it is no longer required for the secondary replicas.

  The VLF can only be marked as inactive when the log records have been hardened in the replicas. Regardless of how fast the disk subsystems are in the primary replica, the transaction log will keep the VLFs until the slowest replica has hardened it.
- **Writable replicas** – A SQL Server read replica is a physical copy and also doesn't allow for writes. You can promote the read replica to make it writable. The promoted read replica has the replicated data up to the point when the request was made to promote it.
- **Backups** – Automatic backups and manual snapshots aren't supported on RDS for SQL Server read replicas.
- **Parallel replication** – Redo log data is always transmitted in parallel from the primary database to all of its read replicas.
- **Mounted state** – Not supported.

Creating a read replica

You can create a read replica from an existing DB instance using the AWS Management Console, AWS CLI, or RDS API. You create a read replica by specifying `SourceDBInstanceIdentifier`, which is the DB instance identifier of the source DB instance that you want to replicate from.

When you create a read replica, Amazon RDS takes a DB snapshot of your source DB instance and begins replication. The source DB instance experiences a very brief I/O suspension when the DB snapshot operation begins. The I/O suspension typically lasts about one second. You can avoid the I/O suspension if the source DB instance is a Multi-AZ deployment, because in that case the snapshot is taken from the secondary DB instance.

An active, long-running transaction can slow the process of creating the read replica. We recommend that you wait for long-running transactions to complete before creating a read replica. If you create multiple read replicas in parallel from the same source DB instance, Amazon RDS takes only one snapshot at the start of the first create action.

When creating a read replica, there are a few things to consider. First, you must enable automatic backups on the source DB instance by setting the backup retention period to a value other than 0. This requirement also applies to a read replica that is the source DB instance for another read replica. To enable automatic backups on an RDS for MySQL read replica, first create the read replica, then modify the read replica to enable automatic backups.

> **Note**
>
> Within an AWS Region, we strongly recommend that you create all read replicas in the same virtual private cloud (VPC) based on Amazon VPC as the source DB instance. If you create a read replica in a different VPC from the source DB instance, classless inter-domain routing (CIDR) ranges can overlap between the replica and the RDS system. CIDR overlap makes the replica unstable, which can negatively impact applications connecting to it.
> If you receive an error when creating the read replica, choose a different destination DB subnet group. For more information, see Working with a DB instance in a VPC.
> There is no direct way to create a read replica in another AWS account using the console or AWS CLI.

Console

To create a read replica from a source DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to use as the source for a read replica.
4. For **Actions**, choose **Create read replica**.
5. For **DB instance identifier**, enter a name for the read replica.
6. Choose your instance configuration. We recommend that you use the same or larger DB instance class and storage type as the source DB instance for the read replica.
7. For **AWS Region**, specify the destination Region for the read replica.
8. For **Storage**, specify the allocated storage size and whether you want to use storage autoscaling.

   If your source DB instance isn't on the latest storage configuration, the **Upgrade storage file system configuration** option is available. You can enable this setting to upgrade the storage file system of the read replica to the preferred configuration. For more information, see the section called “Upgrading the storage file system”.
9. For **Availability**, choose whether to create a standby of your replica in another Availability Zone for failover support for the replica.

> **Note**
>
> Creating your read replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ DB instance.

10. Specify other DB instance settings. For information about each available setting, see Settings for DB instances.
11. To create an encrypted read replica, expand **Additional configuration** and specify the following settings:

    a. Choose **Enable encryption**.

    b. For **AWS KMS key**, choose the AWS KMS key identifier of the KMS key.

> **Note**
>
> The source DB instance must be encrypted. To learn more about encrypting the source DB instance, see Encrypting Amazon RDS resources.

12. Choose **Create read replica**.

After the read replica is created, you can see it on the **Databases** page in the RDS console. It shows **Replica** in the **Role** column.

AWS CLI

To create a read replica from a source DB instance, use the AWS CLI command create-db-instance-read-replica. This example also sets the allocated storage size, enables storage autoscaling, and upgrades the file system to the preferred configuration.

You can specify other settings. For information about each setting, see Settings for DB instances.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance-read-replica \
    --db-instance-identifier myreadreplica \
```

```
    --source-db-instance-identifier mydbinstance \
    --allocated-storage 100 \
    --max-allocated-storage 1000 \
    --upgrade-storage-config
```

For Windows:

```
aws rds create-db-instance-read-replica ^
    --db-instance-identifier myreadreplica ^
    --source-db-instance-identifier mydbinstance ^
    --allocated-storage 100 ^
    --max-allocated-storage 1000 ^
    --upgrade-storage-config
```

RDS API

To create a read replica from a source Db2, MySQL, MariaDB, Oracle, PostgreSQL, or SQL Server DB instance, call the Amazon RDS API CreateDBInstanceReadReplica operation with the following required parameters:

- `DBInstanceIdentifier`
- `SourceDBInstanceIdentifier`

> **Note**
>
> To create an RDS for Db2 standby replica, set the optional `ReplicaMode` operation to `mounted`.

Promoting a read replica to be a standalone DB instance

You can promote a read replica into a standalone DB instance. If a source DB instance has several read replicas, promoting one of the read replicas to a DB instance has no effect on the other replicas.

When you promote a read replica, RDS reboots the DB instance before making it available. The promotion process can take several minutes or longer to complete, depending on the size of the read replica.



Use cases for promoting a read replica

You might want to promote a read replica to a standalone DB instance for any of the following reasons:

- **Implementing failure recovery** – You can use read replica promotion as a data recovery scheme if the primary DB instance fails. This approach complements synchronous replication, automatic failure detection, and failover.

If you are aware of the ramifications and limitations of asynchronous replication and you still want to use read replica promotion for data recovery, you can. To do this, first create a read replica and then monitor the primary DB instance for failures. In the event of a failure, do the following:

1. Promote the read replica.
2. Direct database traffic to the promoted DB instance.
3. Create a replacement read replica with the promoted DB instance as its source.

- **Upgrading storage configuration** – If your source DB instance isn't on the preferred storage configuration, you can create a read replica of the instance and upgrade the storage file system configuration. This option migrates the file system of the read replica to the preferred configuration. You can then promote the read replica to a standalone instance.

  You can use this option to overcome the scaling limitations on storage and file size for older 32-bit file systems. For more information, see the section called “Upgrading the storage file system”.

  This option is only available if your source DB instance is *not* on the latest storage configuration, or if you're modifying the DB instance class within the same request.
- **Sharding** – Sharding embodies the "share-nothing" architecture and essentially involves breaking a large database into several smaller databases. One common way to split a database is splitting tables that are not joined in the same query onto different hosts. Another method is duplicating a table across multiple hosts and then using a hashing algorithm to determine which host receives a given update. You can create read replicas corresponding to each of your shards (smaller databases) and promote them when you decide to convert them into standalone shards. You can then carve out the key space (if you are splitting rows) or distribution of tables for each of the shards depending on your requirements.
- **Performing DDL operations (MySQL and MariaDB only)** – DDL operations, such as creating or rebuilding indexes, can take time and impose a significant performance penalty on your DB instance. You can perform these operations on a MySQL or MariaDB read replica once the read replica is in sync with its primary DB instance. Then you can promote the read replica and direct your applications to use the promoted instance.

> **Note**
>
> If your read replica is an RDS for Oracle DB instance, you can perform a *switchover* instead of a promotion. In a switchover, the source DB instance becomes the new replica, and the

replica becomes the new source DB instance. For more information, see Performing an Oracle Data Guard switchover.

Characteristics of a promoted read replica

After you promote the read replica, it ceases to function as a read replica and becomes a standalone DB instance. The new standalone DB instance has the following characteristics:

- The standalone DB instance retains the option group and the parameter group of the pre-promotion read replica.
- You can create read replicas from the standalone DB instance and perform point-in-time restore operations.
- You can't use the DB instance as a replication target because it is no longer a read replica.

Prerequisites for promoting a read replica

Before you promote a read replica, do the following:

- Review your backup strategy:
  - We recommend that you enable backups and complete at least one backup. Backup duration is a function of the number of changes to the database since the previous backup.
  - If you have enabled backups on your read replica, configure the automated backup window so that daily backups don't interfere with read replica promotion.
  - Make sure that your read replica doesn't have the `backing-up` status. You can't promote a read replica when it is in this state.
- Stop any transactions from being written to the primary DB instance, and then wait for RDS to apply all updates to the read replica.

  Database updates occur on the read replica after they have occurred on the primary DB instance. Replication lag can vary significantly. Use the `Replica Lag` metric to determine when all updates have been made to the read replica.
- (MySQL and MariaDB only) To make changes to a MySQL or MariaDB read replica before you promote it, set the `read_only` parameter to `0` in the DB parameter group for the read replica. You can then perform all needed DDL operations, such as creating indexes, on the read replica. Actions taken on the read replica don't affect the performance of the primary DB instance.

Promoting a read replica: basic steps

The following steps show the general process for promoting a read replica to a DB instance:

1. Promote the read replica by using the **Promote** option on the Amazon RDS console, the AWS CLI command `promote-read-replica`, or the `PromoteReadReplica` Amazon RDS API operation.

   > **Note**
   >
   > The promotion process takes a few minutes to complete. When you promote a read replica, RDS stops replication and reboots the read replica. When the reboot is complete, the read replica is available as a new DB instance.

2. (Optional) Modify the new DB instance to be a Multi-AZ deployment. For more information, see Modifying an Amazon RDS DB instance and Configuring and managing a Multi-AZ deployment for Amazon RDS.

Console

To promote a read replica to a standalone DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the Amazon RDS console, choose **Databases**.

   The **Databases** pane appears. Each read replica shows **Replica** in the **Role** column.
3. Choose the read replica that you want to promote.
4. For **Actions**, choose **Promote**.
5. On the **Promote Read Replica** page, enter the backup retention period and the backup window for the newly promoted DB instance.
6. When the settings are as you want them, choose **Continue**.
7. On the acknowledgment page, choose **Promote Read Replica**.

AWS CLI

To promote a read replica to a standalone DB instance, use the AWS CLI `promote-read-replica` command.

Example

For Linux, macOS, or Unix:

```
aws rds promote-read-replica \
    --db-instance-identifier myreadreplica
```

For Windows:

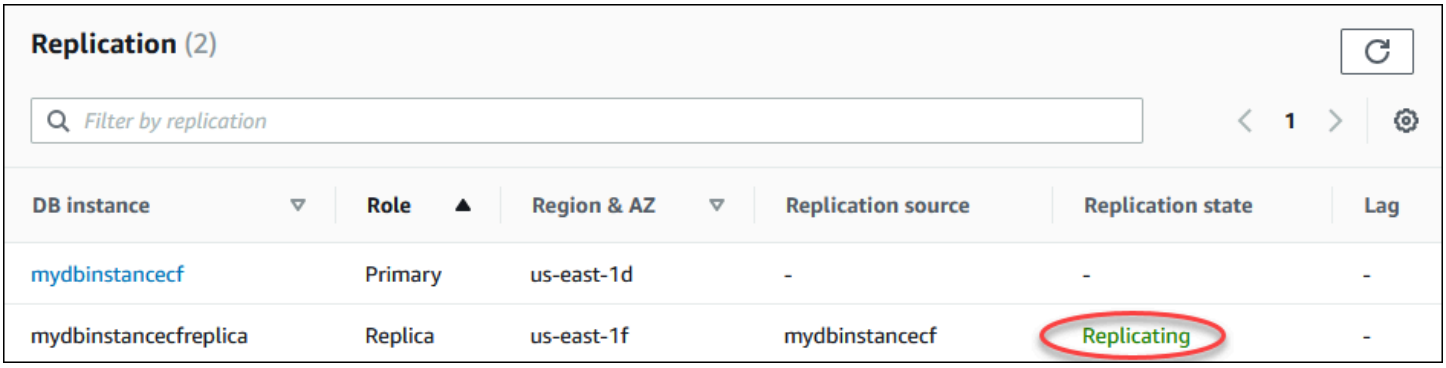
```
aws rds promote-read-replica ^
    --db-instance-identifier myreadreplica
```

RDS API

To promote a read replica to a standalone DB instance, call the Amazon RDS API `PromoteReadReplica` operation with the required parameter `DBInstanceIdentifier`.

Monitoring read replication

You can monitor the status of a read replica in several ways. The Amazon RDS console shows the status of a read replica in the **Replication** section of the **Connectivity & security** tab in the read replica details. To view the details for a read replica, choose the name of the read replica in the list of DB instances in the Amazon RDS console.



You can also see the status of a read replica using the AWS CLI `describe-db-instances` command or the Amazon RDS API `DescribeDBInstances` operation.

The status of a read replica can be one of the following:

- **replicating** – The read replica is replicating successfully.
- **replication degraded (SQL Server and PostgreSQL only) –** Replicas are receiving data from the primary instance, but one or more databases might be not getting updates. This can occur, for example, when a replica is in the process of setting up newly created databases. It can also occur when unsupported DDL or large object changes are made in the blue environment of a blue/green deployment.

  The status doesn't transition from `replication degraded` to `error`, unless an error occurs during the degraded state.
- **error** – An error has occurred with the replication. Check the **Replication Error** field in the Amazon RDS console or the event log to determine the exact error. For more information about troubleshooting a replication error, see Troubleshooting a MySQL read replica problem.
- **terminated (MariaDB, MySQL, or PostgreSQL only)** – Replication is terminated. This occurs if replication is stopped for more than 30 consecutive days, either manually or due to a replication error. In this case, Amazon RDS terminates replication between the primary DB instance and all read replicas. Amazon RDS does this to prevent increased storage requirements on the source DB instance and long failover times.

  Broken replication can affect storage because the logs can grow in size and number due to the high volume of errors messages being written to the log. Broken replication can also affect failure recovery due to the time Amazon RDS requires to maintain and process the large number of logs during recovery.
- **terminated (Oracle only)** – Replication is terminated. This occurs if replication is stopped for more than 8 hours because there isn't enough storage remaining on the read replica. In this case, Amazon RDS terminates replication between the primary DB instance and the affected read replica. This status is a terminal state, and the read replica must be re-created.
- **stopped (MariaDB or MySQL only)** – Replication has stopped because of a customer-initiated request.
- **replication stop point set (MySQL only)** – A customer-initiated stop point was set using the mysql.rds_start_replication_until stored procedure and the replication is in progress.
- **replication stop point reached (MySQL only)** – A customer-initiated stop point was set using the mysql.rds_start_replication_until stored procedure and replication is stopped because the stop point was reached.

You can see where a DB instance is being replicated and if so, check its replication status. On the **Databases** page in the RDS console, it shows **Primary** in the **Role** column. Choose its DB instance name. On its detail page, on the **Connectivity & security** tab, its replication status is under **Replication**.

Monitoring replication lag

You can monitor replication lag in Amazon CloudWatch by viewing the Amazon RDS `ReplicaLag` metric.

For Db2, the `ReplicaLag` metric is the maximum lag of databases that have fallen behind, in seconds. For example, if two databases lag 5 seconds and 10 seconds, respectively, then `ReplicaLag` is 10 seconds. Databases without available High Availability Disaster Recovery (HADR) statuses aren't included in the calculation.

For MariaDB and MySQL, the `ReplicaLag` metric reports the value of the `Seconds_Behind_Master` field of the `SHOW REPLICA STATUS` command. Common causes for replication lag for MySQL and MariaDB are the following:

- A network outage.
- Writing to tables with indexes on a read replica. If the `read_only` parameter is not set to 0 on the read replica, it can break replication.
- Using a nontransactional storage engine such as MyISAM. Replication is only supported for the InnoDB storage engine on MySQL and the XtraDB storage engine on MariaDB.

> **Note**
>
> Previous versions of MariaDB used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MariaDB version lower than 10.5, then use `SHOW SLAVE STATUS`.

When the `ReplicaLag` metric reaches 0, the replica has caught up to the primary DB instance. If the `ReplicaLag` metric returns `-1`, then replication is currently not active. `ReplicaLag = -1` is equivalent to `Seconds_Behind_Master = NULL`.

For Oracle, the `ReplicaLag` metric is the sum of the `Apply Lag` value and the difference between the current time and the apply lag's `DATUM_TIME` value. The `DATUM_TIME` value is the

last time the read replica received data from its source DB instance. For more information, see V$DATAGUARD_STATS in the Oracle documentation.

For SQL Server, the `ReplicaLag` metric is the maximum lag of databases that have fallen behind, in seconds. For example, if you have two databases that lag 5 seconds and 10 seconds, respectively, then `ReplicaLag` is 10 seconds. The `ReplicaLag` metric returns the value of the following query.

```
SELECT MAX(secondary_lag_seconds) max_lag FROM sys.dm_hadr_database_replica_states;
```

For more information, see secondary_lag_seconds in the Microsoft documentation.

`ReplicaLag` returns `-1` if RDS can't determine the lag, such as during replica setup, or when the read replica is in the `error` state.

> **Note**
>
> New databases aren't included in the lag calculation until they are accessible on the read replica.

For PostgreSQL, the `ReplicaLag` metric returns the value of the following query.

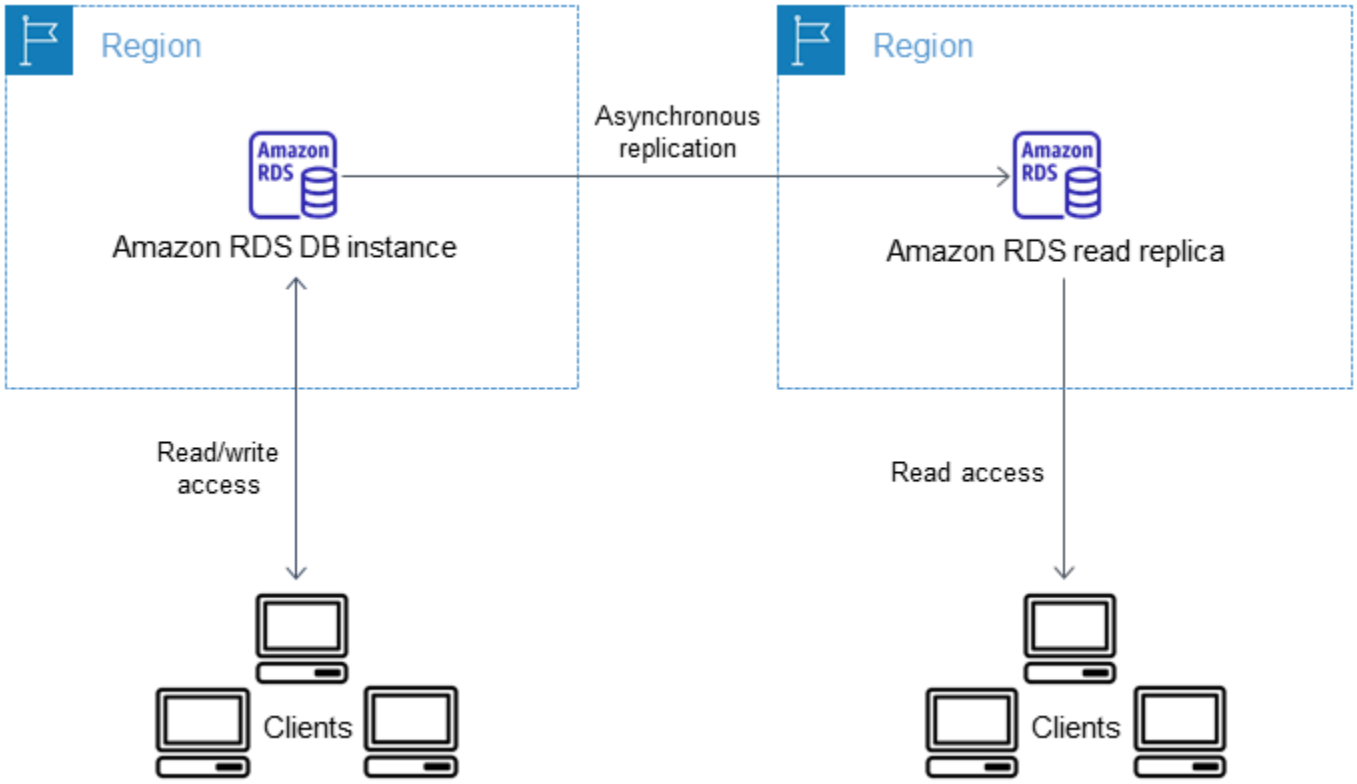
```
SELECT extract(epoch from now() - pg_last_xact_replay_timestamp()) AS reader_lag
```

PostgreSQL versions 9.5.2 and later use physical replication slots to manage write ahead log (WAL) retention on the source instance. For each cross-Region read replica instance, Amazon RDS creates a physical replication slot and associates it with the instance. Two Amazon CloudWatch metrics, `Oldest Replication Slot Lag` and `Transaction Logs Disk Usage`, show how far behind the most lagging replica is in terms of WAL data received and how much storage is being used for WAL data. The `Transaction Logs Disk Usage` value can substantially increase when a cross-Region read replica is lagging significantly.

For more information about monitoring a DB instance with CloudWatch, see Monitoring Amazon RDS metrics with Amazon CloudWatch.

Creating a read replica in a different AWS Region

With Amazon RDS, you can create a read replica in a different AWS Region from the source DB instance.



You create a read replica in a different AWS Region to do the following:

- Improve your disaster recovery capabilities.
- Scale read operations into an AWS Region closer to your users.
- Make it easier to migrate from a data center in one AWS Region to a data center in another AWS Region.

Creating a read replica in a different AWS Region from the source instance is similar to creating a replica in the same AWS Region. You can use the AWS Management Console, run the `create-db-instance-read-replica` command, or call the `CreateDBInstanceReadReplica` API operation.

> **Note**
>
> To create an encrypted read replica in a different AWS Region from the source DB instance, the source DB instance must be encrypted.

Topics

- Region and version availability
- Creating a cross-Region read replica
- How Amazon RDS does cross-Region replication
- Cross-Region replication considerations
- Cross-Region replication costs

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability with cross-Region replication, see Supported Regions and DB engines for cross-Region read replicas in Amazon RDS.

Creating a cross-Region read replica

The following procedures show how to create a read replica from a source Db2, MariaDB, Microsoft SQL Server, MySQL, Oracle, or PostgreSQL DB instance in a different AWS Region.

Console

You can create a read replica across AWS Regions using the AWS Management Console.

To create a read replica across AWS Regions with the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the Db2, MariaDB, Microsoft SQL Server, MySQL, Oracle, or PostgreSQL DB instance that you want to use as the source for a read replica.
4. For **Actions**, choose **Create read replica**.
5. For **DB instance identifier**, enter a name for the read replica.
6. Choose the **Destination Region**.
7. Choose the instance specifications that you want to use. We recommend that you use the same or larger DB instance class and storage type for the read replica.
8. To create an encrypted read replica in another AWS Region:

    a. Choose **Enable encryption**.

b. For **AWS KMS key**, choose the AWS KMS key identifier of the KMS key in the destination AWS Region.

> **Note**
>
> To create an encrypted read replica, the source DB instance must be encrypted. To learn more about encrypting the source DB instance, see Encrypting Amazon RDS resources.

9. Choose other options, such as storage autoscaling.
10. Choose **Create read replica**.

AWS CLI

To create a read replica from a source Db2, MariaDB, Microsoft SQL Server, MySQL, Oracle, or PostgreSQL DB instance in a different AWS Region, you can use the `create-db-instance-read-replica` command. In this case, you use `create-db-instance-read-replica` from the AWS Region where you want the read replica (destination Region) and specify the Amazon Resource Name (ARN) for the source DB instance. An ARN uniquely identifies a resource created in Amazon Web Services.

For example, if your source DB instance is in the US East (N. Virginia) Region, the ARN looks similar to this example:

```
arn:aws:rds:us-east-1:123456789012:db:mydbinstance
```

For information about ARNs, see Amazon Resource Names (ARNs) in Amazon RDS.

To create a read replica in a different AWS Region from the source DB instance, you can use the AWS CLI `create-db-instance-read-replica` command from the destination AWS Region. The following parameters are required for creating a read replica in another AWS Region:

- `--region` – The destination AWS Region where the read replica is created.
- `--source-db-instance-identifier` – The DB instance identifier for the source DB instance. This identifier must be in the ARN format for the source AWS Region.
- `--db-instance-identifier` – The identifier for the read replica in the destination AWS Region.

Example of a cross-Region read replica

The following code creates a read replica in the US West (Oregon) Region from a source DB instance in the US East (N. Virginia) Region.

For Linux, macOS, or Unix:

```
aws rds create-db-instance-read-replica \
    --db-instance-identifier myreadreplica \
    --region us-west-2 \
    --source-db-instance-identifier arn:aws:rds:us-east-1:123456789012:db:mydbinstance
```

For Windows:

```
aws rds create-db-instance-read-replica ^
    --db-instance-identifier myreadreplica ^
    --region us-west-2 ^
    --source-db-instance-identifier arn:aws:rds:us-east-1:123456789012:db:mydbinstance
```

The following parameter is also required for creating an encrypted read replica in another AWS Region:

- `--kms-key-id` – The AWS KMS key identifier of the KMS key to use to encrypt the read replica in the destination AWS Region.

Example of an encrypted cross-Region read replica

The following code creates an encrypted read replica in the US West (Oregon) Region from a source DB instance in the US East (N. Virginia) Region.

For Linux, macOS, or Unix:

```
aws rds create-db-instance-read-replica \
    --db-instance-identifier myreadreplica \
    --region us-west-2 \
    --source-db-instance-identifier arn:aws:rds:us-east-1:123456789012:db:mydbinstance
 \
    --kms-key-id my-us-west-2-key
```

For Windows:

```
aws rds create-db-instance-read-replica ^
    --db-instance-identifier myreadreplica ^
    --region us-west-2 ^
    --source-db-instance-identifier arn:aws:rds:us-east-1:123456789012:db:mydbinstance
 ^
    --kms-key-id my-us-west-2-key
```

The `--source-region` option is required when you're creating an encrypted read replica between the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions. For `--source-region`, specify the AWS Region of the source DB instance.

If `--source-region` isn't specified, specify a `--pre-signed-url` value. A *presigned URL* is a URL that contains a Signature Version 4 signed request for the `create-db-instance-read-replica` command that's called in the source AWS Region. To learn more about the `pre-signed-url` option, see create-db-instance-read-replica in the *AWS CLI Command Reference*.

RDS API

To create a read replica from a source Db2, MariaDB, MySQL, Microsoft SQL Server, Oracle, or PostgreSQL DB instance in a different AWS Region, you can call the Amazon RDS API operation CreateDBInstanceReadReplica. In this case, you call CreateDBInstanceReadReplica from the AWS Region where you want the read replica (destination Region) and specify the Amazon Resource Name (ARN) for the source DB instance. An ARN uniquely identifies a resource created in Amazon Web Services.

To create an encrypted read replica in a different AWS Region from the source DB instance, you can use the Amazon RDS API `CreateDBInstanceReadReplica` operation from the destination AWS Region. To create an encrypted read replica in another AWS Region, you must specify a value for `PreSignedURL`. `PreSignedURL` should contain a request for the `CreateDBInstanceReadReplica` operation to call in the source AWS Region where the read replica is created in. To learn more about `PreSignedUrl`, see `CreateDBInstanceReadReplica`.

For example, if your source DB instance is in the US East (N. Virginia) Region, the ARN looks similar to the following.

```
arn:aws:rds:us-east-1:123456789012:db:mydbinstance
```

For information about ARNs, see Amazon Resource Names (ARNs) in Amazon RDS.

Example

```
https://us-west-2.rds.amazonaws.com/
    ?Action=CreateDBInstanceReadReplica
    &KmsKeyId=my-us-east-1-key
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCreateDBInstanceReadReplica
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526SourceDBInstanceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%123456789012%25253Adb%25253Amydbinstance
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4%2526SourceDBInstanceIdentifier%253Darn%25253Aaws
%25253Ards%25253Aus-west-2%25253A123456789012%25253Ainstance%25253Amydbinstance
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
         %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
         %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &DBInstanceIdentifier=myreadreplica
    &SourceDBInstanceIdentifier=&region-arn;rds:us-east-1:123456789012:db:mydbinstance
    &Version=2012-01-15
    &SignatureVersion=2
    &SignatureMethod=HmacSHA256
    &Timestamp=2012-01-20T22%3A06%3A23.624Z
    &AWSAccessKeyId=<&AWS; Access Key ID>
    &Signature=<Signature>
```

How Amazon RDS does cross-Region replication

Amazon RDS uses the following process to create a cross-Region read replica. Depending on the AWS Regions involved and the amount of data in the databases, this process can take hours to complete. You can use this information to determine how far the process has proceeded when you create a cross-Region read replica:

1. Amazon RDS begins configuring the source DB instance as a replication source and sets the status to *modifying*.

2. Amazon RDS begins setting up the specified read replica in the destination AWS Region and sets the status to *creating*.
3. Amazon RDS creates an automated DB snapshot of the source DB instance in the source AWS Region. The format of the DB snapshot name is `rds:<InstanceID>-<timestamp>`, where `<InstanceID>` is the identifier of the source instance, and `<timestamp>` is the date and time the copy started. For example, `rds:mysourceinstance-2013-11-14-09-24` was created from the instance `mysourceinstance` at `2013-11-14-09-24`. During the creation of an automated DB snapshot, the source DB instance status remains *modifying*, the read replica status remains *creating*, and the DB snapshot status is *creating*. The progress column of the DB snapshot page in the console reports how far the DB snapshot creation has progressed. When the DB snapshot is complete, the status of both the DB snapshot and source DB instance are set to *available*.
4. Amazon RDS begins a cross-Region snapshot copy for the initial data transfer. The snapshot copy is listed as an automated snapshot in the destination AWS Region with a status of *creating*. It has the same name as the source DB snapshot. The progress column of the DB snapshot display indicates how far the copy has progressed. When the copy is complete, the status of the DB snapshot copy is set to *available*.
5. Amazon RDS then uses the copied DB snapshot for the initial data load on the read replica. During this phase, the read replica is in the list of DB instances in the destination, with a status of *creating*. When the load is complete, the read replica status is set to *available*, and the DB snapshot copy is deleted.
6. When the read replica reaches the available status, Amazon RDS starts by replicating the changes made to the source instance since the start of the create read replica operation. During this phase, the replication lag time for the read replica is greater than 0.

   For information about replication lag time, see Monitoring read replication.

Cross-Region replication considerations

All of the considerations for performing replication within an AWS Region apply to cross-Region replication. The following extra considerations apply when replicating between AWS Regions:

- A source DB instance can have cross-Region read replicas in multiple AWS Regions. Because of the limit on the number of access control list (ACL) entries for the source VPC, RDS can't guarantee more than five cross-Region read replica DB instances.

- You can have up to 20 concurrent create cross-Region read replica requests in progress to a single destination AWS Region per account. You must wait for these requests to complete before submitting additional create cross-Region read replica requests to the same destination AWS Region.
- You can replicate between the GovCloud (US-East) and GovCloud (US-West) Regions, but not into or out of GovCloud (US).
- For the following RDS engines, you can create a cross-Region Amazon RDS read replica from a source Amazon RDS DB instance only when it isn't a read replica of another Amazon RDS DB instance:
  - RDS for Db2
  - RDS for SQL Server
  - RDS for Oracle
  - RDS for PostgreSQL versions lower than 14.1

  This limitation doesn't apply to DB instances running RDS for PostgreSQL version 14.1 and higher, RDS for MariaDB, and RDS for MySQL.
- You can expect to see a higher level of lag time for any read replica that is in a different AWS Region than the source instance. This lag time comes from the longer network channels between regional data centers.
- For cross-Region read replicas, any of the create read replica commands that specify the `--db-subnet-group-name` parameter must specify a DB subnet group from the same VPC.
- In most cases, the read replica uses the default DB parameter group and DB option group for the specified DB engine.

  For the Db2 DB engine, you must specify a custom parameter group for the replica. For replicas that use the BYOL model, this custom parameter group must include your IBM Site ID and IBM Customer ID. You can specify this custom parameter group for the replica by using the AWS Management Console, the AWS CLI, or the RDS API. For more information, see IBM IDs for bring your own license (BYOL) for Db2.

  For the MySQL and Oracle DB engines, you can specify a custom parameter group for the read replica in the `--db-parameter-group-name` option of the AWS CLI command create-db-instance-read-replica. You can't specify a custom parameter group when you use the AWS Management Console.
- The read replica uses the default security group.

- For Db2, MariaDB, Microsoft SQL Server, MySQL, and Oracle DB instances, when the source DB instance for a cross-Region read replica is deleted, the read replica is promoted.
- For PostgreSQL DB instances, when the source DB instance for a cross-Region read replica is deleted, the replication status of the read replica is set to `terminated`. The read replica isn't promoted.

  You have to promote the read replica manually or delete it.

Requesting a cross-Region read replica

To communicate with the source Region to request the creation of a cross-Region read replica, the requester (IAM role or IAM user) must have access to the source DB instance and the source Region.

Certain conditions in the requester's IAM policy can cause the request to fail. The following examples assume that the source DB instance is in US East (Ohio) and the read replica is created in US East (N. Virginia). These examples show conditions in the requester's IAM policy that cause the request to fail:

- The requester's policy has a condition for `aws:RequestedRegion`.

```
...
"Effect": "Allow",
"Action": "rds:CreateDBInstanceReadReplica",
"Resource": "*",
"Condition": {
    "StringEquals": {
        "aws:RequestedRegion": "us-east-1"
    }
}
```

  The request fails because the policy doesn't allow access to the source Region. For a successful request, specify both the source and destination Regions.

```
...
"Effect": "Allow",
"Action": "rds:CreateDBInstanceReadReplica",
"Resource": "*",
"Condition": {
    "StringEquals": {
        "aws:RequestedRegion": [
```

```
            "us-east-1",
            "us-east-2"
        ]
    }
}
```

- The requester's policy doesn't allow access to the source DB instance.

```
...
"Effect": "Allow",
"Action": "rds:CreateDBInstanceReadReplica",
"Resource": "arn:aws:rds:us-east-1:123456789012:db:myreadreplica"
...
```

For a successful request, specify both the source instance and the replica.

```
...
"Effect": "Allow",
"Action": "rds:CreateDBInstanceReadReplica",
"Resource": [
    "arn:aws:rds:us-east-1:123456789012:db:myreadreplica",
    "arn:aws:rds:us-east-2:123456789012:db:mydbinstance"
]
...
```

- The requester's policy denies `aws:ViaAWSService`.

```
...
"Effect": "Allow",
"Action": "rds:CreateDBInstanceReadReplica",
"Resource": "*",
"Condition": {
    "Bool": {"aws:ViaAWSService": "false"}
}
```

Communication with the source Region is made by RDS on the requester's behalf. For a successful request, don't deny calls made by AWS services.

- The requester's policy has a condition for `aws:SourceVpc` or `aws:SourceVpce`.

These requests might fail because when RDS makes the call to the remote Region, it isn't from the specified VPC or VPC endpoint.

If you need to use one of the previous conditions that would cause a request to fail, you can include a second statement with `aws:CalledVia` in your policy to make the request succeed. For example, you can use `aws:CalledVia` with `aws:SourceVpce` as shown here:

```
...
"Effect": "Allow",
"Action": "rds:CreateDBInstanceReadReplica",
"Resource": "*",
"Condition": {
    "Condition" : {
        "ForAnyValue:StringEquals" : {
          "aws:SourceVpce": "vpce-1a2b3c4d"
        }
     }
},
{
    "Effect": "Allow",
    "Action": [
        "rds:CreateDBInstanceReadReplica"
    ],
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "aws:CalledVia": [
                "rds.amazonaws.com"
            ]
        }
    }
}
```

For more information, see Policies and permissions in IAM in the *IAM User Guide*.

Authorizing the read replica

After a cross-Region DB read replica creation request returns `success`, RDS starts the replica creation in the background. An authorization for RDS to access the source DB instance is created. This authorization links the source DB instance to the read replica, and allows RDS to copy only to the specified read replica.

The authorization is verified by RDS using the `rds:CrossRegionCommunication` permission in the service-linked IAM role. If the replica is authorized, RDS communicates with the source Region and completes the replica creation.

RDS doesn't have access to DB instances that weren't authorized previously by a `CreateDBInstanceReadReplica` request. The authorization is revoked when read replica creation completes.

RDS uses the service-linked role to verify the authorization in the source Region. If you delete the service-linked role during the replication creation process, the creation fails.

For more information, see Using service-linked roles in the *IAM User Guide*.

Using AWS Security Token Service credentials

Session tokens from the global AWS Security Token Service (AWS STS) endpoint are valid only in AWS Regions that are enabled by default (commercial Regions). If you use credentials from the `assumeRole` API operation in AWS STS, use the regional endpoint if the source Region is an opt-in Region. Otherwise, the request fails. This happens because your credentials must be valid in both Regions, which is true for opt-in Regions only when the regional AWS STS endpoint is used.

To use the global endpoint, make sure that it's enabled for both Regions in the operations. Set the global endpoint to `Valid in all AWS Regions` in the AWS STS account settings.

The same rule applies to credentials in the presigned URL parameter.

For more information, see Managing AWS STS in an AWS Region in the *IAM User Guide*.

Cross-Region replication costs

The data transferred for cross-Region replication incurs Amazon RDS data transfer charges. These cross-Region replication actions generate charges for the data transferred out of the source AWS Region:

- When you create a read replica, Amazon RDS takes a snapshot of the source instance and transfers the snapshot to the read replica AWS Region.
- For each data modification made in the source databases, Amazon RDS transfers data from the source AWS Region to the read replica AWS Region.

For more information about data transfer pricing, see Amazon RDS pricing.

For MySQL and MariaDB instances, you can reduce your data transfer costs by reducing the number of cross-Region read replicas that you create. For example, suppose that you have a source DB instance in one AWS Region and want to have three read replicas in another AWS Region. In this

case, you create only one of the read replicas from the source DB instance. You create the other two replicas from the first read replica instead of the source DB instance.

For example, if you have `source-instance-1` in one AWS Region, you can do the following:

- Create `read-replica-1` in the new AWS Region, specifying `source-instance-1` as the source.
- Create `read-replica-2` from `read-replica-1`.
- Create `read-replica-3` from `read-replica-1`.

In this example, you are only charged for the data transferred from `source-instance-1` to `read-replica-1`. You aren't charged for the data transferred from `read-replica-1` to the other two replicas because they are all in the same AWS Region. If you create all three replicas directly from `source-instance-1` in another AWS Region, you are charged for the data transfers to all three replicas.

Tagging Amazon RDS resources

An Amazon RDS *tag* is a name-value pair that you define and associate with an Amazon RDS resource such as a DB instance or DB snapshot. The name is referred to as the *key*. Optionally, you can supply a value for the key.

You can use the AWS Management Console, the AWS CLI, or the Amazon RDS API to add, list, and delete tags on Amazon RDS resources. When using the CLI or API, make sure to provide the Amazon Resource Name (ARN) for the RDS resource to work with. For more information about constructing an ARN, see Constructing an ARN for Amazon RDS.

You can use tags to add metadata to your Amazon RDS resources. You can use the tags to add your own notations about database instances, snapshots, Aurora clusters, and so on. Doing so can help you to document your Amazon RDS resources. You can also use the tags with automated maintenance procedures.

In particular, you can use these tags with IAM policies. You can use them to manage access to Amazon RDS resources and to control what actions can be applied to those resources. You can also use these tags to track costs by grouping expenses for similarly tagged resources.

You can tag the following Amazon RDS resources:

- DB instances
- DB clusters
- Aurora global clusters
- DB cluster endpoints
- Read replicas
- DB snapshots
- DB cluster snapshots
- Reserved DB instances
- Event subscriptions
- DB option groups
- DB parameter groups
- DB cluster parameter groups
- DB subnet groups

- RDS Proxies
- RDS Proxy endpoints
- Blue/green deployments
- Zero-ETL integrations
- Automated backups
- Cluster automated backups

> **Note**
>
> When you tag a DB instance, Amazon RDS automatically applies those tags to the associated Performance Insights resources. Currently, you can't tag RDS Proxies and RDS Proxy endpoints by using the AWS Management Console.

Topics

- Why use Amazon RDS resource tags?
- How Amazon RDS resource tags work
- Best practices for tagging Amazon RDS resources
- Copying tags to DB snapshots
- Tagging automated backup resources
- Adding and deleting tags in Amazon RDS
- Tutorial: Specify which DB instances to stop by using tags

Why use Amazon RDS resource tags?

You can use tags to do the following:

- Categorize your RDS resources by application, project, department, environment, and so on. For example, you could use a tag key to define a category, where the tag value is an item in this category. You might create the tag `environment=prod`. Or you might define a tag key of `project` and a tag value of `Salix`, which indicates that an Amazon RDS resource is assigned to the Salix project.
- Automate resource management tasks. For example, you could create a maintenance window for instances tagged `environment=prod` that differs from the window for instances tagged

`environment=test`. You could also configure automatic DB snapshots for instances tagged `environment=prod`.

- Control access to RDS resources within an IAM policy. You can do this by using the global `aws:ResourceTag/tag-key` condition key. For example, a policy might allow only users in the `DBAdmin` group to modify DB instances tagged with `environment=prod`. For information about managing access to tagged resources with IAM policies, see Identity and access management for Amazon RDS and Controlling access to AWS resources in the *AWS Identity and Access Management User Guide*.
- Monitor resources based on a tag. For example, you can create an Amazon CloudWatch dashboard for DB instances tagged with `environment=prod`.
- Track costs by grouping expenses for similarly tagged resources. For example, if you tag RDS resources associated with the Salix project with `project=Salix`, you can generate cost reports for and allocate expenses to this project. For more information, see How AWS billing works with tags in Amazon RDS.

How Amazon RDS resource tags work

AWS doesn't apply any semantic meaning to your tags. Tags are interpreted strictly as character strings.

Topics

- Tag sets in Amazon RDS
- Tag structure in Amazon RDS
- Amazon RDS resources eligible for tagging
- How AWS billing works with tags in Amazon RDS

Tag sets in Amazon RDS

Every Amazon RDS resource has a container called a *tag set*. The container includes all the tags that are assigned to the resource. A resource has exactly one tag set.

A tag set contains 0—50 tags. If you add a tag to an RDS resource with the same key as an existing resource tag, the new value overwrites the old.

Tag structure in Amazon RDS

The structure of an RDS tag is as follows:

Tag key

The key is the required name of the tag. The string value must be 1—128 Unicode characters in length and cannot be prefixed with `aws:` or `rds:`. The string can contain only the set of Unicode letters, digits, whitespace, `_`, `.`, `:`, `/`, `=`, `+`, `-`, and `@`. The Java regex is `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"`. Tag keys are case-sensitive. Thus, the keys `project` and `Project` are distinct.

A key is unique to a tag set. For example, you cannot have a key-pair in a tag set with the key the same but with different values, such as `project=Trinity` and `project=Xanadu`.

Tag value

The value is an optional string value of the tag. The string value must be 1—256 Unicode characters in length. The string can contain only the set of Unicode letters, digits, whitespace, `_`, `.`, `:`, `/`, `=`, `+`, `-`, and `@`. The Java regex is `"^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$"`. Tag values are case-sensitive. Thus, the values `prod` and `Prod` are distinct.

Values don't need to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of `project=Trinity` and `cost-center=Trinity`.

Amazon RDS resources eligible for tagging

You can tag the following Amazon RDS resources:

- DB instances
- DB clusters
- DB cluster endpoints
- Read replicas
- DB snapshots
- DB cluster snapshots
- Reserved DB instances
- Event subscriptions
- DB option groups

- DB parameter groups
- DB cluster parameter groups
- DB subnet groups
- RDS Proxies
- RDS Proxy endpoints

> **Note**
>
> Currently, you can't tag RDS Proxies and RDS Proxy endpoints by using the AWS Management Console.

- Blue/green deployments
- Zero-ETL integrations (preview)
- Automated backups
- Cluster automated backups

How AWS billing works with tags in Amazon RDS

Use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see Using Cost Allocation Tags in the *AWS Billing User Guide*.

How cost allocation tags work with DB snapshots

You can add a tag to a DB snapshot. However, your bill won't reflect this grouping. For cost allocation tags to apply to DB snapshots, the following conditions must be met:

- The tags must be attached to the parent DB instance.
- The parent DB instance must exist in the same AWS account as the DB snapshot.
- The parent DB instance must exist in the same AWS Region as the DB snapshot.

DB snapshots are considered orphaned if they don't exist in the same Region as the parent DB instance, or if the parent DB instance is deleted. Orphaned DB snapshots don't support cost

allocation tags. Costs for orphaned snapshots are aggregated in a single untagged line item. Cross-account DB snapshots aren't considered orphaned when the following conditions are met:

- They exist in the same Region as the parent DB instance.
- The parent DB instance is owned by the source account.

> **Note**
>
> If the parent DB instance is owned by a different account, cost allocation tags don't apply to cross-account snapshots in the destination account.

Best practices for tagging Amazon RDS resources

When you use tags, we recommend that you adhere to the following best practices:

- Document conventions for tag use that are followed by all teams in your organization. In particular, ensure the names are both descriptive and consistent. For example, standardize on the format `environment:prod` rather than tagging some resources with `env:production`.

  > **Important**
  >
  > Do not store personally identifiable information (PII) or other confidential or sensitive information in tags.

- Automate tagging to ensure consistency. For example, you can use the following techniques:
  - Include tags in an CloudFormation template. When you create resources with the template, the resources are tagged automatically.
  - Define and apply tags using AWS Lambda functions.
  - Create an SSM document that includes steps to add tags to your RDS resources.
- Use tags only when necessary. You can add up to 50 tags for a single RDS resource, but a best practice is to avoid unnecessary tag proliferation and complexity.
- Review tags periodically for relevance and accuracy. Remove or modify outdated tags as needed.
- Consider creating tags with the AWS Tag Editor in the AWS Management Console. You can use the Tag Editor to add tags to multiple supported AWS resources, including RDS resources, at the same time. For more information, see Tag Editor in the *AWS Resource Groups User Guide*.

Copying tags to DB snapshots

When you create or restore a DB instance, you can specify that the tags from the DB instance are copied to snapshots of the DB instance. Copying tags ensures that the metadata for the DB snapshots matches that of the source DB instance. It also ensures that any access policies for the DB snapshots also match those of the source DB instance.

You can specify that tags are copied to DB snapshots for the following actions:

- Creating a DB instance
- Restoring a DB instance
- Creating a read replica
- Copying a DB snapshot

To copy tags for the preceding actions, choose **Copy tags to snapshots** in the AWS Management Console, or specify `--copy-tags-to-snapshot` in the AWS CLI. While copying a DB snapshot, you can only copy the associated tags if copying the DB snapshot in the same account. You can't copy tags when copying a DB snapshot across accounts.

In most cases, tags aren't copied by default. However, when you restore a DB instance from a DB snapshot, RDS checks whether you specify new tags. If yes, the new tags are added to the restored DB instance. If there are no new tags, and the DB snapshot is in the same account as the source DB instance, RDS adds the tags from the source DB instance at the time of snapshot creation to the restored DB instance. RDS does not add tags from the source DB instance if the DB snapshot is copied to and restored in a different account.

To prevent tags from source DB instances from being added to restored DB instances, we recommend that you specify new tags when restoring a DB instance.

Note

In some cases, you might include a value for the `--tags` parameter of the create-db-snapshot AWS CLI command. Or you might supply at least one tag to the CreateDBSnapshot API operation. In these cases, RDS doesn't copy tags from the source DB instance to the new DB snapshot. This functionality applies even if the source DB instance has the `--copy-tags-to-snapshot` (`CopyTagsToSnapshot`) option turned on.
If you take this approach, you can create a copy of a DB instance from a DB snapshot. This approach avoids adding tags that don't apply to the new DB instance. You create

your DB snapshot using the AWS CLI `create-db-snapshot` command (or the `CreateDBSnapshot` RDS API operation). After you create your DB snapshot, you can add tags as described later in this topic.

Tagging automated backup resources

Automated backup resources are created when you set backup retention period value from 0 to greater than 0. You can tag instance or cluster automated backup resources during creation using the `--tag-specifications` parameter.

Tag-Specifications Parameter

APIs that support the `--tag-specifications` request parameter (like create-db-instance, restore-db-instance-from-db-snapshot, create-db-cluster, etc.) can tag automated backups (Resource Type: `auto-backup` or `cluster-auto-backup`) during creation.

Tagging instance automated backups

Use `--tag-specifications` with `ResourceType=auto-backup` when creating or modifying DB instances that have automated backups enabled.

Tagging cluster automated backups

Use `--tag-specifications` with `ResourceType=cluster-auto-backup` when creating DB clusters that have automated backups enabled.

Tagging automated backups while modifying DB instance

You can only tag automated backups using `--tag-specifications` when setting backup retention from 0 to greater than 0. If automated backups already exists, use AddTagsToResource API instead.

Note

- Automated backup tags are independent of source DB instance tags, DB cluster tags, or DB snapshot tags.

Adding and deleting tags in Amazon RDS

You can do the following:

- Create tags when you create a resource, for example, when you run the AWS CLI command `create-db-instance`.
- Add tags to an existing resource using the command `add-tags-to-resource`.
- List tags associated with a specific resource using the command `list-tags-for-resource`.
- Update tags using the command `add-tags-to-resource`.
- Remove tags from a resource using the command `remove-tags-from-resource`.

The following procedures show how you can perform typical tagging operations on resources related to DB instances. Note that tags are cached for authorization purposes. For this reason, when you add or update tags on Amazon RDS resources, several minutes can pass before the modifications are available.

Console

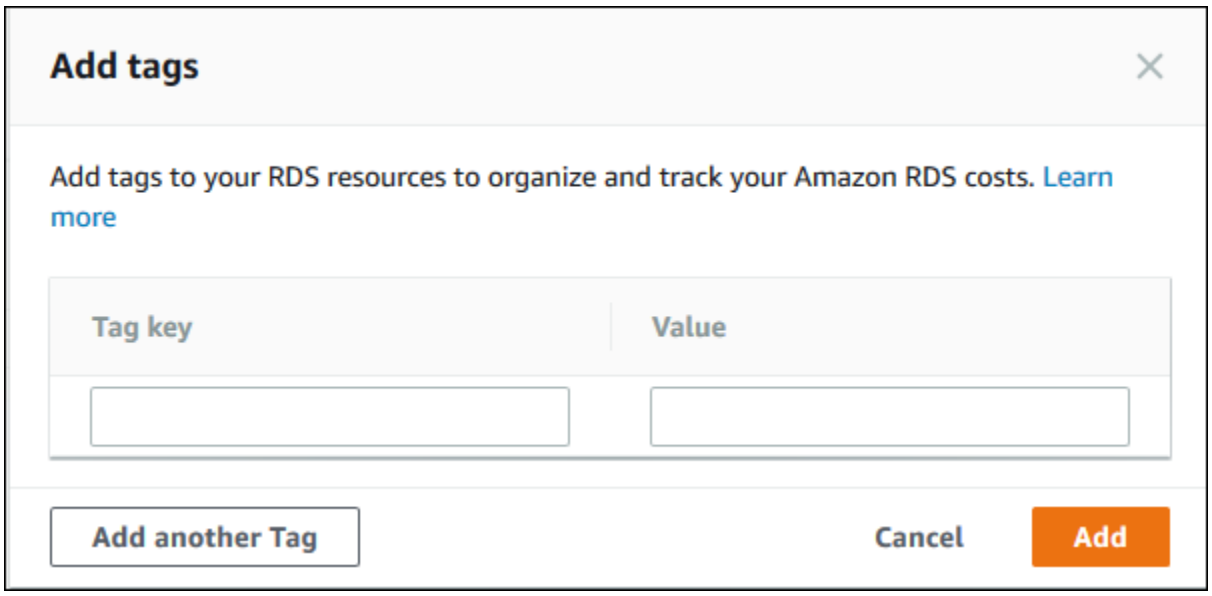
The process to tag an Amazon RDS resource is similar for all resources. The following procedure shows how to tag an Amazon RDS DB instance.

To add a tag to a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.

   > **Note**
   >
   > To filter the list of DB instances in the **Databases** pane, enter a text string for **Filter databases**. Only DB instances that contain the string appear.

3. Choose the name of the DB instance that you want to tag to show its details.
4. In the details section, scroll down to the **Tags** section.
5. Choose **Add**. The **Add tags** window appears.



6. Enter a value for **Tag key** and **Value**.
7. To add another tag, you can choose **Add another Tag** and enter a value for its **Tag key** and **Value**.

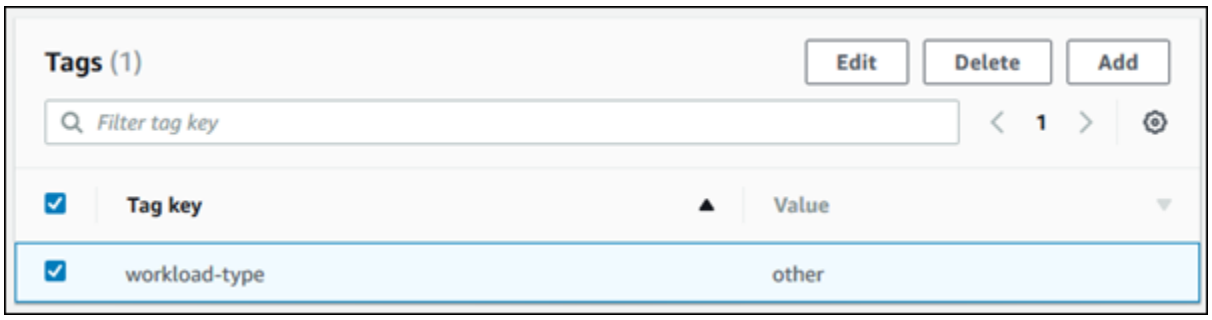
   Repeat this step as many times as necessary.
8. Choose **Add**.

To delete a tag from a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.

   > **Note**
   >
   > To filter the list of DB instances in the **Databases** pane, enter a text string in the **Filter databases** box. Only DB instances that contain the string appear.

3. Choose the name of the DB instance to show its details.
4. In the details section, scroll down to the **Tags** section.
5. Choose the tag you want to delete.



6. Choose **Delete**, and then choose **Delete** in the **Delete tags** window.

AWS CLI

You can add, list, or remove tags for a DB instance using the AWS CLI.

- To add one or more tags to an Amazon RDS resource, use the AWS CLI command `add-tags-to-resource`.
- To list the tags on an Amazon RDS resource, use the AWS CLI command `list-tags-for-resource`.
- To remove one or more tags from an Amazon RDS resource, use the AWS CLI command `remove-tags-from-resource`.

To learn more about how to construct the required ARN, see Constructing an ARN for Amazon RDS.

RDS API

You can add, list, or remove tags for a DB instance using the Amazon RDS API.

- To add a tag to an Amazon RDS resource, use the `AddTagsToResource` operation.
- To list tags that are assigned to an Amazon RDS resource, use the `ListTagsForResource`.
- To remove tags from an Amazon RDS resource, use the `RemoveTagsFromResource` operation.

To learn more about how to construct the required ARN, see Constructing an ARN for Amazon RDS.

When working with XML using the Amazon RDS API, tags use the following schema:

```
<Tagging>
      <TagSet>
          <Tag>
```

```
            <Key>Project</Key>
            <Value>Trinity</Value>
        </Tag>
        <Tag>
            <Key>User</Key>
            <Value>Jones</Value>
        </Tag>
    </TagSet>
</Tagging>
```

The following table provides a list of the allowed XML tags and their characteristics. Values for `Key` and `Value` are case-sensitive. For example, `project=Trinity` and `PROJECT=Trinity` are distinct tags.

Tagging element	Description
TagSet	A tag set is a container for all tags assigned to an Amazon RDS resource. There can be only one tag set per resource. You work with a TagSet only through the Amazon RDS API.
Tag	A tag is a user-defined key-value pair. There can be from 1 to 50 tags in a tag set.
Key	A key is the required name of the tag. For restrictions, see Tag structure in Amazon RDS. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with `aws:` or `rds:`. The string can only contain only the set of Unicode letters, digits, white space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"). Keys must be unique to a tag set. For example, you cannot have a key-pair in a tag set with the key the same but with different values, such as project/Trinity and project/Xanadu.
Value	A value is the optional value of the tag. For restrictions, see Tag structure in Amazon RDS. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with `aws:` or `rds:`. The string can only contain only the set of

Tagging element	Description
	Unicode letters, digits, white space, '_', '.', '/', '=', '+', '-' (Java regex: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"). Values do not have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of project/Trinity and cost-center/Trinity.

Tutorial: Specify which DB instances to stop by using tags

This tutorial assumes that you have several DB instances in a development or test environment. You need to keep these DB instances for several days. Some DB instances run tests overnight, whereas others can be stopped overnight and started again the next day.

The following tutorial shows how to assign a tag to DB instances that are suitable to stop overnight. The tutorial shows how a script can detect which DB instances have the tag and then stop the tagged DB instances. In this example, the value portion of the key-value pair doesn't matter. The presence of the `stoppable` tag signifies that the DB instance has this user-defined property.

In the following tutorial, the commands and APIs for tagging work with ARNs, which allow RDS to work seamlessly across AWS Regions, AWS accounts, and different types of resources that might have identical short names. You can specify the ARN instead of the DB instance ID in CLI commands that operate on DB instances.

To specify which DB instances to stop

1. Determine the ARN of a DB instance that you want to designate as stoppable.

   In the following example, substitute the name of your own DB instances for *`dev-test-db-instance`*. In subsequent commands that use ARN parameters, substitute the ARN of your own DB instance. The ARN includes your own AWS account ID and the name of the AWS Region where your DB instance is located.

   ```
   $ aws rds describe-db-instances --db-instance-identifier dev-test-db-instance \
     --query "*[].{DBInstance:DBInstanceArn}" --output text
   arn:aws:rds:us-east-1:123456789102:db:dev-test-db-instance
   ```

2. Add the tag `stoppable` to this DB instance.

   You choose the name for this tag. Because this example treats the tag as an attribute that is either present or absent, it omits the `Value=` part of the `--tags` parameter. This approach means that you can avoid devising a naming convention that encodes all relevant information in names. In such a convention, you might encode information in the DB instance name or names of other resources.

   ```
   $ aws rds add-tags-to-resource \
     --resource-name arn:aws:rds:us-east-1:123456789102:db:dev-test-db-instance \
     --tags Key=stoppable
   ```

3. Confirm that the tag is present in the DB instance.

   The following commands retrieve the tag information for the DB instance in JSON format and in plain tab-separated text.

   ```
   $ aws rds list-tags-for-resource \
     --resource-name arn:aws:rds:us-east-1:123456789102:db:dev-test-db-instance
   {
       "TagList": [
           {
               "Key": "stoppable",
               "Value": ""

           }
       ]
   }
   aws rds list-tags-for-resource \
     --resource-name arn:aws:rds:us-east-1:123456789102:db:dev-test-db-instance --
   output text
   TAGLIST stoppable
   ```

4. Stop all the DB instances that are designated as `stoppable`.

   The following example create a text file that lists all your DB instances. The shell command loops through the list and checks if each DB instance is tagged with the relevant attribute and performs runs the command `aws rds stop-db-instance` for each DB instance.

   ```
   $ aws rds describe-db-instances --query "*[].[DBInstanceArn]" --output text >/tmp/
   db_instance_arns.lst
   $ for arn in $(cat /tmp/db_instance_arns.lst)
   ```

```
do
  match="$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
 stoppable)"
  if [[ ! -z "$match" ]]
  then
      echo "DB instance $arn is tagged as stoppable. Stopping it now."
# Note that you need to get the DB instance identifier from the ARN.
      dbid=$(echo $arn | sed -e 's/.*://')
      aws rds stop-db-instance --db-instance-identifier $dbid
  fi
done

DB instance arn:arn:aws:rds:us-east-1:123456789102:db:dev-test-db-instance is
 tagged as stoppable. Stopping it now.
{
    "DBInstance": {
        "DBInstanceIdentifier": "dev-test-db-instance",
        "DBInstanceClass": "db.t3.medium",
        ...
```

You can run a script like the preceding one at the end of every day to make sure that nonessential DB instances are stopped. You might also schedule a job using a utility such as `cron` to perform such a check each night. For example, you might do this in case some DB instances were left running by mistake. Here, you might fine-tune the command that prepares the list of DB instances to check.

The following command produces a list of your DB instances, but only the ones in `available` state. The script can ignore DB instances that are already stopped, because they will have different status values such as `stopped` or `stopping`.

```
$ aws rds describe-db-instances \
  --query '*[].{DBInstanceArn:DBInstanceArn,DBInstanceStatus:DBInstanceStatus}|[?
DBInstanceStatus == `available`]|[].{DBInstanceArn:DBInstanceArn}' \
  --output text
arn:aws:rds:us-east-1:123456789102:db:db-instance-2447
arn:aws:rds:us-east-1:123456789102:db:db-instance-3395
arn:aws:rds:us-east-1:123456789102:db:dev-test-db-instance
arn:aws:rds:us-east-1:123456789102:db:pg2-db-instance
```

Tip

You can use assigning tags and finding DB instances with those tags to reduce costs in other ways. For example, take this scenario with DB instances used for development and testing. In this case, you might designate some DB instances to be deleted at the end of each day. Or you might designate them to have their DB instances changed to small DB instance classes during times of expected low usage.

Amazon Resource Names (ARNs) in Amazon RDS

Resources created in Amazon Web Services are each uniquely identified with an Amazon Resource Name (ARN). For certain Amazon RDS operations, you must uniquely identify an Amazon RDS resource by specifying its ARN. For example, when you create an RDS DB instance read replica, you must supply the ARN for the source DB instance.

For information about constructing an ARN and getting an existing ARN, see the following topics.

Topics

- Constructing an ARN for Amazon RDS
- Getting an existing ARN for Amazon RDS

Constructing an ARN for Amazon RDS

Resources created in Amazon Web Services are each uniquely identified with an Amazon Resource Name (ARN). You can construct an ARN for an Amazon RDS resource using the following syntax.

`arn:aws:rds:<region>:<account number>:<resourcetype>:<name>`

Region Name	Region	Endpoint	Protocol	
US East (Ohio)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
		rds-fips.us-east-2.api.aws	HTTPS	
		rds.us-east-2.api.aws	HTTPS	
		rds-fips.us-east-2.amazonaws.com	HTTPS	
US East (N. Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
		rds-fips.us-east-1.api.aws	HTTPS	
		rds-fips.us-east-1.amazonaws.com	HTTPS	
		rds.us-east-1.api.aws	HTTPS	

Region Name	Region	Endpoint	Protocol
US West (N. California)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
		rds.us-west-1.api.aws	HTTPS
		rds-fips.us-west-1.amazonaws.com	HTTPS
		rds-fips.us-west-1.api.aws	HTTPS
US West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
		rds-fips.us-west-2.amazonaws.com	HTTPS
		rds.us-west-2.api.aws	HTTPS
		rds-fips.us-west-2.api.aws	HTTPS
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
		rds.af-south-1.api.aws	HTTPS
Asia Pacific (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
		rds.ap-east-1.api.aws	HTTPS
Asia Pacific (Hyderabad)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
		rds.ap-south-2.api.aws	HTTPS
Asia Pacific (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
		rds.ap-southeast-3.api.aws	HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Malaysia)	ap-southeast-5	rds.ap-southeast-5.amazonaws.com	HTTPS
Asia Pacific (Melbourne)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com rds.ap-southeast-4.api.aws	HTTPS HTTPS
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com rds.ap-south-1.api.aws	HTTPS HTTPS
Asia Pacific (New Zealand)	ap-southeast-6	rds.ap-southeast-6.amazonaws.com	HTTPS
Asia Pacific (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com rds.ap-northeast-3.api.aws	HTTPS HTTPS
Asia Pacific (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com rds.ap-northeast-2.api.aws	HTTPS HTTPS
Asia Pacific (Singapore)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com rds.ap-southeast-1.api.aws	HTTPS HTTPS
Asia Pacific (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com rds.ap-southeast-2.api.aws	HTTPS HTTPS

Region Name	Region	Endpoint	Protocol
Asia Pacific (Taipei)	ap-east-2	rds.ap-east-2.amazonaws.com	HTTPS
Asia Pacific (Thailand)	ap-southeast-7	rds.ap-southeast-7.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com rds.ap-northeast-1.api.aws	HTTPS HTTPS
Canada (Central)	ca-central-1	rds.ca-central-1.amazonaws.com rds.ca-central-1.api.aws rds-fips.ca-central-1.api.aws rds-fips.ca-central-1.amazonaws.com	HTTPS HTTPS HTTPS HTTPS
Canada West (Calgary)	ca-west-1	rds.ca-west-1.amazonaws.com rds-fips.ca-west-1.amazonaws.com	HTTPS HTTPS
Europe (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com rds.eu-central-1.api.aws	HTTPS HTTPS
Europe (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com rds.eu-west-1.api.aws	HTTPS HTTPS
Europe (London)	eu-west-2	rds.eu-west-2.amazonaws.com rds.eu-west-2.api.aws	HTTPS HTTPS

Region Name	Region	Endpoint	Protocol
Europe (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
		rds.eu-south-1.api.aws	HTTPS
Europe (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
		rds.eu-west-3.api.aws	HTTPS
Europe (Spain)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
		rds.eu-south-2.api.aws	HTTPS
Europe (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS
		rds.eu-north-1.api.aws	HTTPS
Europe (Zurich)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS
		rds.eu-central-2.api.aws	HTTPS
Israel (Tel Aviv)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS
		rds.il-central-1.api.aws	HTTPS
Mexico (Central)	mx-central-1	rds.mx-central-1.amazonaws.com	HTTPS
Middle East (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS
		rds.me-south-1.api.aws	HTTPS
Middle East (UAE)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS
		rds.me-central-1.api.aws	HTTPS

Region Name	Region	Endpoint	Protocol
South America (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com rds.sa-east-1.api.aws	HTTPS HTTPS
AWS GovCloud (US-East)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com rds.us-gov-east-1.api.aws	HTTPS HTTPS
AWS GovCloud (US-West)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com rds.us-gov-west-1.api.aws	HTTPS HTTPS

The following table shows the format that you should use when constructing an ARN for a particular Amazon RDS resource type.

Resource type	ARN format
DB instance	arn:aws:rds:*<region>*:*<account>* :db:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :db:my-mysql-instance-1`
DB cluster	arn:aws:rds:*<region>*:*<account>* :cluster:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :cluster: my-aurora-cluster-1`
Event subscription	arn:aws:rds:*<region>*:*<account>* :es:*<name>*

Resource type	ARN format
	For example: `arn:aws:rds: us-east-2 :123456789012 :es:my-subscription`
DB option group	arn:aws:rds:*<region>*:*<account>* :og:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :og:my-og`
DB parameter group	arn:aws:rds:*<region>*:*<account>* :pg:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :pg:my-param-enable-logs`
DB cluster parameter group	arn:aws:rds:*<region>*:*<account>* :cluster-pg:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :cluster-pg: my-cluster-param-timezone`
Reserved DB instance	arn:aws:rds:*<region>*:*<account>* :ri:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :ri:my-reserved-postgresql`

Resource type	ARN format
DB security group	arn:aws:rds:*<region>*:*<account>* :secgrp:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :secgrp:my-public`
Automated DB snapshot	arn:aws:rds:*<region>*:*<account>* :snapshot:rds:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :snapshot:rds: my-mysql-db-2019-07-22-07-23`
Automated DB cluster snapshot	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:rds:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:rds: my-aurora-cluster-2019-07-22-16-16`
Manual DB snapshot	arn:aws:rds:*<region>*:*<account>* :snapshot:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :snapshot: my-mysql-db-snap`
Manual DB cluster snapshot	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-aurora-cluster-snap`

Resource type	ARN format
DB subnet group	arn:aws:rds:*<region>*:*<account>* :subgrp:*<name>* For example: `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`

Getting an existing ARN for Amazon RDS

You can get the ARN of an RDS resource by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or RDS API.

Console

To get an ARN from the AWS Management Console, navigate to the resource you want an ARN for, and view the details for that resource.

For example, you can get the ARN for a DB instance from the **Configuration** tab of the DB instance details.
DB instance ARN.

AWS CLI

To get an ARN from the AWS CLI for a particular RDS resource, you use the `describe` command for that resource. The following table shows each AWS CLI command, and the ARN property used with the command to get an ARN.

AWS CLI command	ARN property
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn
describe-db-parameter-groups	DBParameterGroupArn
describe-db-cluster-parameter-groups	DBClusterParameterGroupArn

AWS CLI command	ARN property
describe-db-instances	DBInstanceArn
describe-db-security-groups	DBSecurityGroupArn
describe-db-snapshots	DBSnapshotArn
describe-events	SourceArn
describe-reserved-db-instances	ReservedDBInstanceArn
describe-db-subnet-groups	DBSubnetGroupArn
describe-option-groups	OptionGroupArn
describe-db-clusters	DBClusterArn
describe-db-cluster-snapshots	DBClusterSnapshotArn

For example, the following AWS CLI command gets the ARN for a DB instance.

Example

For Linux, macOS, or Unix:

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2 \
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

For Windows:

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

The output of that command is like the following:

```
[
    {
        "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",
        "DBInstanceIdentifier": "instance_id"
    }
]
```

RDS API

To get an ARN for a particular RDS resource, you can call the following RDS API operations and use the ARN properties shown following.

RDS API operation	ARN property
DescribeEventSubscriptions	EventSubscriptionArn
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeDBSecurityGroups	DBSecurityGroupArn
DescribeDBSnapshots	DBSnapshotArn
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDBInstanceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeOptionGroups	OptionGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Working with storage for Amazon RDS DB instances

To specify how you want your data stored in Amazon RDS, choose a storage type and provide a storage size when you create or modify a DB instance. Later, you can increase the amount or change the type of storage by modifying the DB instance. For more information about which storage type to use for your workload, see Amazon RDS storage types.

If your instances run RDS for Oracle or RDS for SQL Server, you can add up to three additional volumes to each DB instance. You can choose either gp3 or io2 as the volume type, allowing you to optimize costs and performance based on your data access patterns. The maximum storage capacity of a DB instance that uses additional volumes is 256 TiB.

Topics

- Viewing storage volume details for your DB instance
- Increasing DB instance storage capacity
- Removing additional storage volumes
- Managing capacity automatically with Amazon RDS storage autoscaling
- Upgrading the storage file system for a DB instance
- Modifying settings for Provisioned IOPS SSD storage
- I/O-intensive storage modifications
- Modifying settings for General Purpose SSD (gp3) storage
- Using a dedicated log volume (DLV)

Viewing storage volume details for your DB instance

You can view your storage volume configuration from the AWS Management Console or AWS CLI. This includes details about both your primary storage volume and any additional storage volumes attached to your DB instance.

Console

To view your storage volume configuration from the console:

1. Sign in to the AWS Management Console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.

3. Choose your DB instance.
4. Choose the **Configuration** tab to view details about your storage volumes. The storage information is located in the following sections:

   - **Primary storage**
   - **Additional storage volumes**

   In **Additional storage volumes**, you can choose **Add storage volume** to add another volume.

CLI

To view your storage volume configuration from the AWS CLI, use the `describe-db-instances` command.

```
aws rds describe-db-instances --db-instance-identifier my-database
```

In the output, find the `AdditionalStorageVolumes` array to view details for the added volumes.

```
    "AdditionalStorageVolumes": [
        {
            "VolumeName": "rdsdbdata2",
            "StorageVolumeStatus": "Not-in-use",
            "AllocatedStorage": 5000,
            "IOPS": 25000,
            "StorageThroughput": 500,
            "StorageType": "gp3"
        }
    ]
```

The `StorageVolumeStatus` field indicates whether the volume is currently in use by your database. A status of `Not-in-use` means the volume is attached but it's not in use by the database engine or an RDS feature.

Increasing DB instance storage capacity

To increase storage capacity for your DB instance, use either of the following techniques:

Scale up the storage volume

Every DB instance has a primary storage volume with a maximum capacity of 64 TiB. You can increase the allocated space on a storage volume by a minimum of 10%. You can't deallocate space.

Add up to three storage volumes (io2 or gp3)

You can increase the total storage capacity of an RDS for Oracle or RDS for SQL Server instance by adding up to three volumes to each DB instance. Each additional volume allocates up to 64 TiB of storage, up to a maximum of 256 TiB per instance.

> **Note**
>
> For RDS for Oracle DB instances, you can add a storage volume with the minimum storage size of 200 GiB.

You can use the Amazon RDS Management Console, the Amazon RDS API, or the AWS Command Line Interface (AWS CLI) for the preceding techniques. For information about storage limits, see Amazon RDS DB instance storage.

Scaling up DB instance storage

You can scale up the storage of an existing DB instance by increasing the allocated storage for the primary volume. When you increase the allocated storage, you must increase it by at least 10 percent. If you try to increase the value by less than 10 percent, you get an error. You can't reduce the amount of storage on a volume after you have allocated storage for it.

> **Note**
>
> For RDS for SQL Server DB instances, you can scale storage for only the General Purpose SSD and Provisioned IOPS SSD storage types.

To monitor the amount of free storage for your DB instance so you can respond when necessary, we recommend that you create an Amazon CloudWatch alarm. For more information on setting CloudWatch alarms, see Using CloudWatch alarms.

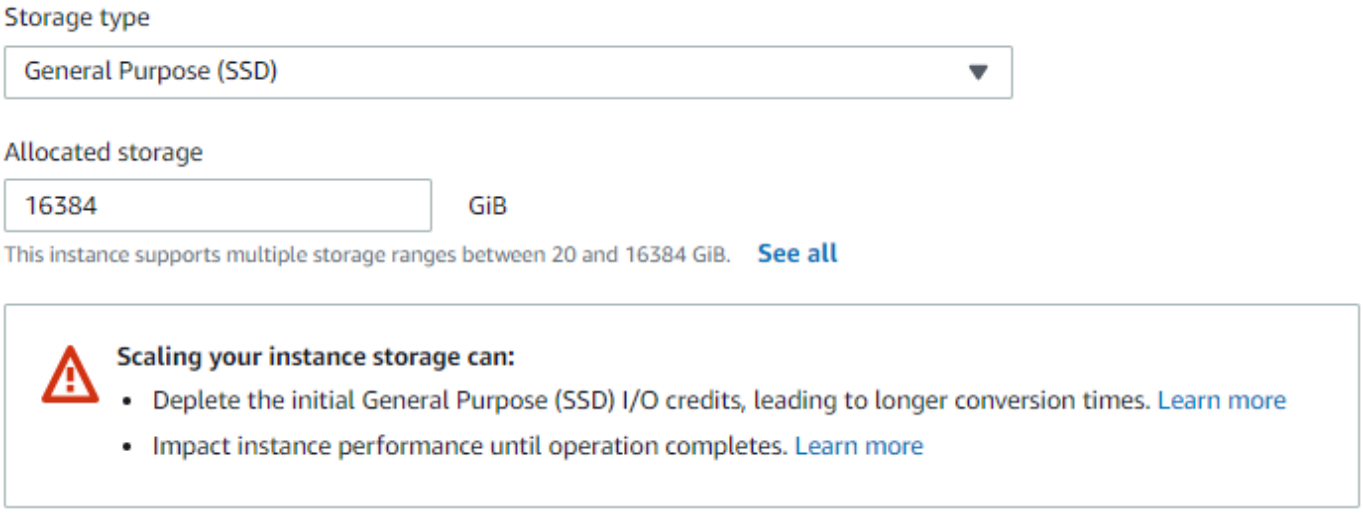
Scaling storage usually doesn't cause any outage or performance degradation of the DB instance. After you modify the storage size for a DB instance, the status of the DB instance is **storage-optimization**.

Storage optimization can take several hours. You can't make further storage modifications for either six (6) hours or until storage optimization has completed on the instance, whichever is longer. You can view the storage optimization progress in the AWS Management Console or by using the describe-db-instances AWS CLI command.

Console

To increase storage for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify.
4. Choose **Modify**.
5. Enter a new value for **Allocated storage**. It must be at least 10% greater than the current value.



6. Choose **Continue**.
7. Choose **Apply immediately** in the **Scheduling of modifications** section to apply the storage changes to the DB instance immediately.

   Or choose **Apply during the next scheduled maintenance window** to apply the changes during the next maintenance window.

8. When the settings are as you want them, choose **Modify DB instance**.

AWS CLI

To increase the storage for a DB instance, use the AWS CLI command `modify-db-instance`. Set the following parameters:

- `--allocated-storage` – Amount of storage to be allocated for the DB instance, in gibibytes.
- `--apply-immediately` – Use `--apply-immediately` to apply the storage changes immediately.

  Or use `--no-apply-immediately` (the default) to apply the changes during the next maintenance window. An immediate outage occurs when the changes are applied.

The following example scales up the storage for `mydbinstance` to 1,000 GiB and applies the change immediately. The command also migrates the storage volume to gp3 and sets the provisioned IOPS to 6000.

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --allocated-storage 1000 \
    --storage-type gp3 \
    --iops 6000 \
    --apply-immediately
```

For more information about storage, see Amazon RDS DB instance storage.

RDS API

To increase storage for a DB instance, use the Amazon RDS API operation `ModifyDBInstance`. Set the following parameters:

- `AllocatedStorage` – Amount of storage to be allocated for the DB instance, in gibibytes.
- `ApplyImmediately` – Set this option to `True` to apply the storage changes immediately. Set this option to `False` (the default) to apply the changes during the next maintenance window. An immediate outage occurs when the changes are applied.

For more information about storage, see Amazon RDS DB instance storage.

Adding storage volumes

For RDS for Oracle and RDS for SQL Server DB instances, you can add up to three storage volumes to increase your total storage capacity up to 256 TiB per instance. Additional storage volumes allow you to use different storage types (gp3 and io2) to optimize costs and performance based on your data access patterns.

> **Note**
>
> For RDS for Oracle DB instances, you can add a storage volume with the minimum storage size of 200 GiB.

You can add, modify, or remove additional storage volumes using the AWS Management Console or AWS CLI. You can configure the volumes with different allocated storage, IOPS, and throughput settings. For example, you might place high-performance data on an io2 volume and historical data on a gp3 volume.

The additional volumes must use the volume names shown in the following table.

RDS for Oracle volume name	RDS for SQL Server volume name
`rdsdbdata2`	`H:`
`rdsdbdata3`	`I:`
`rdsdbdata4`	`J:`

Console

To add an additional storage volume to a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify.
4. Choose **Modify**.

5. In the **Storage** section, choose **Add additional storage volume**.
6. Configure the additional storage volume:
   - **Volume name** – Choose `rdsdbdata2`, `rdsdbdata3`, or `rdsdbdata4`.
   - **Storage type** – Choose **General Purpose SSD (gp3)** or **Provisioned IOPS SSD (io2)**.
   - **Allocated storage** – Enter the storage size in GiB (minimum 200 GiB).
   - For io2 storage, configure **Provisioned IOPS**.
   - For gp3 storage, optionally configure **Storage throughput**.
7. Choose **Continue**.
8. When the settings are as you want them, choose **Modify DB instance**.

> **Important**
>
> When you are adding an additional storage volume using the `modify-db-instance` operation, the RDS adds the storage volume immediately regardless of the `--no-apply-immediately` parameter. If you have other modifications in the request, RDS applies them based on the schedule modifications. See Using the schedule modifications setting.

AWS CLI

To add an additional storage volume to a DB instance, use the AWS CLI command `modify-db-instance`. Set the `--additional-storage-volumes` as a JSON array specifying the additional storage volumes to add or modify.

The following example adds an additional storage volume named `rdsdbdata2` with 5000 GiB of gp3 storage to `mydbinstance`.

When you add an additional storage volume, RDS applies the change immediately regardless of the `--no-apply-immediately` parameter.

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --additional-storage-volumes '[
        {
            "VolumeName": "rdsdbdata2",
            "StorageType": "gp3",
```

```
            "AllocatedStorage": 5000,
            "StorageThroughput": 725
        }
    ]'
```

RDS API

To add an additional storage volume to a DB instance, use the Amazon RDS API operation `ModifyDBInstance`. Set the `AdditionalStorageVolumes` parameter as an array of additional storage volume specifications.

When you are adding an additional storage volume using the ModifyDBInstance API operation, RDS adds the storage volume immediately regardless of the `ApplyImmediately` option being True or False.

Removing additional storage volumes

You can remove additional storage volumes from RDS for Oracle and RDS for SQL Server DB instances when they are no longer needed. Before removing a volume, make sure that you have moved all database files off the volume and that no database objects are referencing it. Verify that the volume status is `Not-in-use`.

> **Important**
>
> You can't remove the primary storage volume. You can only remove additional storage volumes.

Console

To remove an additional storage volume from a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that includes the volume that you want to remove.
4. Choose **Modify**.
5. In the **Storage** section, locate the additional storage volume that you want to remove.

6. Choose **Remove volume** for the volume you want to delete.

> **Note**
>
> You can only remove volumes with a status of `Not-in-use`. If the volume is still in use, move all database files off the volume.

7. Choose **Continue**.
8. When the settings are as you want them, choose **Modify DB instance**.

AWS CLI

To remove an additional storage volume from a DB instance, use the AWS CLI command `modify-db-instance`. Set the following parameter:

- `--additional-storage-volumes` – JSON array specifying the remaining additional storage volumes. Omit the volume you want to remove from this array.

The following example removes the additional storage volume named `rdsdbdata3` from `mydbinstance` by specifying only the remaining volumes and applies the change immediately.

```
aws rds modify-db-instance \
 --db-instance-identifier mydbinstance \
 --additional-storage-volumes '[{ \
  "VolumeName": "rdsdbdata3", \
  "SetForDelete": true
 }]'
```

RDS API

To remove an additional storage volume from a DB instance, use the Amazon RDS API operation `ModifyDBInstance`. Set the following parameters:

- `AdditionalStorageVolumes` – Array of additional storage volume specifications for the volumes you want to keep. Omit the volume you want to remove from this array.
- `ApplyImmediately` – Set this option to `True` to apply the storage changes immediately. Set this option to `False` (the default) to apply the changes during the next maintenance window.

Managing capacity automatically with Amazon RDS storage autoscaling

If your workload is unpredictable, you can enable storage autoscaling for an Amazon RDS DB instance. To do so, you can use the Amazon RDS console, the Amazon RDS API, or the AWS CLI.

> **Note**
>
> Storage autoscaling isn't supported for additional storage volumes.

For example, you might use this feature for a new mobile gaming application that users are adopting rapidly. In this case, a rapidly increasing workload might exceed the available database storage. To avoid having to manually scale up database storage, you can use Amazon RDS storage autoscaling.

With storage autoscaling enabled, when Amazon RDS detects that you are running out of free database space it automatically scales up your storage. Amazon RDS starts a storage modification for an autoscaling-enabled DB instance when these factors apply:

- Free available space is less than or equal to 10 percent of the allocated storage.
- The low-storage condition lasts at least five minutes.
- At least six hours have passed since the last storage modification, or storage optimization has completed on the instance, whichever is longer.

The additional storage is in increments of whichever of the following is greater:

- 10 GiB
- 10 percent of currently allocated storage
- Predicted storage growth exceeding the current allocated storage size in the next 7 hours based on the `FreeStorageSpace` metrics from the past hour. For more information on metrics, see Monitoring with Amazon CloudWatch.

The maximum storage threshold is the limit that you set for autoscaling the DB instance. It has the following constraints:

- You must set the maximum storage threshold to at least 10% more than the current allocated storage. We recommend setting it to at least 26% more to avoid receiving an event notification that the storage size is approaching the maximum storage threshold.

  For example, if you have DB instance with 1000 GiB of allocated storage, then set the maximum storage threshold to at least 1100 GiB. If you don't, you get an error such as Invalid max storage size for *engine_name*. However, we recommend that you set the maximum storage threshold to at least 1260 GiB to avoid the event notification.
- For a DB instance that uses Provisioned IOPS (io1 or io2 Block Express) storage, the ratio of IOPS to maximum storage threshold (in GiB) must be within a certain range. For more information, see Provisioned IOPS SSD storage.
- You can't set the maximum storage threshold for autoscaling-enabled instances to a value greater than the maximum allocated storage for the database engine and DB instance class.

  For example, SQL Server Standard Edition on db.m5.xlarge has a default allocated storage for the instance of 20 GiB (the minimum) and a maximum allocated storage of 16,384 GiB. The default maximum storage threshold for autoscaling is 1,000 GiB. If you use this default, the instance doesn't autoscale above 1,000 GiB. This is true even though the maximum allocated storage for the instance is 16,384 GiB.

> **Note**
>
> We recommend that you carefully choose the maximum storage threshold based on usage patterns and customer needs. If there are any aberrations in the usage patterns, the maximum storage threshold can prevent scaling storage to an unexpectedly high value when autoscaling predicts a very high threshold. After a DB instance has been autoscaled, its allocated storage can't be reduced.

Topics

- Limitations of storage autoscaling
- Enabling storage autoscaling for a new DB instance
- Changing the storage autoscaling settings for a DB instance
- Manually scaling your DB instance down or in
- Turning off storage autoscaling for a DB instance

Limitations of storage autoscaling

The following limitations apply to storage autoscaling:

- Autoscaling doesn't occur if the maximum storage threshold would be exceeded by the storage increment.
- When autoscaling, RDS predicts the storage size for subsequent autoscaling operations. If a subsequent operation is predicted to exceed the maximum storage threshold, then RDS autoscales to the maximum storage threshold.
- Autoscaling can't completely prevent storage-full situations for large data loads. This is because further storage modifications can't be made for either six (6) hours or until storage optimization has completed on the instance, whichever is longer.

  If you perform a large data load, and autoscaling doesn't provide enough space, the database might remain in the storage-full state for several hours. This can harm the database.
- If you start a storage scaling operation at the same time that Amazon RDS starts an autoscaling operation, your storage modification takes precedence. The autoscaling operation is canceled.
- Autoscaling can't decrease the allocated storage. You can't reduce the amount of storage for a DB instance after storage has been allocated.
- Autoscaling can't be used with magnetic storage.
- Autoscaling can't be used with the following previous-generation instance classes that have less than 6 TiB of orderable storage: db.m3.large, db.m3.xlarge, and db.m3.2xlarge.
- Autoscaling operations aren't logged by AWS CloudTrail. For more information on CloudTrail, see Monitoring Amazon RDS API calls in AWS CloudTrail.
- Autoscaling isn't supported for additional storage volumes. For more information about additional storage volumes, see Adding storage volumes.

Although automatic scaling helps you to increase storage on your Amazon RDS DB instance dynamically, you should still configure the initial storage for your DB instance to an appropriate size for your typical workload.

Enabling storage autoscaling for a new DB instance

When you create a new Amazon RDS DB instance, you can choose whether to enable storage autoscaling. You can also set an upper limit on the storage that Amazon RDS can allocate for the DB instance.

Enabling storage autoscaling doesn't require a database reboot and doesn't cause any downtime. The feature takes effect immediately without disrupting database operations.

> **Note**
>
> When you clone an Amazon RDS DB instance that has storage autoscaling enabled, that setting isn't automatically inherited by the cloned instance. The new DB instance has the same amount of allocated storage as the original instance. You can turn storage autoscaling on again for the new instance if the cloned instance continues to increase its storage requirements.

Console

To enable storage autoscaling for a new DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region where you want to create the DB instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**. On the **Select engine** page, choose your database engine and specify your DB instance information as described in Getting started with Amazon RDS.
5. In the **Storage autoscaling** section, set the **Maximum storage threshold** value for the DB instance.
6. Specify the rest of your DB instance information as described in Getting started with Amazon RDS.

AWS CLI

To enable storage autoscaling for a new DB instance, use the AWS CLI command `create-db-instance`. Set the following parameter:

- `--max-allocated-storage` – Turns on storage autoscaling and sets the upper limit on storage size, in gibibytes.

To verify that Amazon RDS storage autoscaling is available for your DB instance, use the AWS CLI `describe-valid-db-instance-modifications` command. To check based on the instance class before creating an instance, use the `describe-orderable-db-instance-options` command. Check the following field in the return value:

- `SupportsStorageAutoscaling` – Indicates whether the DB instance or instance class supports storage autoscaling.

For more information about storage, see Amazon RDS DB instance storage.

RDS API

To enable storage autoscaling for a new DB instance, use the Amazon RDS API operation `CreateDBInstance`. Set the following parameter:

- `MaxAllocatedStorage` – Turns on Amazon RDS storage autoscaling and sets the upper limit on storage size, in gibibytes.

To verify that Amazon RDS storage autoscaling is available for your DB instance, use the Amazon RDS API `DescribeValidDbInstanceModifications` operation for an existing instance, or the `DescribeOrderableDBInstanceOptions` operation before creating an instance. Check the following field in the return value:

- `SupportsStorageAutoscaling` – Indicates whether the DB instance supports storage autoscaling.

For more information about storage, see Amazon RDS DB instance storage.

Changing the storage autoscaling settings for a DB instance

You can turn storage autoscaling on for an existing Amazon RDS DB instance. You can also change the upper limit on the storage that Amazon RDS can allocate for the DB instance.

Changing storage autoscaling settings doesn't require a database reboot and doesn't cause any downtime. The changes take effect immediately without disrupting database operations.

Console

To change the storage autoscaling settings for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify, and choose **Modify**. The **Modify DB instance** page appears.
4. Change the storage limit in the **Autoscaling** section. For more information, see Modifying an Amazon RDS DB instance.
5. When all the changes are as you want them, choose **Continue** and check your modifications.
6. On the confirmation page, review your changes. If they're correct, choose **Modify DB instance** to save your changes. If they aren't correct, choose **Back** to edit your changes or **Cancel** to cancel your changes.

   Changing the storage autoscaling limit occurs immediately. This setting ignores the **Apply immediately** setting.

AWS CLI

To change the storage autoscaling settings for a DB instance, use the AWS CLI command `modify-db-instance`. Set the following parameter:

- `--max-allocated-storage` – Sets the upper limit on storage size, in gibibytes. If the value is greater than the `--allocated-storage` parameter, storage autoscaling is turned on. If the value is the same as the `--allocated-storage` parameter, storage autoscaling is turned off.

To verify that Amazon RDS storage autoscaling is available for your DB instance, use the AWS CLI `describe-valid-db-instance-modifications` command. To check based on the instance class before creating an instance, use the `describe-orderable-db-instance-options` command. Check the following field in the return value:

- `SupportsStorageAutoscaling` – Indicates whether the DB instance supports storage autoscaling.

For more information about storage, see Amazon RDS DB instance storage.

RDS API

To change the storage autoscaling settings for a DB instance, use the Amazon RDS API operation `ModifyDBInstance`. Set the following parameter:

- `MaxAllocatedStorage` – Sets the upper limit on storage size, in gibibytes.

To verify that Amazon RDS storage autoscaling is available for your DB instance, use the Amazon RDS API `DescribeValidDbInstanceModifications` operation for an existing instance, or the `DescribeOrderableDBInstanceOptions` operation before creating an instance. Check the following field in the return value:

- `SupportsStorageAutoscaling` – Indicates whether the DB instance supports storage autoscaling.

For more information about storage, see Amazon RDS DB instance storage.

Manually scaling your DB instance down or in

Amazon RDS provides storage autoscaling to meet growing demand. However, there are limitations regarding scaling down and in:

- **RDS storage** – While RDS supports automatic scaling up of storage as demand increases, it doesn't automatically scale down.
- **Read replicas** – RDS doesn't support automatic scaling out (adding) or scaling in (deleting) of read replicas. You must manually add or remove read replicas according to your load requirements.

To scale down your RDS resources, perform the following manual actions:

- For storage, you can't manually reduce the allocated storage of a DB instance using the `modify-db-instance` command. Instead, choose one of the following techniques:
  - Use a blue/green deployment if your DB engine supports it. Create a green database with a lower storage size, and then promote your green database to be your blue database. For more information, see Modify storage and performance settings.
  - Create a new DB instance with lower allocated storage, manually migrate the data from your current database to the newly created database instance, and switch your database endpoints.

- For read replicas, manually delete any unused replicas through the RDS console or AWS CLI.

Turning off storage autoscaling for a DB instance

If you no longer need Amazon RDS to automatically increase the storage for an Amazon RDS DB instance, you can turn off storage autoscaling. After you do, you can still manually increase the amount of storage for your DB instance.

Console

To turn off storage autoscaling for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify and choose **Modify**. The **Modify DB instance** page appears.
4. Clear the **Enable storage autoscaling** check box in the **Storage autoscaling** section. For more information, see Modifying an Amazon RDS DB instance.
5. When all the changes are as you want them, choose **Continue** and check the modifications.
6. On the confirmation page, review your changes. If they're correct, choose **Modify DB instance** to save your changes. If they aren't correct, choose **Back** to edit your changes or **Cancel** to cancel your changes.

Changing the storage autoscaling limit occurs immediately. This setting ignores the **Apply immediately** setting.

AWS CLI

To turn off storage autoscaling for a DB instance, use the AWS CLI command `modify-db-instance` and the following parameter:

- `--max-allocated-storage` – Specify a value equal to the `--allocated-storage` setting to prevent further Amazon RDS storage autoscaling for the specified DB instance.

For more information about storage, see Amazon RDS DB instance storage.

RDS API

To turn off storage autoscaling for a DB instance, use the Amazon RDS API operation `ModifyDBInstance`. Set the following parameter:

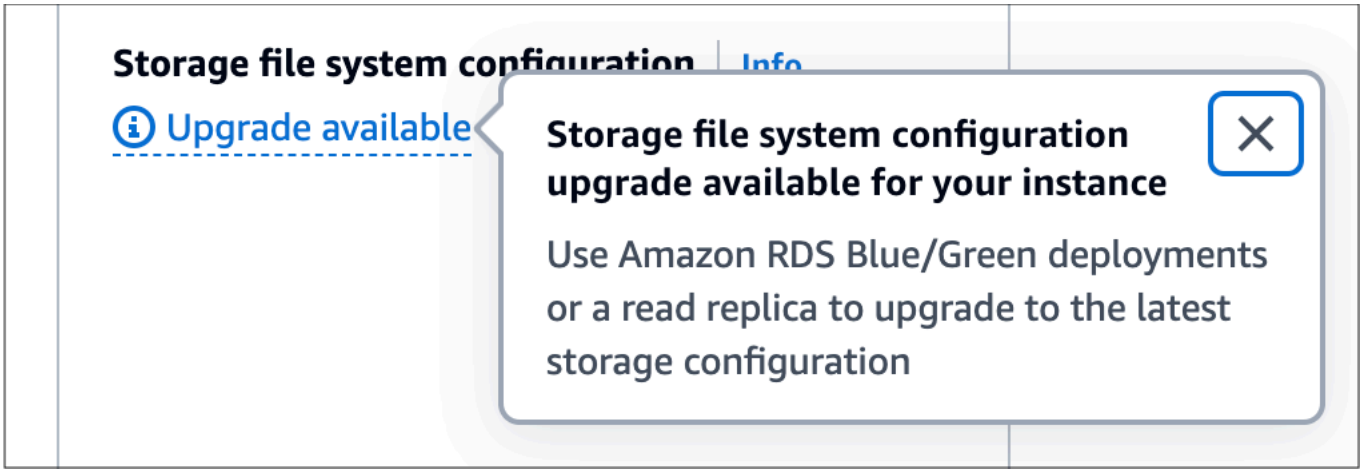
- `MaxAllocatedStorage` – Specify a value equal to the `AllocatedStorage` setting to prevent further Amazon RDS storage autoscaling for the specified DB instance.

For more information about storage, see Amazon RDS DB instance storage.

Upgrading the storage file system for a DB instance

Most RDS DB instances offer a maximum storage size of 64 TiB for RDS for MariaDB, MySQL, and PostgreSQL databases. However, some older 32-bit file systems have lower storage capacities. To determine the storage capacity of your DB instance, use the describe-valid-db-instance-modifications AWS CLI command.

RDS checks whether your storage system has a 16 TiB storage size, a file size limit of 2 TiB, or non-optimized writes. If your DB instances meet these conditions, RDS alerts you that your file system configuration is eligible for an upgrade. You can check the upgrade eligibility of a DB instance on the **Storage** panel of the DB instance details page.



If your DB instance is eligible for a file system upgrade, use either of the following techniques:

- Create a blue/green deployment and specify **Upgrade storage file system configuration**.

  This option upgrades the file system in the green environment to the preferred configuration. You can then switch over the blue/green deployment, which promotes the green environment to

be the new production environment. For detailed instructions, see the section called “Creating a blue/green deployment”.

- Create a DB instance read replica and specify **Upgrade storage file system configuration**.

  This option upgrades the file system of the read replica to the preferred configuration. You can then promote the read replica to be a standalone instance. For detailed instructions, see the section called “Creating a read replica”.

During the storage upgrade, your database engine isn't available. Upgrading the storage configuration is an I/O-intensive operation and leads to longer creation times for read replicas and blue/green deployments. The storage upgrade process is faster when both of the following conditions are met:

- The source DB instance uses Provisioned IOPS SSD (io1 or io2 Block Express) storage.
- You provisioned the green environment or read replica with an instance size of 4xlarge or larger.

Storage upgrades involving General Purpose SSD (gp2) storage can deplete your I/O credit balance, resulting in longer upgrade times. For more information, see the section called “DB instance storage”.

During a storage upgrade, RDS increases the allocated storage size by 10% for the green instance or read replica if both of the following conditions are met:

- The storage consumption on your source DB instance is greater than or equal to 90% of the allocated storage size.
- Storage autoscaling is enabled.

RDS turns off autoscaling when the new storage size is greater than or equal to the maximum allocated storage that was set for the instance. If storage autoscaling is disabled before the storage upgrade begins, the storage size doesn't increase during the upgrade.

Modifying settings for Provisioned IOPS SSD storage

You can modify the settings for a DB instance that uses Provisioned IOPS SSD storage by using the Amazon RDS console, AWS CLI, or Amazon RDS API. Specify the storage type, allocated storage, and the amount of Provisioned IOPS that you require. The range depends on your database engine and instance type.

Although you can reduce the amount of IOPS provisioned for your instance, you can't reduce the storage size.

In most cases, scaling storage doesn't require any outage and doesn't degrade performance of the server. After you modify the storage IOPS for a DB instance, the status of the DB instance is **storage-optimization**.

> **Note**
>
> Storage optimization can take several hours. You can't make further storage modifications for either six (6) hours or until storage optimization has completed on the instance, whichever is longer.

For information on the ranges of allocated storage and Provisioned IOPS available for each database engine, see Provisioned IOPS SSD storage.

Console

To change the Provisioned IOPS settings for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.

   To filter the list of DB instances, for **Filter databases** enter a text string for Amazon RDS to use to filter the results. Only DB instances whose names contain the string appear.
3. Choose the DB instance with Provisioned IOPS that you want to modify.
4. Choose **Modify**.
5. On the **Modify DB instance** page, choose **Provisioned IOPS SSD (io1)** or **Provisioned IOPS SSD (io2)** for **Storage type**.
6. For **Provisioned IOPS**, enter a value.

   If the value that you specify for either **Allocated storage** or **Provisioned IOPS** is outside the limits supported by the other parameter, a warning message is displayed. This message gives the range of values required for the other parameter.
7. Choose **Continue**.

8. Choose **Apply immediately** in the **Scheduling of modifications** section to apply the changes to the DB instance immediately. Or choose **Apply during the next scheduled maintenance window** to apply the changes during the next maintenance window.
9. Review the parameters to be changed, and choose **Modify DB instance** to complete the modification.

   The new value for allocated storage or for Provisioned IOPS appears in the **Status** column.

AWS CLI

To change the Provisioned IOPS setting for a DB instance, use the AWS CLI command `modify-db-instance`. Set the following parameters:

- `--storage-type` – Set to `io1` or `io2` for Provisioned IOPS.
- `--allocated-storage` – Amount of storage to be allocated for the DB instance, in gibibytes.
- `--iops` – The new amount of Provisioned IOPS for the DB instance, expressed in I/O operations per second.
- `--apply-immediately` – Use `--apply-immediately` to apply changes immediately. Use `--no-apply-immediately` (the default) to apply changes during the next maintenance window.

RDS API

To change the Provisioned IOPS settings for a DB instance, use the Amazon RDS API operation `ModifyDBInstance`. Set the following parameters:

- `StorageType` – Set to `io1` or `io2` for Provisioned IOPS.
- `AllocatedStorage` – Amount of storage to be allocated for the DB instance, in gibibytes.
- `Iops` – The new IOPS rate for the DB instance, expressed in I/O operations per second.
- `ApplyImmediately` – Set this option to `True` to apply changes immediately. Set this option to `False` (the default) to apply changes during the next maintenance window.

I/O-intensive storage modifications

Amazon RDS DB instances use Amazon Elastic Block Store (EBS) volumes for database and log storage. Depending on the amount of storage requested, RDS (except for RDS for SQL Server) automatically *stripes* across multiple Amazon EBS volumes to enhance performance. RDS DB

instances with SSD storage types are backed by either one or four striped Amazon EBS volumes in a RAID 0 configuration. By design, storage modification operations for an RDS DB instance have minimal impact on ongoing database operations.

In most cases, storage scaling modifications are completely offloaded to the Amazon EBS layer and are transparent to the database. This process is typically completed within a few minutes. However, some older RDS storage volumes require a different process for modifying the size, Provisioned IOPS, or storage type. This involves making a full copy of the data using a potentially I/O-intensive operation.

Storage modification uses an I/O-intensive operation if any of the following factors apply:

- The source storage type is magnetic. Magnetic storage doesn't support elastic volume modification.
- The RDS DB instance isn't on a one- or four-volume Amazon EBS layout. You can view the number of Amazon EBS volumes in use on your RDS DB instances by using Enhanced Monitoring metrics. For more information, see Viewing OS metrics in the RDS console.
- The target size of the modification request increases the allocated storage above 400 GiB for RDS for MariaDB, MySQL, and PostgreSQL instances, and 200 GiB for RDS for Oracle. Storage autoscaling operations have the same effect when they increase the allocated storage size of your DB instance above these thresholds.

If your storage modification involves an I/O-intensive operation, it consumes I/O resources and increases the load on your DB instance. Storage modifications with I/O-intensive operations involving General Purpose SSD (gp2) storage can deplete your I/O credit balance, resulting in longer conversion times.

We recommend as a best practice to schedule these storage modification requests outside of peak hours to help reduce the time required to complete the storage modification operation. Alternatively, you can create a read replica of the DB instance and perform the storage modification on the read replica. Then promote the read replica to be the primary DB instance. For more information, see Working with DB instance read replicas.

For more information, see Why is an Amazon RDS DB instance stuck in the modifying state when I try to increase the allocated storage?

Modifying settings for General Purpose SSD (gp3) storage

You can modify the settings for a DB instance that uses General Purpose SSD (gp3) storage by using the Amazon RDS console, AWS CLI, or Amazon RDS API. Specify the storage type, allocated storage, amount of Provisioned IOPS, and storage throughput that you require.

Although you can reduce the amount of Provisioned IOPS and storage throughput for your DB instance, you can't reduce the storage size.

In most cases, scaling storage doesn't require any outage. After you modify the storage IOPS for a DB instance, the status of the DB instance is **storage-optimization**. You can expect elevated latencies, but still within the single-digit millisecond range, during storage optimization. The DB instance is fully operational after a storage modification.

> **Note**
>
> You can't make further storage modifications until six (6) hours after storage optimization has completed on the instance.

For information on the ranges of allocated storage, Provisioned IOPS, and storage throughput available for each database engine, see gp3 storage (recommended).

Console

To change the storage performance settings for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.

   To filter the list of DB instances, for **Filter databases** enter a text string for Amazon RDS to use to filter the results. Only DB instances whose names contain the string appear.
3. Choose the DB instance with gp3 storage that you want to modify.
4. Choose **Modify**.
5. On the **Modify DB instance page**, choose **General Purpose SSD (gp3)** for **Storage type**, then do the following:

   a. For **Provisioned IOPS**, choose a value.

If the value that you specify for either **Allocated storage** or **Provisioned IOPS** is outside the limits supported by the other parameter, a warning message appears. This message gives the range of values required for the other parameter.

b. For **Storage throughput**, choose a value.

If the value that you specify for either **Provisioned IOPS** or **Storage throughput** is outside the limits supported by the other parameter, a warning message appears. This message gives the range of values required for the other parameter.

6. Choose **Continue**.
7. Choose **Apply immediately** in the **Scheduling of modifications** section to apply the changes to the DB instance immediately. Or choose **Apply during the next scheduled maintenance window** to apply the changes during the next maintenance window.
8. Review the parameters to be changed, and choose **Modify DB instance** to complete the modification.

The new value for Provisioned IOPS appears in the **Status** column.

AWS CLI

To change the storage performance settings for a DB instance, use the AWS CLI command `modify-db-instance`. Set the following parameters:

- `--storage-type` – Set to gp3 for General Purpose SSD (gp3).
- `--allocated-storage` – Amount of storage to be allocated for the DB instance, in gibibytes.
- `--iops` – The new amount of Provisioned IOPS for the DB instance, expressed in I/O operations per second.
- `--storage-throughput` – The new storage throughput for the DB instance, expressed in MiBps.
- `--apply-immediately` – Use `--apply-immediately` to apply changes immediately. Use `--no-apply-immediately` (the default) to apply changes during the next maintenance window.

RDS API

To change the storage performance settings for a DB instance, use the Amazon RDS API operation `ModifyDBInstance`. Set the following parameters:

- `StorageType` – Set to gp3 for General Purpose SSD (gp3).
- `AllocatedStorage` – Amount of storage to be allocated for the DB instance, in gibibytes.
- `Iops` – The new IOPS rate for the DB instance, expressed in I/O operations per second.
- `StorageThroughput` – The new storage throughput for the DB instance, expressed in MiBps.
- `ApplyImmediately` – Set this option to `True` to apply changes immediately. Set this option to `False` (the default) to apply changes during the next maintenance window.

Using a dedicated log volume (DLV)

You can use a dedicated log volume (DLV) for a DB instance that uses Provisioned IOPS (PIOPS) storage. A DLV moves PostgreSQL database transaction logs and MySQL/MariaDB redo logs and binary logs to a storage volume that's separate from the volume containing the database tables. A DLV makes transaction write logging more efficient and consistent. DLVs are ideal for databases with large allocated storage, high I/O per second (IOPS) requirements, or latency-sensitive workloads.

DLVs are supported for PIOPS storage (io1 and io2 Block Express) and are created with a fixed size of 1,024 GiB and 3,000 Provisioned IOPS.

Amazon RDS supports DLVs in all AWS Regions for the following versions:

- MariaDB 10.6.7 and higher 10 versions
- MySQL 8.0.28 and higher 8.0 versions, MySQL 8.4.3 and higher 8.4 versions
- PostgreSQL 13.10 and higher 13 versions, 14.7 and higher 14 versions, and 15.2 and higher 15 versions

RDS supports DLVs with Multi-AZ deployments. When you modify or create a Multi-AZ instance, a DLV is created for both the primary and the secondary.

RDS supports DLVs with read replicas. If the primary DB instance has a DLV enabled, all read replicas created after enabling DLV will also have a DLV. Any read replicas created before the switch to DLV will not have it enabled unless explicitly modified to do so. We recommend all read replicas attached to a primary instance before DLV was enabled also be manually modified to have A DLV.

> **Note**
>
> We recommend DLVs for database configurations of 5 TiB or greater.

For more information on the benefits of DLVs, see the following blog posts:

- Enhance database performance with Amazon RDS dedicated log volumes
- Benchmark Amazon RDS for PostgreSQL with dedicated log volumes
- Maximizing performance of AWS RDS for MySQL with dedicated log volumes in the Percona documentation

For information on the ranges of allocated storage, Provisioned IOPS, and storage throughput available for each database engine, see Provisioned IOPS SSD storage.

Topics

- Considerations when enabling and disabling DLV
- Enabling DLV when you create a DB instance
- Enabling DLV on an existing DB instance
- Monitoring DLV storage

Considerations when enabling and disabling DLV

Enabling and disabling DLV can be time consuming and cause downtime. The process involves copying all transaction logs or redo and binary logs (depending on the database engine) to the new volume when enabling, or back to the original storage when disabling. The duration of this operation is influenced by several factors:

- Number of transaction logs:
  - Larger databases with more transactions generate more logs, increasing the time required for copying.
  - Transaction logs can accumulate on the primary DB instance if replication slots are inactive or if replication is lagging, increasing the time required for copying. Make sure that replication is current, and remove any unnecessary slots.
- Storage configuration:

- DB instance EBS bandwidth – Higher bandwidth allows for faster data transfer.
- Number of Provisioned IOPS – More input/output operations per second (IOPS) can speed up the copying process.
- Database activity – High levels of database activity during configuration can slow down the process.

To minimize downtime, we recommend that you plan and schedule during periods of low activity or maintenance windows.

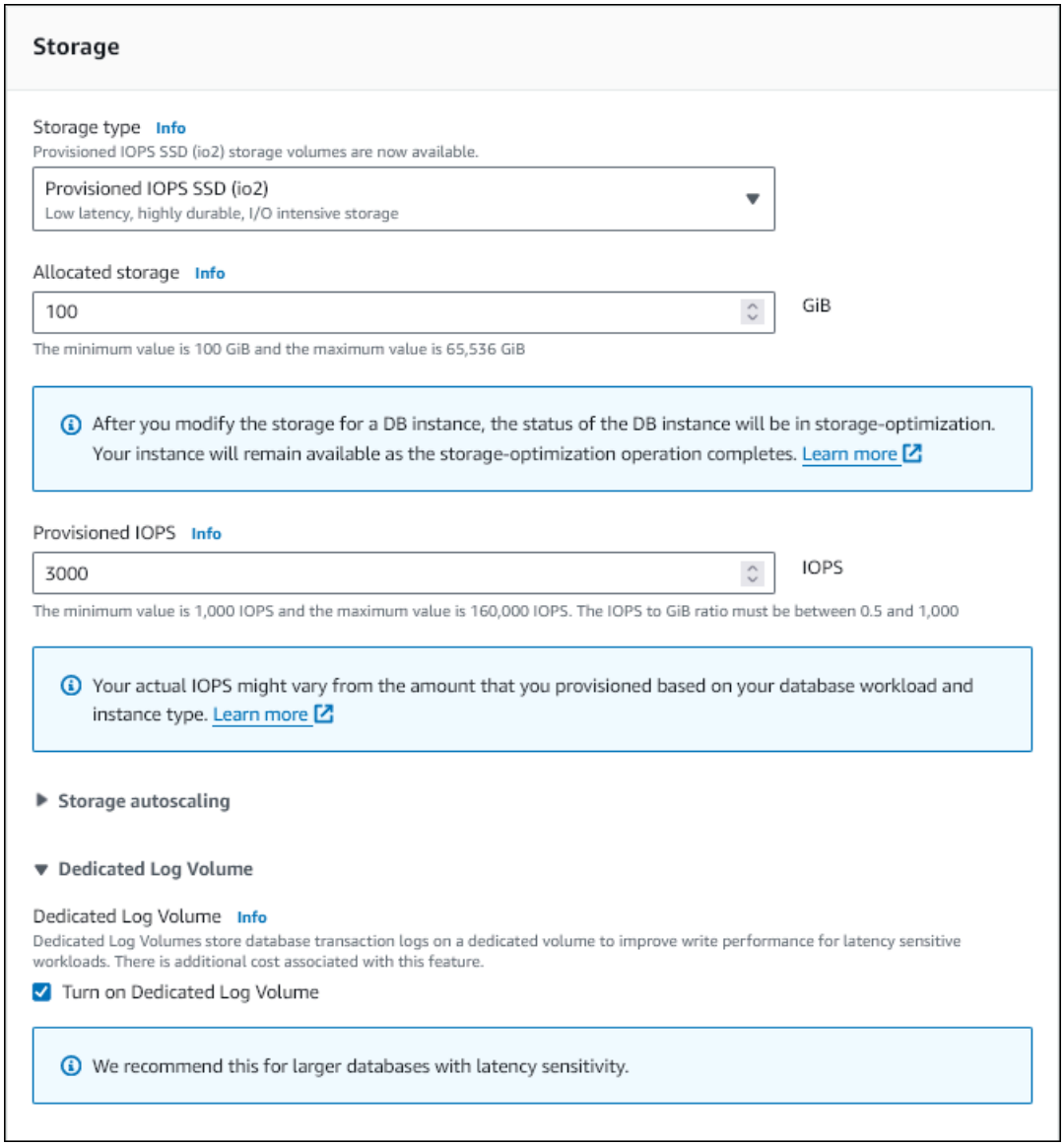
Enabling DLV when you create a DB instance

You can use the AWS Management Console, AWS CLI, or RDS API to create a DB instance with DLV enabled.

Console

To enable DLV on a new DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Create database**.
3. On the **Create DB instance page**, choose a DB engine that supports DLV.
4. For **Storage**:
    a. Choose either **Provisioned IOPS SSD (io1)** or **Provisioned IOPS SSD (io2)**.
    b. Enter the **Allocated storage** and **Provisioned IOPS** that you want.
    c. Expand **Dedicated Log Volume**, then select **Turn on Dedicated Log Volume**.



5. Choose other settings as needed.
6. Choose **Create database**.

After the database is created, the value for Dedicated Log Volume appears on the **Configuration** tab of the database details page.

CLI

To enable DLV when you create a DB instance using Provisioned IOPS storage, use the AWS CLI command create-db-instance. Set the following parameters:

- `--dedicated-log-volume` – Enables a dedicated log volume.
- `--storage-type` – Set to `io1` or `io2` for Provisioned IOPS.
- `--allocated-storage` – Amount of storage to be allocated for the DB instance, in gibibytes.
- `--iops` – The amount of Provisioned IOPS for the DB instance, expressed in I/O operations per second.

RDS API

To enable DLV when you create a DB instance using Provisioned IOPS storage, use the Amazon RDS API operation CreateDBInstance. Set the following parameters:

- `DedicatedLogVolume` – Set to `true` to enable a dedicated log volume.
- `StorageType` – Set to `io1` or `io2` for Provisioned IOPS.
- `AllocatedStorage` – Amount of storage to be allocated for the DB instance, in gibibytes.
- `Iops` – The IOPS rate for the DB instance, expressed in I/O operations per second.

Enabling DLV on an existing DB instance

You can use the AWS Management Console, AWS CLI, or RDS API to modify a DB instance to enable DLV.

After you modify the DLV setting for a DB instance, you must reboot the DB instance.

Console

To enable DLV on an existing DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.

   To filter the list of DB instances, for **Filter databases** enter a text string for Amazon RDS to use to filter the results. Only DB instances whose names contain the string appear.
3. Choose the DB instance with Provisioned IOPS storage that you want to modify.
4. Choose **Modify**.
5. On the **Modify DB instance page**:

- For **Storage**, expand **Dedicated Log Volume**, then select **Turn on Dedicated Log Volume**.

6. Choose **Continue**.
7. Choose **Apply immediately** to apply the changes to the DB instance immediately. Or choose **Apply during the next scheduled maintenance window** to apply the changes during the next maintenance window.
8. Review the parameters to be changed, and choose **Modify DB instance** to complete the modification.

The new value for Dedicated Log Volume appears on the **Configuration** tab of the database details page.

CLI

To enable or disable DLV on an existing DB instance using Provisioned IOPS storage, use the AWS CLI command `modify-db-instance`. Set the following parameters:

- `--dedicated-log-volume` – Enables a dedicated log volume.

  Use `--no-dedicated-log-volume` (the default) to disable a dedicated log volume.
- `--apply-immediately` – Use `--apply-immediately` to apply changes immediately.

  Use `--no-apply-immediately` (the default) to apply changes during the next maintenance window.

RDS API

To enable or disable DLV on an existing DB instance using Provisioned IOPS storage, use the Amazon RDS API operation `ModifyDBInstance`. Set the following parameters:

- `DedicatedLogVolume` – Set this option to `true` to enable a dedicated log volume.

  Set this option to `false` to disable a dedicated log volume. This is the default value.
- `ApplyImmediately` – Set this option to `True` to apply changes immediately.

  Set this option to `False` (the default) to apply changes during the next maintenance window.

Monitoring DLV storage

You can monitor the DLV storage usage by using the `FreeStorageSpaceLogVolume` metric in CloudWatch.

You can use the following query for RDS for PostgreSQL to find the size occupied by transaction logs:

```
SELECT pg_size_pretty(COALESCE(sum(size), 0)) AS total_wal_generated_size
FROM pg_catalog.pg_ls_waldir();
```

If the DLV runs out of storage, the DB instance will enter the `storage-full` state, causing downtime.

Deleting a DB instance

> **Warning**
>
> Deleting a database instance permanently removes all its data and can affect related resources. Ensure that you have backed up any important data before proceeding.

You can delete a DB instance using the AWS Management Console, the AWS CLI, or the RDS API. If you want to delete a DB instance in an Aurora DB cluster, see Deleting Aurora DB clusters and DB instances.

Topics

- Prerequisites for deleting a DB instance
- Considerations when deleting a DB instance
- Deleting a DB instance

Prerequisites for deleting a DB instance

Before you try to delete your DB instance, make sure that deletion protection is turned off. By default, deletion protection is turned on for a DB instance that was created with the console.

If your DB instance has deletion protection turned on, you can turn it off by modifying your instance settings. Choose **Modify** in the database details page or call the modify-db-instance command. This operation doesn't cause an outage. For more information, see Settings for DB instances.

Considerations when deleting a DB instance

Deleting a DB instance has an effect on instance recoverability, backup availability, and read replica status. Consider the following issues:

- You can choose whether to create a final DB snapshot. You have the following options:
  - If you take a final snapshot, you can use it to restore your deleted DB instance. RDS retains both the final snapshot and any manual snapshots that you took previously. You can't create a final DB snapshot of your DB instance if it isn't in the `Available` state. For more information, see Viewing Amazon RDSDB instance status.

  - If you don't take a final snapshot, then no final snapshot exists that you can restore later. If you decide to restore your deleted DB instance, either retain automated backups or use an earlier manual snapshot to restore your DB instance to the point in time of the earlier snapshot.
- You can choose whether to retain automated backups. You have the following options:
  - If you retain automated backups, RDS keeps them for the retention period that is in effect for the DB instance at the time when you delete it. You can use automated backups to restore your DB instance to a time during but not after your retention period. The retention period is in effect regardless of whether you create a final DB snapshot. To delete a retained automated backup, see Deleting retained automated backups.
  - Retained automated backups and manual snapshots incur billing charges until they're deleted. For more information, see Retention costs.
  - If you don't retain automated backups, RDS deletes the automated backups that reside in the same AWS Region as your DB instance. You can't recover these backups. If your automated backups have been replicated to another AWS Region, RDS keeps them even if you don't choose to retain automated backups. For more information, see Replicating automated backups to another AWS Region.
- When you delete your DB instance, RDS doesn't delete manual DB snapshots. For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.
- If you want to delete all RDS resources, note that the following resources incur billing charges:
  - DB instances
  - DB snapshots
  - DB clusters

  If you purchased reserved instances, then they are billed according to contract that you agreed to when you purchased the instance. For more information, see Reserved DB instances for Amazon RDS . You can get billing information for all your AWS resources by using the AWS Cost Explorer. For more information, see Analyzing your costs with AWS Cost Explorer.
- If you delete a DB instance that has read replicas in the same AWS Region, each read replica is automatically promoted to a standalone DB instance. For more information, see Promoting a read replica to be a standalone DB instance. If your DB instance has read replicas in different AWS Regions, see Cross-Region replication considerations for information related to deleting the source DB instance for a cross-Region read replica.

- When the status for a DB instance is `deleting`, its CA certificate value doesn't appear in the RDS console or in output for AWS CLI commands or RDS API operations. For more information about CA certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
- The time required to delete a DB instance varies depending on the backup retention period, how much data is deleted, whether a final snapshot is taken, and whether the final snapshot is incremental or full. Even if you don't take a final snapshot, the service must perform internal operations to ensure data consistency. These operations might extend the delete time. They can't be skipped. The DB instance status remains in a deleting state until all operations are complete. To reduce the time required to delete the DB instance, we recommend that you take a manual snapshot before deleting the DB instance.

Deleting a DB instance

You can delete a DB instance using the AWS Management Console, the AWS CLI, or the RDS API. You must do the following:

- Provide the name of the DB instance
- Enable or disable the option to take a final DB snapshot of the instance
- Enable or disable the option to retain automated backups

> **Note**
>
> You can't delete a DB instance when deletion protection is turned on. For more information, see Prerequisites for deleting a DB instance.

Console

To delete a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to delete.
3. For **Actions**, choose **Delete**.
4. To create a final DB snapshot for the DB instance, choose **Create final snapshot?**.

5. If you chose to create a final snapshot, enter the **Final snapshot name**.
6. To retain automated backups, choose **Retain automated backups**.
7. Enter **`delete me`** in the box.
8. Choose **Delete**.

AWS CLI

To find the instance IDs of the DB instances in your account, call the [describe-db-instances](#) command:

```
aws rds describe-db-instances --query 'DBInstances[*].[DBInstanceIdentifier]' --output
 text
```

To delete a DB instance by using the AWS CLI, call the [delete-db-instance](#) command with the following options:

- `--db-instance-identifier`
- `--final-db-snapshot-identifier` or `--skip-final-snapshot`

Example With a final snapshot and no retained automated backups

For Linux, macOS, or Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance \
    --final-db-snapshot-identifier mydbinstancefinalsnapshot \
    --delete-automated-backups
```

For Windows:

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance ^
    --final-db-snapshot-identifier mydbinstancefinalsnapshot ^
    --delete-automated-backups
```

Example With retained automated backups and no final snapshot

For Linux, macOS, or Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance \
    --skip-final-snapshot \
    --no-delete-automated-backups
```

For Windows:

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance ^
    --skip-final-snapshot ^
    --no-delete-automated-backups
```

RDS API

To delete a DB instance by using the Amazon RDS API, call the `DeleteDBInstance` operation with the following parameters:

- `DBInstanceIdentifier`
- `FinalDBSnapshotIdentifier` or `SkipFinalSnapshot`

Tutorial: Managing a MySQL DB instance environment from development to production

Introduction

A common scenario when managing an Amazon RDS DB instance involves the oversight of its lifecycle from initial development through to production deployment. This tutorial offers guidance to handle key tasks to ensure your database performs optimally and adapts to meet your evolving operational needs. Additionally, it outlines options to synchronize changes made between your development and production environments to ensure consistency and reliability.

By completing these steps, you learn:

- How to perform specific tasks with MySQL DB instances, such as adding and updating Amazon RDS tags, expanding storage, creating read replicas, and deleting resources.
- How to synchronize updates from a production environment to a development environment for comprehensive testing and validation.

To complete this tutorial, carry out the following tasks:

1. Create a MySQL DB instance.
2. Add Amazon RDS tags to categorize your DB instance as a development environment.
3. Increase the storage capacity of your DB instance to accommodate increased workloads.
4. Create read replicas to enhance the resilience and availability of your DB instance.
5. Update Amazon RDS tags to categorize your DB instance as a production environment.
6. Delete DB instance that you no longer need so that they don't incur additional costs.
7. Next Steps: Synchronize your development instance with production for consistency across environments

Prerequisites

Before you begin, complete the steps in the following sections:

- Sign up for an AWS account
- Create a user with administrative access

Add tags to categorize your DB instance as a development environment

To categorize the DB instance as a development environment, add an Amazon RDS tag to the instance you created. An Amazon RDS tag is a key-value pair that you define and associate with your RDS instance. Tagging your AWS resources helps distinguish between your development and production AWS resources. For more information on Amazon RDS tags, see Tagging Amazon RDS resources.

1. In the Amazon RDS console, choose **Databases**.
2. Select the DB instance you want to tag.
3. In the details section, scroll to the **Tags** section.
4. Choose **Manage tags** and select**dd new tag**.
5. Enter a value for **Tag key** and **Value**. For instance, you might use the tag key environment with the value dev to specify that the database instance is part of the development environment.
6. Choose **Add new tag**and **Save changes**.

Your DB instance is now tagged as a development environment. This makes it easier to identify the DB instance and to manage costs associated with this resource.

Increase the storage capacity of a DB instance to accommodate growing data needs

Next, modify the storage capacity of the MySQL DB instance to accommodate additional data. Initially, the storage capacity of your DB instance is set to meet the immediate needs of your application. However, as data volumes grow, it might be necessary to adjust the storage settings to ensure the database's continued performance and stability. This process involves increasing the allocated storage of your DB instance. For more information on modifying the storage capacity of your DB instance, see Working with storage for Amazon RDS DB instances.

1. In the Amazon RDS console, choose **Databases**.
2. Select the DB instance you want to modify.
3. Choose **Modify**.
4. In Storage, increase the **Allocated Storage**. The modified storage value must be greater than the current one.
5. Choose **Continue**.

6. In **Scheduling of modifications**, you can either choose **Apply immediately** to apply the storage changes to the DB instance immediately or choose **Apply during the next scheduled maintenance window** to apply the changes during the next maintenance window.
7. When the settings are as you want them, choose **Modify DB instance**.

The storage capacity of your DB instance is now increased. This enables it to effectively handle larger data volumes and ensures continued performance and stability as your application's data needs grow.

Create read replicas to enhance the resilience and availability of a DB instance

Create a read replica of the MySQL DB instance. Read replicas enhance the resilience and availability of your DB instance. To reduce the read traffic on your primary DB instance, create a read replica of your DB instance. This routes queries to the read replica, which can help distribute the load and improve overall database performance. For more information on DB instance read replicas, see Working with DB instance read replicas.

Before a MySQL DB instance can serve as a replication source, automatic backups must be enabled on the source DB instance. This can be done by setting the backup retention period to a value other than 0. For more information on MySQL read replicas, see Working with MySQL read replicas.

1. In the Amazon RDS console, choose **Databases**.
2. Select the DB instance you want to use as the source for the read replica.
3. In Actions, select **Create read replica**.
4. For **DB instance identifier**, enter a name for the read replica in all lowercase letters.
5. Choose your instance configuration. We recommend that you use the same or larger DB instance class and storage type as the source DB instance for the read replica.
6. For **AWS Region**, specify the destination Region for the read replica.
7. Leave the default settings or modify them as you require.
8. Choose **Create read replica**.

The read replica appears underneath your source DB instance on the **Databases** page in the RDS console. It shows **Replica** in the Role column.

Update tags to categorize a DB instance as a production environment

When your DB instance is ready to move from the development phase to production, it is important to update its tags to reflect its transition. To align your DB instance with your operational and monitoring strategies, update the initial tags to indicate that the DB instance is now part of the production environment. This ensures better visibility and management of the database.

1. In the Amazon RDS console, choose **Databases**.
2. Select the DB instance you want to update
3. In the details section, scroll to the **Tags** section.
4. Select **Manage tags**.
5. **Remove** your initial tag signifying a development environment.
6. Select**Add new tag**.
7. Enter a new value for **Tag key** and **Value**. For instance, you might use the tag key environment with the value prod to specify that the DB instance is part of the production environment.
8. Choose **Add new tag** and **Save changes**.

The tag on your DB instance is updated to signify the database's transition to a production environment.

Delete a DB instance when it is no longer needed to avoid incurring additional costs

Before the end of this tutorial, it is crucial to address the management of your resources. If you have any resources that are no longer required, you should proceed to delete them to prevent incurring additional costs and optimize your cloud environment.

1. In the Amazon RDS console, choose **Databases**.
2. Select the DB instance you want to delete
3. In Actions, select **Delete**. Deleting a DB instance will permanently delete the instance with all its content and related resources.
4. Confirm the deletion of the DB instance and select **Delete**.

Alternatively, if you choose to maintain your DB instance for future use, you can continue to manage it as part of your production environment. This involves maintaining a synchronized development environment to facilitate comprehensive testing and validation. For more information, see Next steps: Synchronize your development instance with production for consistency across environments.

Next steps: Synchronize your development instance with production for consistency across environments

Create a development environment

To manage a production environment, it's important to maintain a synchronized development environment for comprehensive testing and validation. To create a new development environment, first create a DB snapshot of the current production DB instance. A DB snapshot captures the entire DB instance by creating a storage volume snapshot. For instructions on how to create a DB snapshot on the Amazon RDS console, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

After you create the DB snapshot of your production environment, create a new DB instance for your development environment by restoring a DB snapshot. Restored DB instances are automatically associated with the default DB parameter and option groups. However, you can apply a custom parameter group and option group by specifying them during a restore. For instructions on restoring a DB snapshot, see Tutorial: Restore an Amazon RDS DB instance from a DB snapshot.

Finally, designate the new DB instance as your new development environment by updating its Amazon RDS tags. For guidance on updating Amazon RDS tags to reflect this change, see the previous section Update tags to categorize a DB instance as a production environment.

You now have a new development environment that mirrors the database configuration of your production environment.

Synchronize a development environment with production environment

Once your new development environment is established, it is necessary to keep it synchronized with any changes that occur in the production environment. This ensures that your development environment accurately reflects the current state of production, which is essential for effective testing, validation, and troubleshooting. Amazon RDS provides a variety of different ways to keep your development environment up to date with your production environment. For more

information on these options, see Orchestrating database refreshes for Amazon RDS and Amazon Aurora.

One of the primary ways in which you can synchronize your development and production environments is through creating and restoring DB snapshots. A DB snapshot allows you to create a development environment that reflects the database configuration of the production environment during the time the snapshot was created. For more information on DB snapshots, see Managing manual backups. For more information on restoring a DB instance, see Restoring to a DB instance.

DB snapshots are particularly valuable for the following use cases.

- Initial setup of a development environment: DB snapshots are useful to create the initial development environment for testing as it provides a consistent baseline that mirrors the exact state of the production environment at the time of the snapshot.
- High traffic applications: In production environments where continuous operation is critical, using Multi-AZ deployments for snapshots avoids I/O suspension on the primary database, ensuring uninterrupted performance and high availability.
- Sharing data across different RDS accounts: DB snapshots can be shared across different AWS accounts, facilitating the transfer of data between accounts or regions. This is useful for collaborative projects or scenarios where data needs to be shared for various purposes. For more information, see Sharing a DB snapshot for Amazon RDS.

In this tutorial, you explored essential tasks for managing your DB instance throughout its lifecycle. You learned how to create a DB instance, add and update Amazon RDS tags, expand storage, and create read replicas. You also learned ways to build on these fundamental operations and manage your production environment effectively. This included establishing a development environment for testing and synchronizing it with the production environment for consistency. These tasks help maintain a resilient and scalable database infrastructure, ensuring your Amazon RDS environment operates efficiently.

Configuring and managing a Multi-AZ deployment for Amazon RDS

Multi-AZ deployments can have one standby or two standby DB instances. When the deployment has one standby DB instance, it's called a *Multi-AZ DB instance deployment*. A Multi-AZ DB instance deployment has one standby DB instance that provides failover support, but doesn't serve read traffic. When the deployment has two standby DB instances, it's called a *Multi-AZ DB cluster deployment*. A Multi-AZ DB cluster deployment has standby DB instances that provide failover support and can also serve read traffic.

You can use the AWS Management Console to determine whether a Multi-AZ deployment is a Multi-AZ DB instance deployment or a Multi-AZ DB cluster deployment. In the navigation pane, choose **Databases**, and then choose a **DB identifier**.

- A Multi-AZ DB instance deployment has the following characteristics:
  - There is only one row for the DB instance.
  - The value of **Role** is **Instance** or **Primary**.
  - The value of **Multi-AZ** is **Yes**.
- A Multi-AZ DB cluster deployment has the following characteristics:
  - There is a cluster-level row with three DB instance rows under it.
  - For the cluster-level row, the value of **Role** is **Multi-AZ DB cluster**.
  - For each instance-level row, the value of **Role** is **Writer instance** or **Reader instance**.
  - For each instance-level row, the value of **Multi-AZ** is **3 Zones**.

Topics

- Multi-AZ DB instance deployments for Amazon RDS
- Multi-AZ DB cluster deployments for Amazon RDS

In addition, the following topics apply to both DB instances and Multi-AZ DB clusters.

- the section called “Tagging RDS resources”
- the section called “ARNs in Amazon RDS”
- the section called “Working with storage”

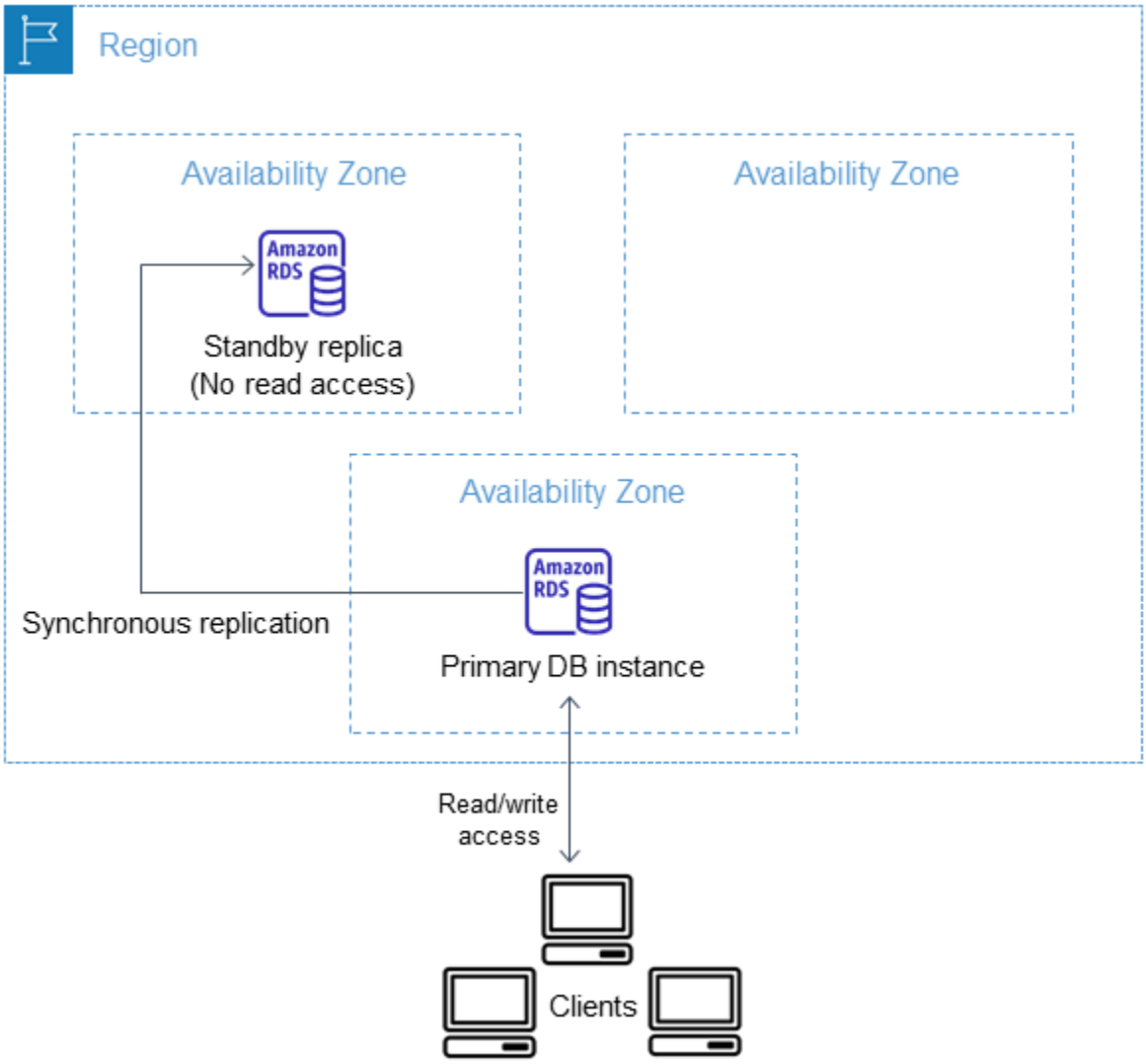
- the section called “Maintaining a DB instance”
- the section called “Upgrading the engine version”

Multi-AZ DB instance deployments for Amazon RDS

Amazon RDS provides high availability and failover support for DB instances using Multi-AZ deployments with a single standby DB instance. This type of deployment is called a *Multi-AZ DB instance deployment*. Amazon RDS uses several different technologies to provide this failover support. Multi-AZ deployments for MariaDB, MySQL, Oracle, PostgreSQL, and RDS Custom for SQL Server DB instances use the Amazon failover technology. Microsoft SQL Server DB instances use SQL Server Database Mirroring (DBM) or Always On Availability Groups (AGs). For information on SQL Server version support for Multi-AZ, see Multi-AZ deployments for Amazon RDS for Microsoft SQL Server. For information on working with RDS Custom for SQL Server for Multi-AZ, see Managing a Multi-AZ deployment for RDS Custom for SQL Server.

In a Multi-AZ DB instance deployment, Amazon RDS automatically provisions and maintains a synchronous standby replica in a different Availability Zone. The primary DB instance is synchronously replicated across Availability Zones to a standby replica to provide data redundancy and minimize latency spikes during system backups. Running a DB instance with high availability can enhance availability during planned system maintenance. It can also help protect your databases against DB instance failure and Availability Zone disruption. For more information on Availability Zones, see Regions, Availability Zones, and Local Zones.

> **Note**
>
> The high availability option isn't a scaling solution for read-only scenarios. You can't use a standby replica to serve read traffic. To serve read-only traffic, use a Multi-AZ DB cluster or a read replica instead. For more information about Multi-AZ DB clusters, see Multi-AZ DB cluster deployments for Amazon RDS. For more information about read replicas, see Working with DB instance read replicas.



Using the RDS console, you can create a Multi-AZ DB instance deployment by simply specifying Multi-AZ when creating a DB instance. You can use the console to convert existing DB instances to Multi-AZ DB instance deployments by modifying the DB instance and specifying the Multi-AZ option. You can also specify a Multi-AZ DB instance deployment with the AWS CLI or Amazon RDS API. Use the create-db-instance or modify-db-instance CLI command, or the CreateDBInstance or ModifyDBInstance API operation.

The RDS console shows the Availability Zone of the standby replica (called the secondary AZ). You can also use the describe-db-instances CLI command or the DescribeDBInstances API operation to find the secondary AZ.

DB instances using Multi-AZ DB instance deployments can have increased write and commit latency compared to a Single-AZ deployment. This can happen because of the synchronous data replication that occurs. You might have a change in latency if your deployment fails over to the

standby replica, although AWS is engineered with low-latency network connectivity between Availability Zones. For production workloads, we recommend that you use Provisioned IOPS (input/output operations per second) for fast, consistent performance. For more information about DB instance classes, see DB instance classes.

Converting a DB instance to a Multi-AZ deployment for Amazon RDS

Modifying a DB instance to a Multi-AZ deployment improves availability by adding a standby instance in another Availability Zone. The process involves minimal downtime and requires careful planning around storage and performance impacts. This change enhances fault tolerance and reduces recovery time in case of failures, making it ideal for high-availability environments.

If you have a DB instance in a Single-AZ deployment and modify it to a Multi-AZ DB instance deployment, Amazon RDS performs the following actions:

1. Takes a snapshot of the primary DB instance's Amazon Elastic Block Store (EBS) volumes.
2. Creates new volumes for the standby replica from the snapshot. These volumes initialize in the background, and maximum volume performance is achieved after the data is fully initialized.
3. Turns on synchronous block-level replication between the volumes of the primary and standby replicas.

> **Important**
>
> Creating a standby DB instance from a snapshot during a Single-AZ to Multi-AZ conversion avoids downtime but might impact performance, particularly for write-sensitive workloads. The synchronous replication can increase I/O latency, affecting database performance. As a best practice, avoid converting a production DB instance to a Multi-AZ DB instance. Instead, create a read replica, enable backups on it, convert it to Multi-AZ, load data into its volumes, and then promote it to the primary DB instance. For more information, see Working with DB instance read replicas.

There are two ways to modify a DB instance to be a Multi-AZ DB instance deployment:

Topics

- Convert to a Multi-AZ DB instance deployment with the RDS console
- Modifying a DB instance to be a Multi-AZ DB instance deployment

Convert to a Multi-AZ DB instance deployment with the RDS console

You can use the RDS console to convert a DB instance to a Multi-AZ DB instance deployment.

You can only use the console to complete the conversion. To use the AWS CLI or RDS API, follow the instructions in Modifying a DB instance to be a Multi-AZ DB instance deployment.

To convert to a Multi-AZ DB instance deployment with the RDS console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. From **Actions**, choose **Convert to Multi-AZ deployment**.
4. On the confirmation page, choose **Apply immediately** to apply the changes immediately. Choosing this option doesn't cause downtime, but there is a possible performance impact. Alternatively, you can choose to apply the update during the next maintenance window. For more information, see Using the schedule modifications setting.
5. Choose **Convert to Multi-AZ**.

Modifying a DB instance to be a Multi-AZ DB instance deployment

You can modify a DB instance to be a Multi-AZ DB instance deployment in the following ways:

- Using the RDS console, modify the DB instance, and set **Multi-AZ deployment** to **Yes**.
- Using the AWS CLI, call the modify-db-instance command, and set the `--multi-az` option.
- Using the RDS API, call the ModifyDBInstance operation, and set the `MultiAZ` parameter to `true`.

For information about modifying a DB instance, see Modifying an Amazon RDS DB instance. After the modification is complete, Amazon RDS triggers an event (RDS-EVENT-0025) that indicates the process is complete. You can monitor Amazon RDS events. For more information about events, see Working with Amazon RDS event notification.

Failing over a Multi-AZ DB instance for Amazon RDS

If a planned or unplanned outage of your Multi-AZ DB instance results from an infrastructure defect, Amazon RDS automatically switches to a standby replica in another Availability Zone.

The time that it takes for the failover to complete depends on the database activity and other conditions at the time the primary DB instance became unavailable. Failover times are typically 60–120 seconds. However, large transactions or a lengthy recovery process can increase failover time. When the failover is complete, it can take additional time for the RDS console to reflect the new Availability Zone.

> **Note**
>
> You can force a failover manually when you reboot a Multi-AZ DB instance. For more information, see Rebooting a DB instance.

Amazon RDS handles failovers automatically so you can resume database operations as quickly as possible without administrative intervention. The primary DB instance switches over automatically to the standby replica if any of the conditions described in the following table occurs. You can view these failover reasons in the event log.

Failover reason	Description
The operating system underlying the RDS database instance is being patched in an offline operation.	A failover was triggered during the maintenance window for an OS patch or a security update. For more information, see Maintaining a DB instance.
The primary host of the RDS Multi-AZ instance is unhealthy.	The Multi-AZ DB instance deployment detected an impaired primary DB instance and failed over.
The primary host of the RDS Multi-AZ instance is unreachable due to loss of network connectivity.	RDS monitoring detected a network reachability failure to the primary DB instance and triggered a failover.

Failover reason	Description
The RDS instance was modified by customer.	An RDS DB instance modification triggered a failover. For more information, see Modifying an Amazon RDS DB instance.

Failover reason	Description
The RDS Multi-AZ primary instance is busy and unresponsive.	The primary DB instance is unresponsive. We recommend that you do the following: • Examine the event and CloudWatch logs for excessive CPU, memory, or swap space usage. For more information, see Working with Amazon RDS event notification and Creating a rule that triggers on an Amazon RDS event. • Evaluate your workload to determine whether you're using the appropriate DB instance class. For more information, see DB instance classes. • Use Enhanced Monitoring for real-time operating system metrics. For more information, see Monitoring OS metrics with Enhanced Monitoring. • Use Performance Insights to help analyze any issues that affect your DB instance's performance. For more information, see Monitoring DB load with Performance Insights on Amazon RDS. For more information on these recommend ations, see Monitoring tools for Amazon RDS and Best practices for Amazon RDS.
The storage volume underlying the primary host of the RDS Multi-AZ instance experienced a failure.	The Multi-AZ DB instance deployment detected a storage issue on the primary DB instance and failed over.

Failover reason	Description
The user requested a failover of the DB instance.	You rebooted the DB instance and chose **Reboot with failover**. For more information, see Rebooting a DB instance.

To determine if your Multi-AZ DB instance has failed over, you can do the following:

- Set up DB event subscriptions to notify you by email or SMS that a failover has been initiated. For more information about events, see Working with Amazon RDS event notification.
- View your DB events by using the RDS console or API operations.
- View the current state of your Multi-AZ DB instance deployment by using the RDS console or API operations.

For information on how you can respond to failovers, reduce recovery time, and other best practices for Amazon RDS, see Best practices for Amazon RDS.

Setting the JVM TTL for DNS name lookups

The failover mechanism automatically changes the Domain Name System (DNS) record of the DB instance to point to the standby DB instance. As a result, you need to re-establish any existing connections to your DB instance. In a Java virtual machine (JVM) environment, due to how the Java DNS caching mechanism works, you might need to reconfigure JVM settings.

The JVM caches DNS name lookups. When the JVM resolves a host name to an IP address, it caches the IP address for a specified period of time, known as the *time-to-live* (TTL).

Because AWS resources use DNS name entries that occasionally change, we recommend that you configure your JVM with a TTL value of no more than 60 seconds. Doing this makes sure that when a resource's IP address changes, your application can receive and use the resource's new IP address by requerying the DNS.

On some Java configurations, the JVM default TTL is set so that it never refreshes DNS entries until the JVM is restarted. Thus, if the IP address for an AWS resource changes while your application is still running, it can't use that resource until you manually restart the JVM and the cached IP

information is refreshed. In this case, it's crucial to set the JVM's TTL so that it periodically refreshes its cached IP information.

You can get the JVM default TTL by retrieving the `networkaddress.cache.ttl` property value:

```
String ttl = java.security.Security.getProperty("networkaddress.cache.ttl");
```

> **Note**
>
> The default TTL can vary according to the version of your JVM and whether a security manager is installed. Many JVMs provide a default TTL less than 60 seconds. If you're using such a JVM and not using a security manager, you can ignore the rest of this topic. For more information on security managers in Oracle, see The security manager in the Oracle documentation.

To modify the JVM's TTL, set the `networkaddress.cache.ttl` property value. Use one of the following methods, depending on your needs:

- To set the property value globally for all applications that use the JVM, set `networkaddress.cache.ttl` in the `$JAVA_HOME/jre/lib/security/java.security` file.

  ```
  networkaddress.cache.ttl=60
  ```

- To set the property locally for your application only, set `networkaddress.cache.ttl` in your application's initialization code before any network connections are established.

  ```
  java.security.Security.setProperty("networkaddress.cache.ttl" , "60");
  ```

Multi-AZ failover with additional storage volumes

Multi-AZ deployments support DB instances with additional storage volumes. During a failover, RDS automatically fails over to the standby instance with any additional storage volumes attached to the DB instance. This process ensures data consistency and availability.

When you configure a Multi-AZ deployment for a DB instance with additional storage volumes, Amazon RDS automatically replicates all volumes to the standby instance in a different Availability Zone. The replicated storage includes:

- The primary storage volume
- All additional storage volumes attached to your DB instance

During a failover, Amazon RDS promotes the standby instance and ensures that all storage volumes are available and consistent. The failover maintains the same storage configuration, including volume names, storage types, and performance characteristics.

After a successful failover, you can verify that all storage volumes are properly attached and accessible by viewing the storage configuration details. For more information, see Viewing storage volume details for your DB instance.

The failover time for DB instances with additional storage volumes is similar to DB instances with only primary storage.

Multi-AZ DB cluster deployments for Amazon RDS

A *Multi-AZ DB cluster deployment* is a semisynchronous, high availability deployment mode of Amazon RDS with two readable replica DB instances. A Multi-AZ DB cluster has a writer DB instance and two reader DB instances in three separate Availability Zones in the same AWS Region. Multi-AZ DB clusters provide high availability, increased capacity for read workloads, and lower write latency when compared to Multi-AZ DB instance deployments.

You can import data from an on-premises database to a Multi-AZ DB cluster by following the instructions in Importing data to an Amazon RDS for MySQL database with reduced downtime.

You can purchase reserved DB instances for a Multi-AZ DB cluster. For more information, see Reserved DB instances for a Multi-AZ DB cluster.

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability of Amazon RDS with Multi-AZ DB clusters, see Supported Regions and DB engines for Multi-AZ DB clusters in Amazon RDS.

Topics

- Instance class availability for Multi-AZ DB clusters
- Multi-AZ DB cluster architecture
- Parameter groups for Multi-AZ DB clusters
- RDS Proxy with Multi-AZ DB clusters
- Replica lag and Multi-AZ DB clusters
- Multi-AZ DB cluster snapshots
- Creating a Multi-AZ DB cluster for Amazon RDS
- Connecting to a Multi-AZ DB cluster for Amazon RDS
- Automatically connecting an AWS compute resource and a Multi-AZ DB cluster for Amazon RDS
- Modifying a Multi-AZ DB cluster for Amazon RDS
- Upgrading the engine version of a Multi-AZ DB cluster for Amazon RDS
- Renaming a Multi-AZ DB cluster for Amazon RDS
- Rebooting a Multi-AZ DB cluster and reader DB instances for Amazon RDS
- Failing over a Multi-AZ DB cluster for Amazon RDS
- Setting up PostgreSQL logical replication with Multi-AZ DB clusters for Amazon RDS
- Working with Multi-AZ DB cluster read replicas for Amazon RDS

- Setting up external replication from Multi-AZ DB clusters for Amazon RDS
- Deleting a Multi-AZ DB cluster for Amazon RDS
- Limitations of Multi-AZ DB clusters for Amazon RDS

Important

Multi-AZ DB clusters aren't the same as Aurora DB clusters. For information about Aurora DB clusters, see the Amazon Aurora User Guide.

Instance class availability for Multi-AZ DB clusters

Multi-AZ DB cluster deployments are supported for the following DB instance classes: `db.m5d`, `db.m6gd`, `db.m6id`, `db.m6idn`, `db.r5d`, `db.r6gd`, `db.x2iedn`, `db.r6id`, and `db.r6idn`, and `db.c6gd`.

Note

The c6gd instance classes are the only ones that support the `medium` instance size.

For more information about DB instance classes, see the section called “DB instance classes”.

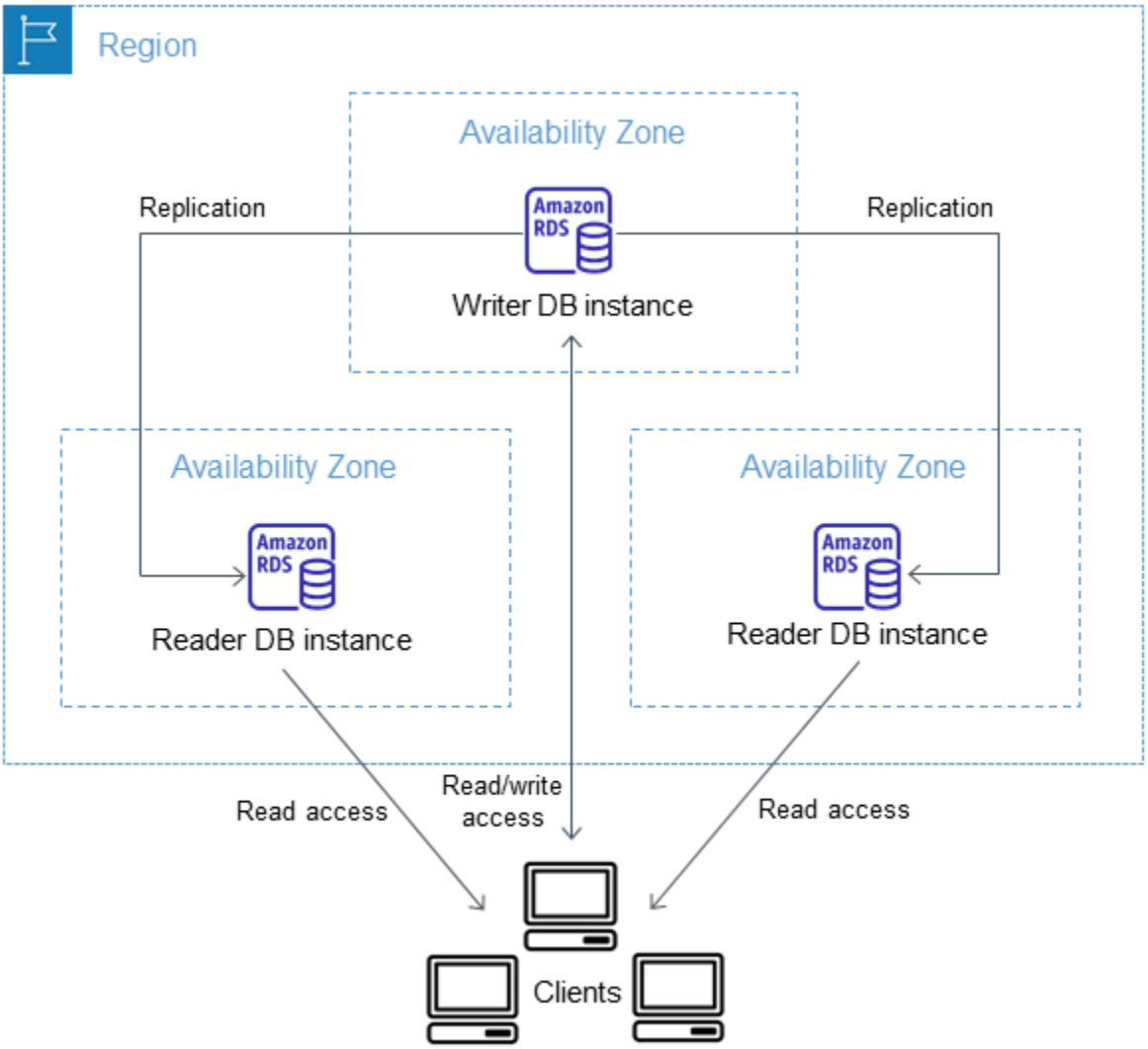
Multi-AZ DB cluster architecture

With a Multi-AZ DB cluster, Amazon RDS replicates data from the writer DB instance to both of the reader DB instances using the DB engine's native replication capabilities. When a change is made on the writer DB instance, it's sent to each reader DB instance.

Multi-AZ DB cluster deployments use semisynchronous replication, which requires acknowledgment from at least one reader DB instance in order for a change to be committed. It doesn't require acknowledgment that events have been fully executed and committed on *all* replicas.

Reader DB instances act as automatic failover targets and also serve read traffic to increase application read throughput. If an outage occurs on your writer DB instance, RDS manages failover to one of the reader DB instances. RDS does this based on which reader DB instance has the most recent change record.

The following diagram shows a Multi-AZ DB cluster.



Multi-AZ DB clusters typically have lower write latency when compared to Multi-AZ DB instance deployments. They also allow read-only workloads to run on reader DB instances. The RDS console shows the Availability Zone of the writer DB instance and the Availability Zones of the reader DB instances. You can also use the describe-db-clusters CLI command or the DescribeDBClusters API operation to find this information.

Important

To prevent replication errors in RDS for MySQL Multi-AZ DB clusters, we strongly recommend that all tables have a primary key.

Parameter groups for Multi-AZ DB clusters

In a Multi-AZ DB cluster, a *DB cluster parameter group* acts as a container for engine configuration values that are applied to every DB instance in the Multi-AZ DB cluster.

In a Multi-AZ DB cluster, a *DB parameter group* is set to the default DB parameter group for the DB engine and DB engine version. The settings in the DB cluster parameter group are used for all of the DB instances in the cluster.

For information about parameter groups, see the section called "DB cluster parameter groups".

RDS Proxy with Multi-AZ DB clusters

You can use Amazon RDS Proxy to create a proxy for your Multi-AZ DB clusters. By using RDS Proxy, your applications can pool and share database connections to improve their ability to scale. Each proxy performs connection *multiplexing*, also known as connection reuse. With multiplexing, RDS Proxy performs all the operations for a transaction using one underlying database connection. RDS Proxy can also reduce the downtime for a minor version upgrade of a Multi-AZ DB cluster to one second or less. For more information about the benefits of RDS Proxy, see *Amazon RDS Proxy*.

To set up a proxy for a Multi-AZ DB cluster, choose **Create an RDS Proxy** when creating the cluster. For instructions to create and manage RDS Proxy endpoints, see the section called "Working with RDS Proxy endpoints".

Replica lag and Multi-AZ DB clusters

Replica lag is the difference in time between the latest transaction on the writer DB instance and the latest applied transaction on a reader DB instance. The Amazon CloudWatch metric `ReplicaLag` represents this time difference. For more information about CloudWatch metrics, see Monitoring Amazon RDS metrics with Amazon CloudWatch.

Although Multi-AZ DB clusters allow for high write performance, replica lag can still occur due to the nature of engine-based replication. Because any failover must first resolve the replica lag before it promotes a new writer DB instance, monitoring and managing this replica lag is a consideration.

For RDS for MySQL Multi-AZ DB clusters, failover time depends on replica lag of both remaining reader DB instances. Both the reader DB instances must apply unapplied transactions before one of them is promoted to the new writer DB instance.

For RDS for PostgreSQL Multi-AZ DB clusters, failover time depends on the lowest replica lag of the two remaining reader DB instances. The reader DB instance with the lowest replica lag must apply unapplied transactions before it is promoted to the new writer DB instance.

For a tutorial that shows you how to create a CloudWatch alarm when replica lag exceeds a set amount of time, see Tutorial: Creating an Amazon CloudWatch alarm for Multi-AZ DB cluster replica lag for Amazon RDS.

Common causes of replica lag

In general, replica lag occurs when the write workload is too high for the reader DB instances to apply the transactions efficiently. Various workloads can incur temporary or continuous replica lag. Some examples of common causes are the following:

- High write concurrency or heavy batch updating on the writer DB instance, causing the apply process on the reader DB instances to fall behind.
- Heavy read workload that is using resources on one or more reader DB instances. Running slow or large queries can affect the apply process and can cause replica lag.
- Transactions that modify large amounts of data or DDL statements can sometimes cause a temporary increase in replica lag because the database must preserve commit order.

Mitigating replica lag

For Multi-AZ DB clusters for RDS for MySQL and RDS for PostgreSQL, you can mitigate replica lag by reducing the load on your writer DB instance. You can also use flow control to reduce replica lag. *Flow control* works by throttling writes on the writer DB instance, which ensures that replica lag doesn't continue to grow unbounded. Write throttling is accomplished by adding a delay into the end of a transaction, which decreases the write throughput on the writer DB instance. Although flow control doesn't guarantee lag elimination, it can help reduce overall lag in many workloads. The following sections provide information about using flow control with RDS for MySQL and RDS for PostgreSQL.

Mitigating replica lag with flow control for RDS for MySQL

When you are using RDS for MySQL Multi-AZ DB clusters, flow control is turned on by default using the dynamic parameter `rpl_semi_sync_master_target_apply_lag`. This parameter specifies the upper limit that you want for replica lag. As replica lag approaches this configured limit, flow control throttles the write transactions on the writer DB instance to try to contain the replica lag

below the specified value. In some cases, replica lag can exceed the specified limit. By default, this parameter is set to 120 seconds. To turn off flow control, set this parameter to its maximum value of 86,400 seconds (one day).

To view the current delay injected by flow control, show the parameter `Rpl_semi_sync_master_flow_control_current_delay` by running the following query.

```
SHOW GLOBAL STATUS like '%flow_control%';
```

Your output should look similar to the following.

```
+-----------------------------------------------+-------+
| Variable_name                                 | Value |
+-----------------------------------------------+-------+
| Rpl_semi_sync_master_flow_control_current_delay | 2010  |
+-----------------------------------------------+-------+
1 row in set (0.00 sec)
```

Note

The delay is shown in microseconds.

When you have Performance Insights turned on for an RDS for MySQL Multi-AZ DB cluster, you can monitor the wait event corresponding to a SQL statement indicating that the queries were delayed by a flow control. When a delay was introduced by a flow control, you can view the wait event `/wait/synch/cond/semisync/semi_sync_flow_control_delay_cond` corresponding to the SQL statement on the Performance Insights dashboard. To view these metrics, make sure that the Performance Schema is turned on. For information about Performance Insights, see Monitoring DB load with Performance Insights on Amazon RDS.

Mitigating replica lag with flow control for RDS for PostgreSQL

When you are using RDS for PostgreSQL Multi-AZ DB clusters, flow control is deployed as an extension. It turns on a background worker for all DB instances in the DB cluster. By default, the background workers on the reader DB instances communicate the current replica lag with the background worker on the writer DB instance. If the lag exceeds two minutes on any reader DB instance, the background worker on the writer DB instance adds a delay at the end of a transaction. To control the lag threshold, use the parameter `flow_control.target_standby_apply_lag`.

When a flow control throttles a PostgreSQL process, the `Extension` wait event in `pg_stat_activity` and Performance Insights indicates that. The function `get_flow_control_stats` displays details about how much delay is currently being added.

Flow control can benefit most online transaction processing (OLTP) workloads that have short but highly concurrent transactions. If the lag is caused by long-running transactions, such as batch operations, flow control doesn't provide as strong a benefit.

You can turn off flow control by removing the extension from the `shared_preload_libraries` and rebooting your DB instance.

Multi-AZ DB cluster snapshots

Amazon RDS creates and saves automated backups of your Multi-AZ DB cluster during the configured backup window. RDS creates a storage volume snapshot of your DB cluster, backing up the entire cluster and not just individual instances.

You can also take manual backups of your Multi-AZ DB cluster. For very long-term backups, consider exporting the snapshot data to Amazon S3. For more information, see the section called “Creating a Multi-AZ DB cluster snapshot”.

You can restore a Multi-AZ DB cluster to a specific point in time, creating a new Multi-AZ DB cluster. For instructions, see the section called “Restoring a Multi-AZ DB cluster to a specified time”.

Alternately, you can restore a Multi-AZ DB cluster snapshot to a Single-AZ deployment or Multi-AZ DB instance deployment. For instructions, see the section called “Restoring from a Multi-AZ DB cluster snapshot to a DB instance”.

Creating a Multi-AZ DB cluster for Amazon RDS

A Multi-AZ DB cluster has a writer DB instance and two reader DB instances in three separate Availability Zones. Multi-AZ DB clusters provide high availability, increased capacity for read workloads, and lower latency when compared to Multi-AZ deployments. For more information about Multi-AZ DB clusters, see Multi-AZ DB cluster deployments for Amazon RDS.

Note

Multi-AZ DB clusters are supported only for the MySQL and PostgreSQL DB engines.

DB cluster prerequisites

Important

Before you can create a Multi-AZ DB cluster, you must complete the tasks in Setting up your Amazon RDS environment.

The following are prerequisites to complete before creating a Multi-AZ DB cluster.

Topics

- Configure the network for the DB cluster
- Additional prerequisites

Configure the network for the DB cluster

You can create a Multi-AZ DB cluster only in a virtual private cloud (VPC) based on the Amazon VPC service. It must be in an AWS Region that has at least three Availability Zones. The DB subnet group that you choose for the DB cluster must cover at least three Availability Zones. This configuration ensures that each DB instance in the DB cluster is in a different Availability Zone.

To set up connectivity between your new DB cluster and an Amazon EC2 instance in the same VPC, do so when you create the DB cluster. To connect to your DB cluster from resources other than EC2 instances in the same VPC, configure the network connections manually.

Topics

- Configure automatic network connectivity with an EC2 instance
- Configure the network manually

Configure automatic network connectivity with an EC2 instance

When you create a Multi-AZ DB cluster, you can use the AWS Management Console to set up connectivity between an EC2 instance and the new DB cluster. When you do so, RDS configures your VPC and network settings automatically. The DB cluster is created in the same VPC as the EC2 instance so that the EC2 instance can access the DB cluster.

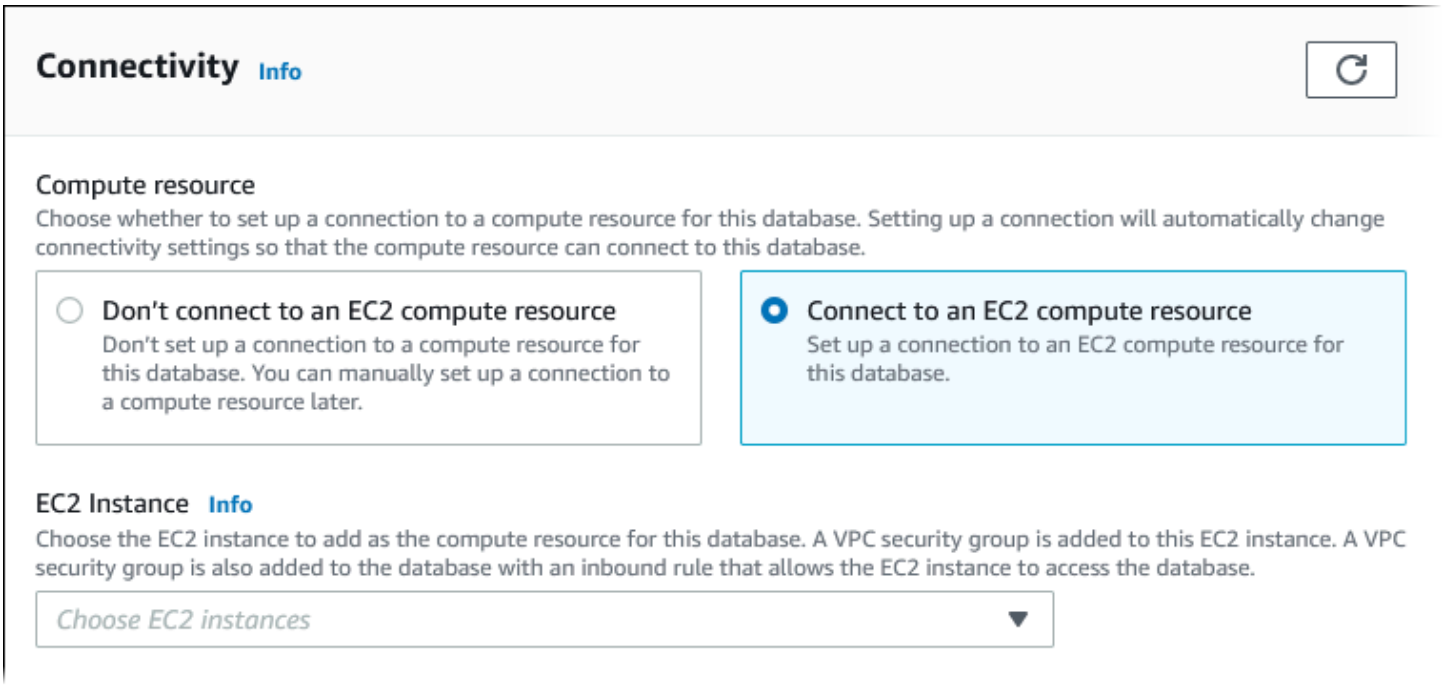
The following are requirements for connecting an EC2 instance with the DB cluster:

- The EC2 instance must exist in the AWS Region before you create the DB cluster.

  If no EC2 instances exist in the AWS Region, the console provides a link to create one.
- The user who is creating the DB cluster must have permissions to perform the following operations:
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Using this option creates a private DB cluster. The DB cluster uses a DB subnet group with only private subnets to restrict access to resources within the VPC.

To connect an EC2 instance to the DB cluster, choose **Connect to an EC2 compute resource** in the **Connectivity** section on the **Create database** page.



When you choose **Connect to an EC2 compute resource**, RDS sets the following options automatically. You can't change these settings unless you choose not to set up connectivity with an EC2 instance by choosing **Don't connect to an EC2 compute resource**.

Console option	Automatic setting
Virtual Private Cloud (VPC)	RDS sets the VPC to the one associated with the EC2 instance.
DB subnet group	RDS requires a DB subnet group with a private subnet in the same Availability Zone as the EC2 instance. If a DB subnet group that meets this requirement exists, then RDS uses the existing DB subnet group. By default, this option is set to **Automatic setup**. When you choose **Automatic setup** and there is no DB subnet group that meets this requirement, the following action happens. RDS uses three available private subnets in three Availability Zones where one of the Availability Zones is the same as the EC2 instance. If a private subnet isn't available in

Console option	Automatic setting
	an Availability Zone, RDS creates a private subnet in the Availability Zone. Then RDS creates the DB subnet group. When a private subnet is available, RDS uses the route table associated with the subnet and adds any subnets it creates to this route table. When no private subnet is available, RDS creates a route table without internet gateway access and adds the subnets it creates to the route table. RDS also allows you to use existing DB subnet groups. Select **Choose existing** if you want to use an existing DB subnet group of your choice.
Public access	RDS chooses **No** so that the DB cluster isn't publicly accessible. For security, it is a best practice to keep the database private and make sure it isn't accessible from the internet.

Console option	Automatic setting
VPC security group (firewall)	RDS creates a new security group that is associated with the DB cluster. The security group is named `rds-ec2-`*n*, where *n* is a number. This security group includes an inbound rule with the EC2 VPC security group (firewall) as the source. This security group that is associated with the DB cluster allows the EC2 instance to access the DB cluster. RDS also creates a new security group that is associated with the EC2 instance. The security group is named `ec2-rds-`*n*, where *n* is a number. This security group includes an outbound rule with the VPC security group of the DB cluster as the source. This security group allows the EC2 instance to send traffic to the DB cluster. You can add another new security group by choosing **Create new** and typing the name of the new security group. You can add existing security groups by choosing **Choose existing** and selecting security groups to add.
Availability Zone	RDS chooses the Availability Zone of the EC2 instance for one DB instance in the Multi-AZ DB cluster deployment. RDS randomly chooses a different Availability Zone for both of the other DB instances. The writer DB instance is created in the same Availability Zone as the EC2 instance. There is the possibility of cross Availability Zone costs if a failover occurs and the writer DB instance is in a different Availability Zone.

For more information about these settings, see Settings for creating Multi-AZ DB clusters.

If you change these settings after the DB cluster is created, the changes might affect the connection between the EC2 instance and the DB cluster.

Configure the network manually

To connect to your DB cluster from resources other than EC2 instances in the same VPC, configure the network connections manually. If you use the AWS Management Console to create your Multi-AZ DB cluster, you can have Amazon RDS automatically create a VPC for you. Or you can use an existing VPC or create a new VPC for your Multi-AZ DB cluster. Your VPC must have at least one subnet in each of at least three Availability Zones for you to use it with a Multi-AZ DB cluster. For information on VPCs, see Amazon VPC and Amazon RDS.

If you don't have a default VPC or you haven't created a VPC, and you don't plan to use the console, do the following:

- Create a VPC with at least one subnet in each of at least three of the Availability Zones in the AWS Region where you want to deploy your DB cluster. For more information, see Working with a DB instance in a VPC.
- Specify a VPC security group that authorizes connections to your DB cluster. For more information, see Provide access to your DB instance in your VPC by creating a security group and Controlling access with security groups.
- Specify an RDS DB subnet group that defines at least three subnets in the VPC that can be used by the Multi-AZ DB cluster. For more information, see Working with DB subnet groups.

For information about limitations that apply to Multi-AZ DB clusters, see Limitations of Multi-AZ DB clusters for Amazon RDS.

If you want to connect to a resource that isn't in the same VPC as the Multi-AZ DB cluster, see the appropriate scenarios in Scenarios for accessing a DB instance in a VPC.

Additional prerequisites

Before you create your Multi-AZ DB cluster, consider the following additional prerequisites:

- To tailor the configuration parameters for your DB cluster, specify a DB cluster parameter group with the required parameter settings. For information about creating or modifying a DB cluster parameter group, see Parameter groups for Multi-AZ DB clusters.
- Determine the TCP/IP port number to specify for your DB cluster. The firewalls at some companies block connections to the default ports. If your company firewall blocks the default port, choose another port for your DB cluster. All DB instances in a DB cluster use the same port.

- If the major engine version for your database reached the RDS end of standard support date, you must use the Extended Support CLI option or the RDS API parameter. For more information, see RDS Extended Support in Settings for creating Multi-AZ DB clusters.

Creating a DB cluster

You can create a Multi-AZ DB cluster using the AWS Management Console, the AWS CLI, or the RDS API.

Console

You can create a Multi-AZ DB cluster by choosing **Multi-AZ DB cluster** in the **Availability and durability** section.

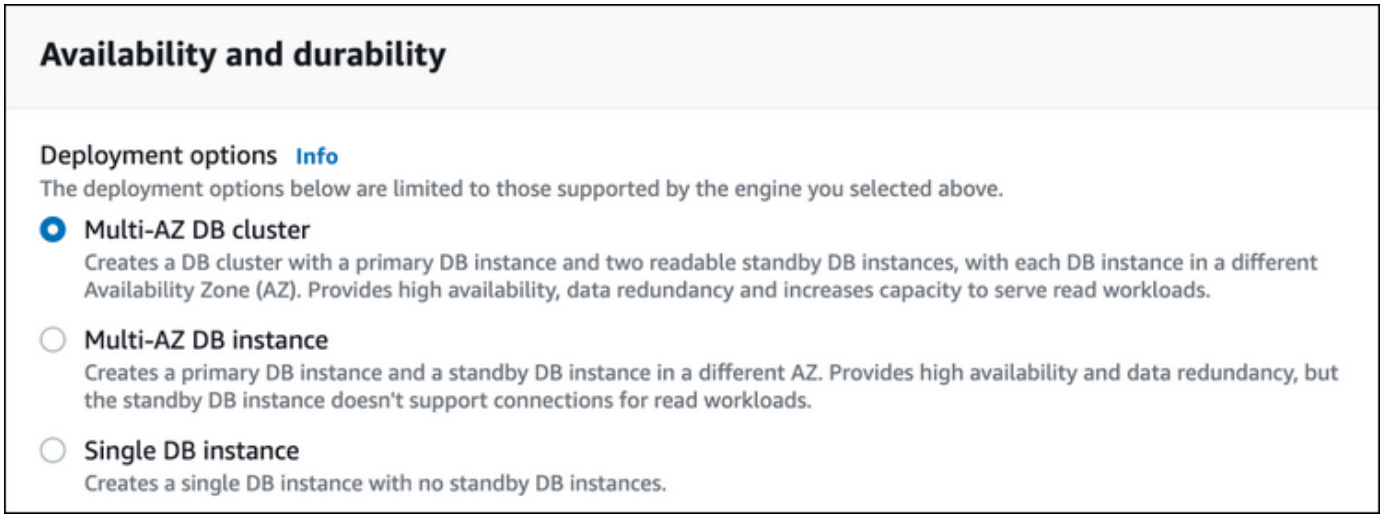
To create a Multi-AZ DB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you want to create the DB cluster.

   For information about the AWS Regions that support Multi-AZ DB clusters, see Limitations of Multi-AZ DB clusters for Amazon RDS.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.

   To create a Multi-AZ DB cluster, make sure that **Standard Create** is selected and **Easy Create** isn't.
5. In **Engine type**, choose **MySQL** or **PostgreSQL**.
6. For **Version**, choose the DB engine version.

   For information about the DB engine versions that support Multi-AZ DB clusters, see Limitations of Multi-AZ DB clusters for Amazon RDS.
7. In **Templates**, choose the appropriate template for your deployment.
8. In **Availability and durability**, choose **Multi-AZ DB cluster**.



9. In **DB cluster identifier**, enter the identifier for your DB cluster.
10. In **Master username**, enter your master user name, or keep the default setting.
11. Enter your master password:

    a. In the **Settings** section, open **Credential Settings**.

    b. If you want to specify a password, clear the **Auto generate a password** box if it is selected.

    c. (Optional) Change the **Master username** value.

    d. Enter the same password in **Master password** and **Confirm password**.
12. For **DB instance class**, choose a DB instance class. For a list of supported DB instance classes, see the section called “Instance class availability for Multi-AZ DB clusters”.
13. (Optional) Set up a connection to a compute resource for this DB cluster.

    You can configure connectivity between an Amazon EC2 instance and the new DB cluster during DB cluster creation. For more information, see Configure automatic network connectivity with an EC2 instance.
14. In the **Connectivity** section under **VPC security group (firewall)**, if you select **Create new**, a VPC security group is created with an inbound rule that allows your local computer's IP address to access the database.
15. For the remaining sections, specify your DB cluster settings. For information about each setting, see Settings for creating Multi-AZ DB clusters.
16. Choose **Create database**.

    If you chose to use an automatically generated password, the **View credential details** button appears on the **Databases** page.

To view the master user name and password for the DB cluster, choose **View credential details**.

To connect to the DB cluster as the master user, use the user name and password that appear.

> **Important**
>
> You can't view the master user password again.

17. For **Databases**, choose the name of the new DB cluster.

On the RDS console, the details for the new DB cluster appear. The DB cluster has a status of **Creating** until the DB cluster is created and ready for use. When the state changes to **Available**, you can connect to the DB cluster. Depending on the DB cluster class and storage allocated, it can take several minutes for the new DB cluster to be available.

AWS CLI

Before you create a Multi-AZ DB cluster using the AWS CLI, make sure to fulfill the required prerequisites. These include creating a VPC and an RDS DB subnet group. For more information, see DB cluster prerequisites.

To create a Multi-AZ DB cluster by using the AWS CLI, call the create-db-cluster command. Specify the `--db-cluster-identifier`. For the `--engine` option, specify either `mysql` or `postgres`.

For information about each option, see Settings for creating Multi-AZ DB clusters.

For information about the AWS Regions, DB engines, and DB engine versions that support Multi-AZ DB clusters, see Limitations of Multi-AZ DB clusters for Amazon RDS.

The `create-db-cluster` command creates the writer DB instance for your DB cluster, and two reader DB instances. Each DB instance is in a different Availability Zone.

For example, the following command creates a MySQL 8.0 Multi-AZ DB cluster named `mysql-multi-az-db-cluster`.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
```

```
    --db-cluster-identifier mysql-multi-az-db-cluster \
    --engine mysql \
    --engine-version 8.0.32  \
    --master-username admin \
    --manage-master-user-password  \
    --port 3306 \
    --backup-retention-period 1  \
    --db-subnet-group-name default \
    --allocated-storage 4000 \
    --storage-type io1 \
    --iops 10000 \
    --db-cluster-instance-class db.m5d.xlarge
```

For Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier mysql-multi-az-db-cluster ^
    --engine mysql ^
    --engine-version 8.0.32 ^
    --manage-master-user-password ^
    --master-username admin ^
    --port 3306 ^
    --backup-retention-period 1 ^
    --db-subnet-group-name default ^
    --allocated-storage 4000 ^
    --storage-type io1 ^
    --iops 10000 ^
    --db-cluster-instance-class db.m5d.xlarge
```

The following command creates a PostgreSQL 13.4 Multi-AZ DB cluster named `postgresql-multi-az-db-cluster`.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier postgresql-multi-az-db-cluster \
    --engine postgres \
    --engine-version 13.4 \
    --manage-master-user-password \
    --master-username postgres \
    --port 5432 \
```

```
    --backup-retention-period 1  \
    --db-subnet-group-name default \
    --allocated-storage 4000 \
    --storage-type io1 \
    --iops 10000 \
    --db-cluster-instance-class db.m5d.xlarge
```

For Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier postgresql-multi-az-db-cluster ^
    --engine postgres ^
    --engine-version 13.4 ^
    --manage-master-user-password ^
    --master-username postgres ^
    --port 5432 ^
    --backup-retention-period 1 ^
    --db-subnet-group-name default ^
    --allocated-storage 4000 ^
    --storage-type io1 ^
    --iops 10000 ^
    --db-cluster-instance-class db.m5d.xlarge
```

RDS API

Before you can create a Multi-AZ DB cluster using the RDS API, make sure to fulfill the required prerequisites, such as creating a VPC and an RDS DB subnet group. For more information, see DB cluster prerequisites.

To create a Multi-AZ DB cluster by using the RDS API, call the CreateDBCluster operation. Specify the `DBClusterIdentifier`. For the `Engine` parameter, specify either `mysql` or `postgresql`.

For information about each option, see Settings for creating Multi-AZ DB clusters.

The `CreateDBCluster` operation creates the writer DB instance for your DB cluster, and two reader DB instances. Each DB instance is in a different Availability Zone.

Settings for creating Multi-AZ DB clusters

For details about settings that you choose when you create a Multi-AZ DB cluster, see the following table. For more information about the AWS CLI options, see create-db-cluster. For more information about the RDS API parameters, see CreateDBCluster.

Console setting	Setting description	CLI option and RDS API parameter

Allocated storage	The amount of storage to allocate for each DB instance in your DB cluster (in gibibyte). For more information, see Amazon RDS DB instance storage.	**CLI option:**
		`--allocated-storage`
		API parameter:
		`AllocatedStorage`

Auto minor version upgrade	**Enable auto minor version upgrade** to have your DB cluster receive preferred minor DB engine version upgrades automatically when they become available. Amazon RDS performs automatic minor version upgrades in the maintenance window.	**CLI option:**
		`--auto-minor-version-upgrade`
		`--no-auto-minor-version-upgrade`
		API parameter:
		`AutoMinorVersionUpgrade`

Backup retention period	The number of days that you want automatic backups of your DB cluster to be retained. For a Multi-AZ DB cluster, this value must be set to **1** or greater.
	For more information, see Introduction to backups.
		CLI option:
		`--backup-retention-period`
		API parameter:
		`BackupRetentionPeriod`

Backup window	The time period during which Amazon RDS automatically takes a backup of your DB cluster. Unless you have a specific time that you want to have your database backed up, use the default of **No preference**.	**CLI option:**
		`--preferred-backup-window`
		API parameter:
		`PreferredBackupWindow`

Console setting	Setting description	CLI option and RDS API parameter
	For more information, see Introduction to backups.	

Certificate authority	The certificate authority (CA) for the server certificate used by the DB cluster.
	
	For more information, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
		CLI option:
		
		`--ca-certificate-identifier`
		
		RDS API parameter:
		
		`CACertificateIdentifier`

Copy tags to snapshots	This option copies any DB cluster tags to a DB snapshot when you create a snapshot.
	
	For more information, see Tagging Amazon RDS resources.
		CLI option:
		
		`-copy-tags-to-snapshot`
		
		`-no-copy-tags-to-snapshot`
		
		RDS API parameter:
		
		`CopyTagsToSnapshot`

Database authentication	The database authentication option that you want to use.
	
	Choose **Password authentication** to authenticate database users with database passwords only.
	
	Choose **Password and IAM DB authentication** to authenticate database users with database passwords and user credentials through users and roles. For more information, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.
		CLI option:
		
		`--enable-iam-database-authentication`
		
		`--no-enable-iam-database-authentication`
		
		RDS API parameter:
		
		`EnableIAMDatabaseAuthentication`

Console setting	Setting description	CLI option and RDS API parameter

Database port	The port that you want to access the DB cluster through. The default port is shown.
	
	The port can't be changed after the DB cluster is created.
	
	The firewalls at some companies block connections to the default ports. If your company firewall blocks the default port, enter another port for your DB cluster.
		CLI option:
		
		`--port`
		
		RDS API parameter:
		
		`Port`

DB cluster identifier	The name for your DB cluster. Name your DB clusters in the same way that you name your on-premises servers. Your DB cluster identifier can contain up to 63 alphanumeric characters, and must be unique for your account in the AWS Region you chose.	**CLI option:**
		
		`--db-cluster-identifier`
		
		RDS API parameter:
		
		`DBClusterIdentifier`

Console setting	Setting description	CLI option and RDS API parameter

DB instance class	The compute and memory capacity of each DB instance in the Multi-AZ DB cluster, for example `db.m5d.xlarge` .
	
	If possible, choose a DB instance class large enough that a typical query working set can be held in memory. When working sets are held in memory the system can avoid writing to disk, which improves performance.
	
	For a list of supported DB instance classes, see the section called "Instance class availability for Multi-AZ DB clusters".
		CLI option:
		
		`--db-cluster-instance-class`
		
		RDS API parameter:
		
		`DBClusterInstanceClass`

DB cluster parameter group	The DB cluster parameter group that you want associated with the DB cluster.
	
	For more information, see Parameter groups for Multi-AZ DB clusters.
		CLI option:
		
		`--db-cluster-parameter-group-name`
		
		RDS API parameter:
		
		`DBClusterParameterGroupName`

DB engine version	The version of database engine that you want to use.	**CLI option:**
		
		`--engine-version`
		
		RDS API parameter:
		
		`EngineVersion`

Console setting	Setting description	CLI option and RDS API parameter
DB cluster parameter group	The DB instance parameter group to associate with the DB cluster. For more information, see Parameter groups for Multi-AZ DB clusters.	**CLI option:** `--db-cluster-parameter-group-name` **RDS API parameter:** **`DBClusterParameterGroupName`**

DB subnet group	The DB subnet group you want to use for the DB cluster.
	Select **Choose existing** to use an existing DB subnet group. Then choose the required subnet group from the **Existing DB subnet groups** dropdown list.
	
	Choose **Automatic setup** to let RDS select a compatible DB subnet group. If none exist, RDS creates a new subnet group for your cluster.
	
	For more information, see Working with DB subnet groups.
		CLI option:
		
		`--db-subnet-group-name`
		
		RDS API parameter:
		
		`DBSubnetGroupName`

Deletion protection	**Enable deletion protection** to prevent your DB cluster from being deleted. If you create a production DB cluster with the console, deletion protection is turned on by default.
	
	For more information, see Deleting a DB instance.
		CLI option:
		
		`--deletion-protection`
		
		`--no-deletion-protection`
		
		RDS API parameter:
		
		`DeletionProtection`

Console setting	Setting description	CLI option and RDS API parameter

Encryption	**Enable Encryption** to turn on encryption at rest for this DB cluster.
	
	Encryption is turned on by default for Multi-AZ DB clusters.
	
	For more information, see Encrypting Amazon RDS resources.
		CLI options:
		
		`--kms-key-id`
		
		`--storage-encrypted`
		
		`--no-storage-encrypted`
		
		RDS API parameters:
		
		`KmsKeyId`
		
		`StorageEncrypted`

Enhanced Monitoring	**Enable enhanced monitoring** to turn on metrics gathering in real time for the operating system that your DB cluster runs on.
	
	For more information, see Monitoring OS metrics with Enhanced Monitoring.
		CLI options:
		
		`--monitoring-interval`
		
		`--monitoring-role-arn`
		
		RDS API parameters:
		
		`MonitoringInterval`
		
		`MonitoringRoleArn`

Console setting	Setting description	CLI option and RDS API parameter

Initial database name	The name for the database on your DB cluster. If you don't provide a name, Amazon RDS doesn't create a database on the DB cluster for MySQL. However, it does create a database on the DB cluster for PostgreSQL. The name can't be a word reserved by the database engine. It has other constraints depending on the DB engine.
	
	MySQL:
	
	• It must contain 1–64 alphanumeric characters.
	
	PostgreSQL:
	
	• It must contain 1–63 alphanumeric characters.
	• It must begin with a letter or an underscore. Subsequent characters can be letters, underscores, or digits (0-9).
	• The initial database name is `postgres`.
		CLI option:
		
		`--database-name`
		
		RDS API parameter:
		
		`DatabaseName`

Console setting	Setting description	CLI option and RDS API parameter

Log exports	The types of database log files to publish to Amazon CloudWatch Logs.
	
	For more information, see Publishing database logs to Amazon CloudWatch Logs.
		CLI option:
		
		`-enable-cloudwatch-logs-exports`
		
		RDS API parameter:
		
		`EnableCloudwatchLogsExports`

Maintenance window	The 30-minute window in which pending modifications to your DB cluster are applied. If the time period doesn't matter, choose **No preference**.
	
	For more information, see Amazon RDS maintenance window.
		CLI option:
		
		`--preferred-maintenance-window`
		
		RDS API parameter:
		
		`PreferredMaintenanceWindow`

Manage master credentials in AWS Secrets Manager	Select **Manage master credentials in AWS Secrets Manager** to manage the master user password in a secret in Secrets Manager.
	
	Optionally, choose a KMS key to use to protect the secret. Choose from the KMS keys in your account, or enter the key from a different account.
	
	For more information, see Password management with Amazon RDS and AWS Secrets Manager.
		CLI option:
		
		`--manage-master-user-password | --no-manage-master-user-password`
		
		`--master-user-secret-kms-key-id`
		
		RDS API parameter:
		
		`ManageMasterUserPassword`
		
		`MasterUserSecretKmsKeyId`

Console setting	Setting description	CLI option and RDS API parameter

Master password	The password for your master user account.	**CLI option:**
		`--master-user-password`
		RDS API parameter:
		`MasterUserPassword`

Master username	The name that you use as the master user name to log on to your DB cluster with all database privileges.
	• It can contain 1–16 alphanumeric characters and underscores.
	• Its first character must be a letter.
	• It can't be a word reserved by the database engine.
	You can't change the master user name after the Multi-AZ DB cluster is created.
	For more information on privileges granted to the master user, see Master user account privileges.
		CLI option:
		`--master-username`
		RDS API parameter:
		`MasterUsername`

Console setting	Setting description	CLI option and RDS API parameter

Performance Insights	**Enable Performance Insights** to monitor your DB cluster load so that you can analyze and troubleshoot your database performance.
	
	Choose a retention period to determine how much Performance Insights data history to keep. The retention setting is **Default (7 days)**. To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see Pricing and data retention for Performance Insights.
	
	Choose a KMS key to use to protect the key used to encrypt this database volume. Choose from the KMS keys in your account, or enter the key from a different account.
	
	For more information, see Monitoring DB load with Performance Insights on Amazon RDS.
		CLI options:
		
		`--enable-performance-insights`
		
		`--no-enable-performance-insights`
		
		`--performance-insights-retention-period`
		
		`--performance-insights-kms-key-id`
		
		RDS API parameters:
		
		`EnablePerformanceInsights`
		
		`PerformanceInsightsRetentionPeriod`
		
		`PerformanceInsightsKMSKeyId`

Provisioned IOPS	The amount of Provisioned IOPS (input/output operations per second) to be initially allocated for the DB cluster.	**CLI option:**
		
		`--iops`
		
		RDS API parameter:
		
		`Iops`

Console setting	Setting description	CLI option and RDS API parameter

Public access	**Publicly accessible** to give the DB cluster a public IP address, meaning that it's accessible outside the VPC. To be publicly accessible, the DB cluster also has to be in a public subnet in the VPC.
	
	Not publicly accessible to make the DB cluster accessible only from inside the VPC.
	
	For more information, see Hiding a DB instance in a VPC from the internet.
	
	To connect to a DB cluster from outside of its VPC, the DB cluster must be publicly accessible. Also, access must be granted using the inbound rules of the DB cluster's security group, and other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance.
	
	If your DB cluster isn't publicly accessible, you can use an AWS Site-to-Site VPN connection or an Direct Connect connection to access it from a private network. For more information, see Internetwork traffic privacy.
		CLI option:
		
		`--publicly-accessible`
		
		`--no-publicly-accessible`
		
		RDS API parameter:
		
		`PubliclyAccessible`

Console setting	Setting description	CLI option and RDS API parameter

RDS Extended Support	Select **Enable RDS Extended Support** to allow supported major engine versions to continue running past the RDS end of standard support date.
	
	When you create a DB cluster, Amazon RDS defaults to RDS Extended Support. To prevent the creation of a new DB cluster after the RDS end of standard support date and to avoid charges for RDS Extended Support, disable this setting. Your existing DB clusters won't incur charges until the RDS Extended Support pricing start date.
	
	For more information, see Amazon RDS Extended Support with Amazon RDS.
		CLI option:
		
		`--engine-lifecycle-support`
		
		RDS API parameter:
		
		`EngineLifecycleSupport`

Storage throughput	The storage throughput value for the DB cluster. This setting is visible only if you choose General Purpose SSD (gp3) for the storage type. This setting is not configurable and is automatically set based on the IOPS that you specify. For more information, see gp3 storage (recommended).	This value is automatically calculated and doesn't have a CLI option.

Console setting	Setting description	CLI option and RDS API parameter
RDS Proxy	Choose **Create an RDS Proxy** to create a proxy for your DB cluster. Amazon RDS automatically creates an IAM role and a Secrets Manager secret for the proxy.	Not available when creating a DB cluster.

Storage type	The storage type for your DB cluster.
	
	Only General Purpose SSD (gp3), Provisioned IOPS (io1), and Provisioned IOPS SSD (io2) stor age are supported.
	
	For more information, see Amazon RDS storage types.
		CLI option:
		
		`--storage-type`
		
		RDS API parameter:
		
		`StorageType`

Virtual Private Cloud (VPC)	A VPC based on the Amazon VPC service to associate with this DB cluster. For more information, see Amazon VPC and Amazon RDS.	For the CLI and API, you specify the VPC security group IDs.

VPC security group (firewall)	The security groups to associate with the DB cluster.
	
	For more information, see Overview of VPC security groups.
		CLI option:
		
		`--vpc-security-group-ids`
		
		RDS API parameter:
		
		`VpcSecurityGroupIds`

Settings that don't apply when creating Multi-AZ DB clusters

The following settings in the AWS CLI command `create-db-cluster` and the RDS API operation `CreateDBCluster` don't apply to Multi-AZ DB clusters.

You also can't specify these settings for Multi-AZ DB clusters in the console.

AWS CLI setting	RDS API setting
`--availability-zones`	`AvailabilityZones`
`--backtrack-window`	`BacktrackWindow`
`--character-set-name`	`CharacterSetName`
`--domain`	`Domain`
`--domain-iam-role-name`	`DomainIAMRoleName`
`--enable-global-write-forwarding | --no-enable-global-write-fo rwarding`	`EnableGlobalWriteForwarding`
`--enable-http-endpoint | --no-enable-http-endpoint`	`EnableHttpEndpoint`
`--global-cluster-identifier`	`GlobalClusterIdentifier`
`--option-group-name`	`OptionGroupName`
`--pre-signed-url`	`PreSignedUrl`
`--replication-source-identifier`	`ReplicationSourceIdentifier`
`--scaling-configuration`	`ScalingConfiguration`

Connecting to a Multi-AZ DB cluster for Amazon RDS

A Multi-AZ DB cluster has three DB instances instead of a single DB instance. Each connection is handled by a specific DB instance. When you connect to a Multi-AZ DB cluster, the hostname and port that you specify point to a fully qualified domain name called an *endpoint*. The Multi-AZ DB cluster uses the endpoint mechanism to abstract these connections so that you don't need to specify exactly which DB instance in the DB cluster to connect to. Thus, you don't have to hardcode all the hostnames or write your own logic for rerouting connections when some DB instances aren't available.

The writer endpoint connects to the writer DB instance of the DB cluster, which supports both read and write operations. The reader endpoint connects to either of the two reader DB instances, which support only read operations.

Using endpoints, you can map each connection to the appropriate DB instance or group of DB instances based on your use case. For example, to perform DDL and DML statements, you can connect to whichever DB instance is the writer DB instance. To perform queries, you can connect to the reader endpoint, with the Multi-AZ DB cluster automatically managing connections among the reader DB instances. For diagnosis or tuning, you can connect to a specific DB instance endpoint to examine details about a specific DB instance.

For information about connecting to a DB instance, see Connecting to an Amazon RDS DB instance.

For more information about connecting to Multi-AZ DB clusters, see the following topics.

Topics

- Cluster endpoints
- Reader endpoints
- Instance endpoints
- High availability connections
- Connecting to Multi-AZ DB clusters with the AWS drivers for Amazon RDS

Types of Multi-AZ DB cluster endpoints

An endpoint is represented by a unique identifier that contains a host address. The following types of endpoints are available from a Multi-AZ DB cluster:

Cluster endpoint

A *cluster endpoint* (or *writer endpoint*) for a Multi-AZ DB cluster connects to the current writer DB instance for that DB cluster. This endpoint is the only one that can perform write operations such as DDL and DML statements. This endpoint can also perform read operations.

Each Multi-AZ DB cluster has one cluster endpoint and one writer DB instance.

You use the cluster endpoint for all write operations on the DB cluster, including inserts, updates, deletes, and DDL changes. You can also use the cluster endpoint for read operations, such as queries.

If the current writer DB instance of a DB cluster fails, the Multi-AZ DB cluster automatically fails over to a new writer DB instance. During a failover, the DB cluster continues to serve connection requests to the cluster endpoint from the new writer DB instance, with minimal interruption of service.

The following example illustrates a cluster endpoint for a Multi-AZ DB cluster.

```
mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com
```

For more information about connecting to cluster endpoints, see Cluster endpoints.

Reader endpoint

A *reader endpoint* for a Multi-AZ DB cluster provides support for read-only connections to the DB cluster. Use the reader endpoint for read operations, such as `SELECT` queries. By processing those statements on the reader DB instances, this endpoint reduces the overhead on the writer DB instance. It also helps the cluster to scale the capacity to handle simultaneous `SELECT` queries. Each Multi-AZ DB cluster has one reader endpoint.

The reader endpoint sends each connection request to one of the reader DB instances. When you use the reader endpoint for a session, you can only perform read-only statements such as `SELECT` in that session.

The following example illustrates a reader endpoint for a Multi-AZ DB cluster. The read-only intent of a reader endpoint is denoted by the `-ro` within the cluster endpoint name.

```
mydbcluster.cluster-ro-123456789012.us-east-1.rds.amazonaws.com
```

For more information about connecting to reader endpoints, see Reader endpoints.

Instance endpoint

An *instance endpoint* connects to a specific DB instance within a Multi-AZ DB cluster. Each DB instance in a DB cluster has its own unique instance endpoint. So there is one instance endpoint for the current writer DB instance of the DB cluster, and there is one instance endpoint for each of the reader DB instances in the DB cluster.

The instance endpoint provides direct control over connections to the DB cluster. This control can help you address scenarios where using the cluster endpoint or reader endpoint might not be appropriate. For example, your client application might require more fine-grained load balancing based on workload type. In this case, you can configure multiple clients to connect to different reader DB instances in a DB cluster to distribute read workloads.

The following example illustrates an instance endpoint for a DB instance in a Multi-AZ DB cluster.

```
mydbinstance.123456789012.us-east-1.rds.amazonaws.com
```

For more information about connecting to instance endpoints, see Instance endpoints.

Viewing endpoints

Use the console, AWS CLI, or Amazon RDS API to view the cluster, reader, and instance endpoints.

Console

In the AWS Management Console, you see the cluster endpoint and the reader endpoint on the details page for each Multi-AZ DB cluster. You see the instance endpoint in the details page for each DB instance.

AWS CLI

With the AWS CLI, you see the writer and reader endpoints in the output of the describe-db-clusters command. For example, the following command shows the endpoint attributes for all clusters in your current AWS Region.

```
aws rds describe-db-cluster-endpoints
```

Amazon RDS API

With the Amazon RDS API, you retrieve the endpoints by calling the DescribeDBClusterEndpoints action. The output also shows Amazon Aurora DB cluster endpoints, if any exist.

Cluster endpoints

Each Multi-AZ DB cluster has a single built-in cluster endpoint, whose name and other attributes are managed by Amazon RDS. You can't create, delete, or modify this kind of endpoint.

You use the cluster endpoint when you administer your DB cluster, perform extract, transform, load (ETL) operations, or develop and test applications. The cluster endpoint connects to the writer DB instance of the cluster. The writer DB instance is the only DB instance where you can create tables and indexes, run `INSERT` statements, and perform other DDL and DML operations.

The physical IP address pointed to by the cluster endpoint changes when the failover mechanism promotes a new DB instance to be the writer DB instance for the cluster. If you use any form of connection pooling or other multiplexing, be prepared to flush or reduce the time-to-live for any cached DNS information. Doing so ensures that you don't try to establish a read/write connection to a DB instance that became unavailable or is now read-only after a failover.

Reader endpoints

You use the reader endpoint for read-only connections to your Multi-AZ DB cluster. This endpoint helps your DB cluster handle a query-intensive workload. The reader endpoint is the endpoint that you supply to applications that do reporting or other read-only operations on the cluster. The reader endpoint sends connections to available reader DB instances in a Multi-AZ DB cluster.

Each Multi-AZ cluster has a single built-in reader endpoint, whose name and other attributes are managed by Amazon RDS. You can't create, delete, or modify this kind of endpoint.

Instance endpoints

Each DB instance in a Multi-AZ DB cluster has its own built-in instance endpoint, whose name and other attributes are managed by Amazon RDS. You can't create, delete, or modify this kind of endpoint. With a Multi-AZ DB cluster, you typically use the writer and reader endpoints more often than the instance endpoints.

In day-to-day operations, the main way that you use instance endpoints is to diagnose capacity or performance issues that affect one specific DB instance in a Multi-AZ DB cluster. While connected to a specific DB instance, you can examine its status variables, metrics, and so on. Doing this can help you determine what's happening for that DB instance that's different from what's happening for other DB instances in the cluster.

High availability connections

For Multi-AZ DB clusters where high availability is important, use the writer endpoint for read/write or general-purpose connections and the reader endpoint for read-only connections. The writer and reader endpoints manage DB instance failover better than instance endpoints do. Unlike the instance endpoints, the writer and reader endpoints automatically change which DB instance they connect to if a DB instance in your cluster becomes unavailable.

If the writer DB instance of a DB cluster fails, Amazon RDS automatically fails over to a new writer DB instance. It does so by promoting a reader DB instance to a new writer DB instance. If a failover occurs, you can use the writer endpoint to reconnect to the newly promoted writer DB instance. Or you can use the reader endpoint to reconnect to one of the reader DB instances in the DB cluster. During a failover, the reader endpoint might direct connections to the new writer DB instance of a DB cluster for a short time after a reader DB instance is promoted to the new writer DB instance. If you design your own application logic to manage instance endpoint connections, you can manually or programmatically discover the resulting set of available DB instances in the DB cluster.

Connecting to Multi-AZ DB clusters with the AWS drivers for Amazon RDS

The AWS suite of drivers has been designed to provide support for faster switchover and failover times, and authentication with AWS Secrets Manager, AWS Identity and Access Management (IAM), and Federated Identity. The AWS drivers rely on monitoring DB cluster status and being aware of the cluster topology to determine the new writer. This approach reduces switchover and failover times to single-digit seconds, compared to tens of seconds for open-source drivers.

As new service features are introduced, the goal of the AWS suite of drivers is to have built-in support for these service features.

Connecting to Multi-AZ DB clusters with the Amazon Web Services (AWS) JDBC Driver

The Amazon Web Services (AWS) JDBC Driver is designed as an advanced JDBC wrapper to help applications take advantage of the features of clustered databases. This wrapper is complementary to and extends the functionality of an existing JDBC driver. The driver is drop-in compatible with the following community drivers:

- MySQL Connector/J
- MariaDB Connector/J
- pgJDBC

To install the AWS JDBC Driver, append the AWS JDBC Driver .jar file (located in the application CLASSPATH), and keep references to the respective community driver. Update the respective connection URL prefix as follows:

- `jdbc:mysql://` to `jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://` to `jdbc:aws-wrapper:mariadb://`
- `jdbc:postgresql://` to `jdbc:aws-wrapper:postgresql://`

For more information about the AWS JDBC Driver and complete instructions for using it, see the Amazon Web Services (AWS) JDBC Driver GitHub repository.

Connecting to Multi-AZ DB clusters with the Amazon Web Services (AWS) Python Driver

The Amazon Web Services (AWS) Python Driver is designed as an advanced Python wrapper. This wrapper is complementary to and extends the functionality of the open-source Psycopg driver. The AWS Python Driver supports Python versions 3.8 and higher. You can install the `aws-advanced-python-wrapper` package using the `pip` command, along with the `psycopg` open-source packages.

For more information about the AWS Python Driver and complete instructions for using it, see the Amazon Web Services (AWS) Python Driver GitHub repository.

Automatically connecting an AWS compute resource and a Multi-AZ DB cluster for Amazon RDS

You can automatically connect a Multi-AZ DB cluster and AWS compute resources such as Amazon Elastic Compute Cloud (Amazon EC2) instances and AWS Lambda functions.

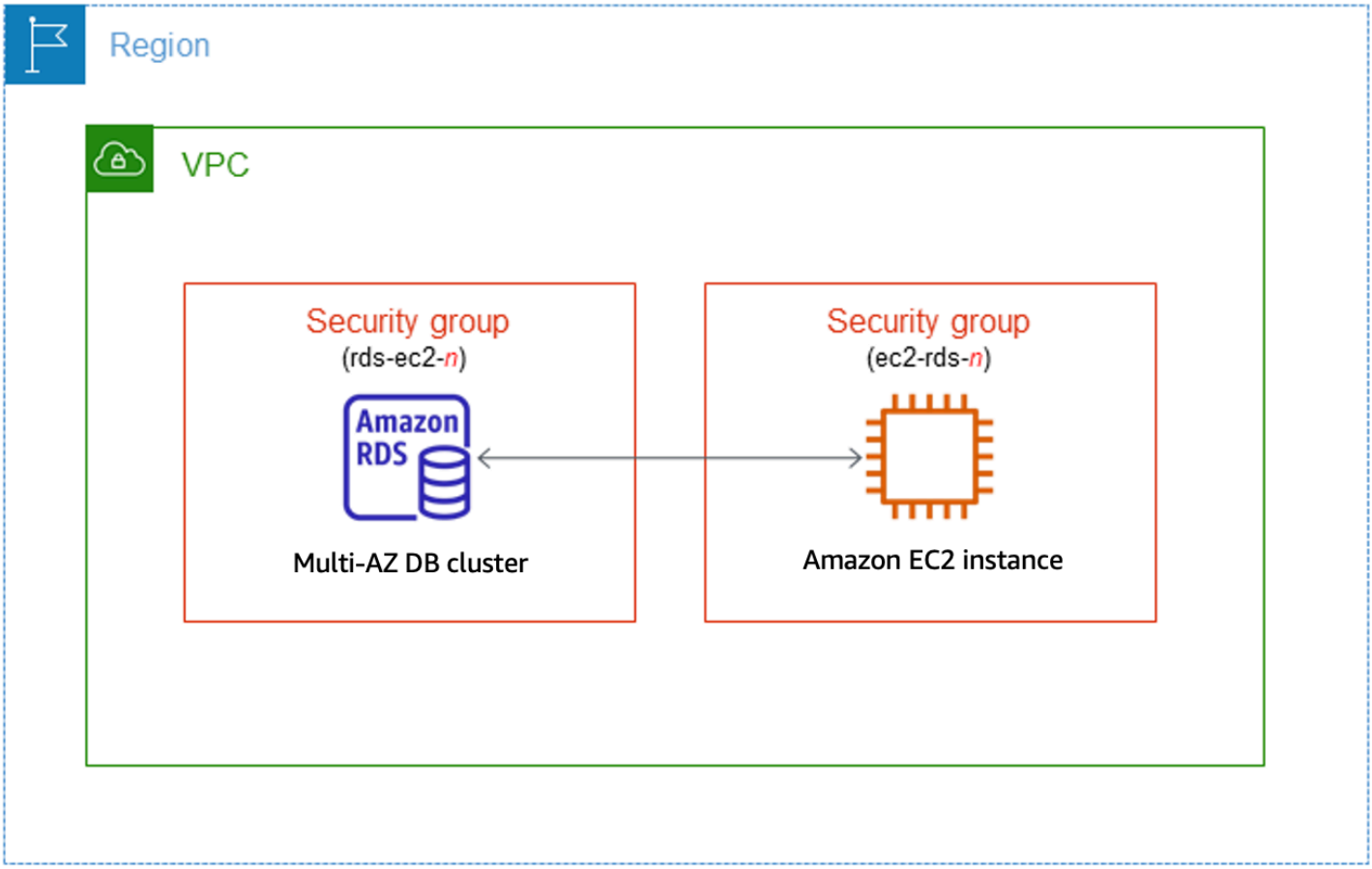
The following topics provide detailed instructions for configuring network settings, security groups, and connection parameters to establish reliable connections to Amazon RDS DB instances within a Multi-AZ DB cluster deployment. They focus on optimizing network connectivity and performance for applications interacting with Multi-AZ DB cluster, ensuring secure and efficient data operations.

Topics

- Automatically connecting an EC2 instance and a Multi-AZ DB cluster
- Automatically connecting a Lambda function and a Multi-AZ DB cluster

Automatically connecting an EC2 instance and a Multi-AZ DB cluster

You can use the Amazon RDS console to simplify setting up a connection between an Amazon Elastic Compute Cloud (Amazon EC2) instance and a Multi-AZ DB cluster. Often, your Multi-AZ DB cluster is in a private subnet and your EC2 instance is in a public subnet within a VPC. You can use a SQL client on your EC2 instance to connect to your Multi-AZ DB cluster. The EC2 instance can also run web servers or applications that access your private Multi-AZ DB cluster.



If you want to connect to an EC2 instance that isn't in the same VPC as the Multi-AZ DB cluster, see the scenarios in the section called "Scenarios for accessing a DB instance in a VPC".

Topics

- Overview of automatic connectivity with an EC2 instance
- Connecting an EC2 instance and a Multi-AZ DB cluster automatically
- Viewing connected compute resources

Overview of automatic connectivity with an EC2 instance

When you set up a connection between an EC2 instance and a Multi-AZ DB cluster automatically, Amazon RDS configures the VPC security group for your EC2 instance and for your DB cluster.

The following are requirements for connecting an EC2 instance with a Multi-AZ DB cluster:

- The EC2 instance must exist in the same VPC as the Multi-AZ DB cluster.

  If no EC2 instances exist in the same VPC, the console provides a link to create one.

- The user who is setting up connectivity must have permissions to perform the following EC2 operations:
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

When you set up a connection to an EC2 instance, Amazon RDS acts according to the current configuration of the security groups associated with the Multi-AZ DB cluster and EC2 instance, as described in the following table.

Current RDS security group configuration	Current EC2 security group configuration	RDS action
There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-ec2-`*n* (where *n* is a number). A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	There are one or more security groups associated with the EC2 instance with a name that matches the pattern `rds-ec2-`*n* (where *n* is a number). A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the Multi-AZ DB cluster as the source.	Amazon RDS takes no action. A connection was already configured automatically between the EC2 instance and Multi-AZ DB cluster. Because a connection already exists between the EC2 instance and the RDS database, the security groups aren't modified.
Either of the following conditions apply:	Either of the following conditions apply:	RDS action: create new security groups

Current RDS security group configuration	Current EC2 security group configuration	RDS action
• There is no security group associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-ec2-`*n*. • There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-ec2-`*n*. However, none of these security groups can be used for the connection with the EC2 instance. A security group can't be used if it doesn't have one inbound rule with the VPC security group of the EC2 instance as the source. A security group also can't be used if it has been modified. Examples of modifications include adding a rule or changing the port of an existing rule.	• There is no security group associated with the EC2 instance with a name that matches the pattern `ec2-rds-`*n*. • There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-rds-`*n*. However, none of these security groups can be used for the connection with the Multi-AZ DB cluster. A security group can't be used if it doesn't have one outbound rule with the VPC security group of the Multi-AZ DB cluster as the source. A security group also can't be used if it has been modified.	

Current RDS security group configuration	Current EC2 security group configuration	RDS action
There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-ec2-`*n*. A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	There are one or more security groups associated with the EC2 instance with a name that matches the pattern `ec2-rds-`*n*. However, none of these security groups can be used for the connection with the Multi-AZ DB cluster. A security group can't be used if it doesn't have one outbound rule with the VPC security group of the Multi-AZ DB cluster as the source. A security group also can't be used if it has been modified.	RDS action: create new security groups
There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-ec2-`*n*. A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the EC2 instance as the source.	A valid EC2 security group for the connection exists, but it is not associated with the EC2 instance. This security group has a name that matches the pattern `rds-ec2-`*n*. It hasn't been modified. It has only one outbound rule with the VPC security group of the Multi-AZ DB cluster as the source.	RDS action: associate EC2 security group

Current RDS security group configuration	Current EC2 security group configuration	RDS action
Either of the following conditions apply: • There is no security group associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-ec2-`*n*. • There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-ec2-`*n*. However, none of these security groups can be used for the connection with the EC2 instance. A security group can't be used if it doesn't have one inbound rule with the VPC security group of the EC2 instance as the source. A security group also can't be used if it has been modified.	There are one or more security groups associated with the EC2 instance with a name that matches the pattern `rds-ec2-`*n*. A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the Multi-AZ DB cluster as the source.	RDS action: create new security groups

RDS action: create new security groups

Amazon RDS takes the following actions:

- Creates a new security group that matches the pattern `rds-ec2-`*n*. This security group has an inbound rule with the VPC security group of the EC2 instance as the source. This security group is associated with the Multi-AZ DB cluster and allows the EC2 instance to access the Multi-AZ DB cluster.
- Creates a new security group that matches the pattern `ec2-rds-`*n*. This security group has an outbound rule with the VPC security group of the Multi-AZ DB cluster as the source. This security group is associated with the EC2 instance and allows the EC2 instance to send traffic to the Multi-AZ DB cluster.

RDS action: associate EC2 security group

Amazon RDS associates the valid, existing EC2 security group with the EC2 instance. This security group allows the EC2 instance to send traffic to the Multi-AZ DB cluster.

Connecting an EC2 instance and a Multi-AZ DB cluster automatically

Before setting up a connection between an EC2 instance and an RDS database, make sure you meet the requirements described in Overview of automatic connectivity with an EC2 instance.

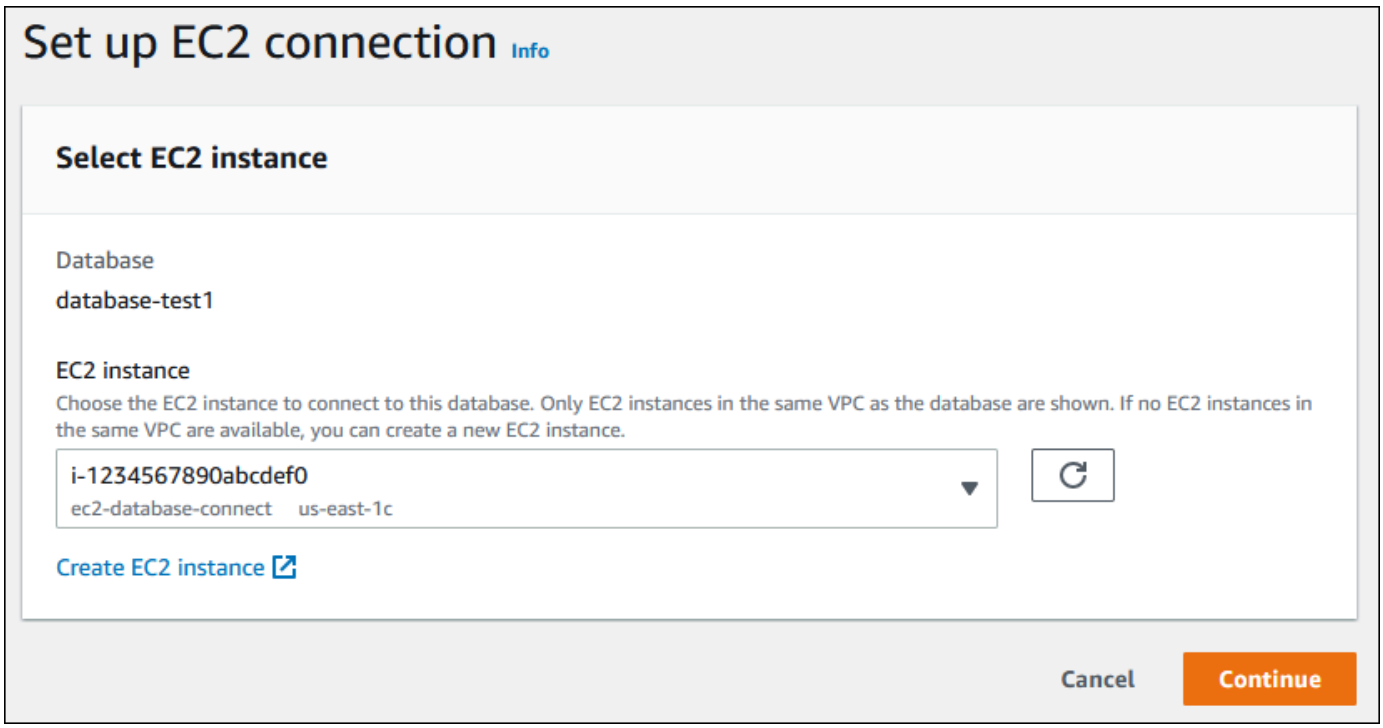
If you make changes to security groups after you configure connectivity, the changes might affect the connection between the EC2 instance and the RDS database.

> **Note**
>
> You can only set up a connection between an EC2 instance and an RDS database automatically by using the AWS Management Console. You can't set up a connection automatically with the AWS CLI or RDS API.

To connect an EC2 instance and an RDS database automatically

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS database.
3. From **Actions**, choose **Set up EC2 connection**.

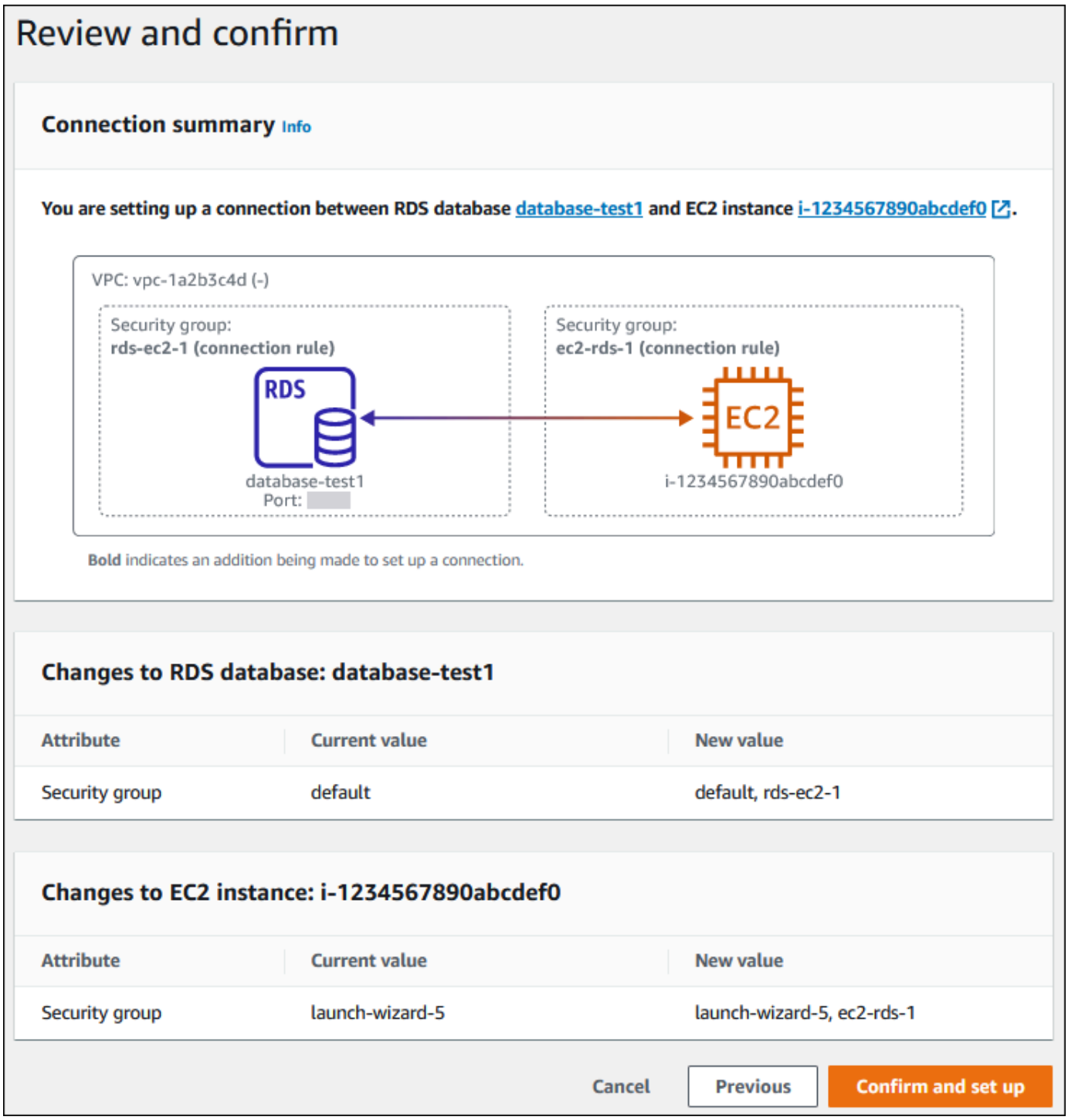
   The **Set up EC2 connection** page appears.
4. On the **Set up EC2 connection** page, choose the EC2 instance.



If no EC2 instances exist in the same VPC, choose **Create EC2 instance** to create one. In this case, make sure the new EC2 instance is in the same VPC as the RDS database.

5. Choose **Continue**.

   The **Review and confirm** page appears.



6. On the **Review and confirm** page, review the changes that RDS will make to set up connectivity with the EC2 instance.

   If the changes are correct, choose **Confirm and set up**.

   If the changes aren't correct, choose **Previous** or **Cancel**.

Viewing connected compute resources

You can use the AWS Management Console to view the compute resources that are connected to an RDS database. The resources shown include compute resource connections that were set up automatically. You can set up connectivity with compute resources automatically in the following ways:

- You can select the compute resource when you create the database.

  For more information, see Creating an Amazon RDS DB instance and Creating a Multi-AZ DB cluster for Amazon RDS.
- You can set up connectivity between an existing database and a compute resource.

  For more information, see Automatically connecting an EC2 instance and an RDS database.

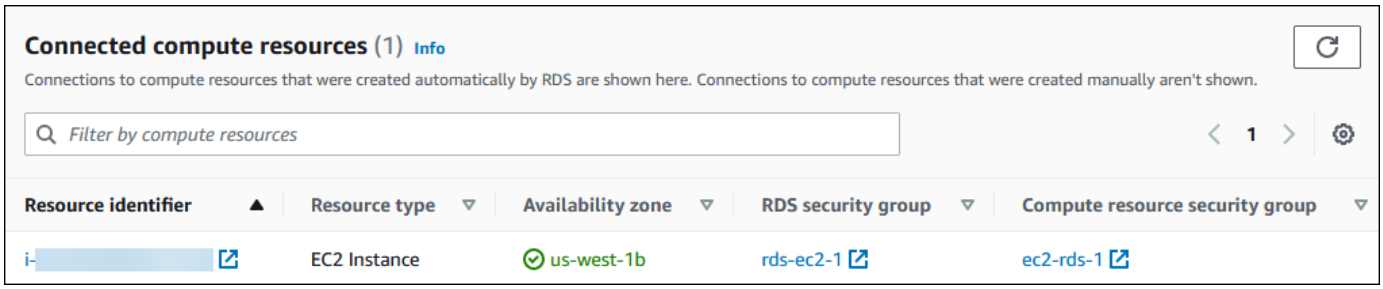
The listed compute resources don't include ones that were connected to the database manually. For example, you can allow a compute resource to access a database manually by adding a rule to the VPC security group associated with the database.

For a compute resource to be listed, the following conditions must apply:

- The name of the security group associated with the compute resource matches the pattern `ec2-rds-`*n* (where *n* is a number).
- The security group associated with the compute resource has an outbound rule with the port range set to the port that the RDS database uses.
- The security group associated with the compute resource has an outbound rule with the source set to a security group associated with the RDS database.
- The name of the security group associated with the RDS database matches the pattern `rds-ec2-`*n* (where *n* is a number).
- The security group associated with the RDS database has an inbound rule with the port range set to the port that the RDS database uses.
- The security group associated with the RDS database has an inbound rule with the source set to a security group associated with the compute resource.

To view compute resources connected to an RDS database

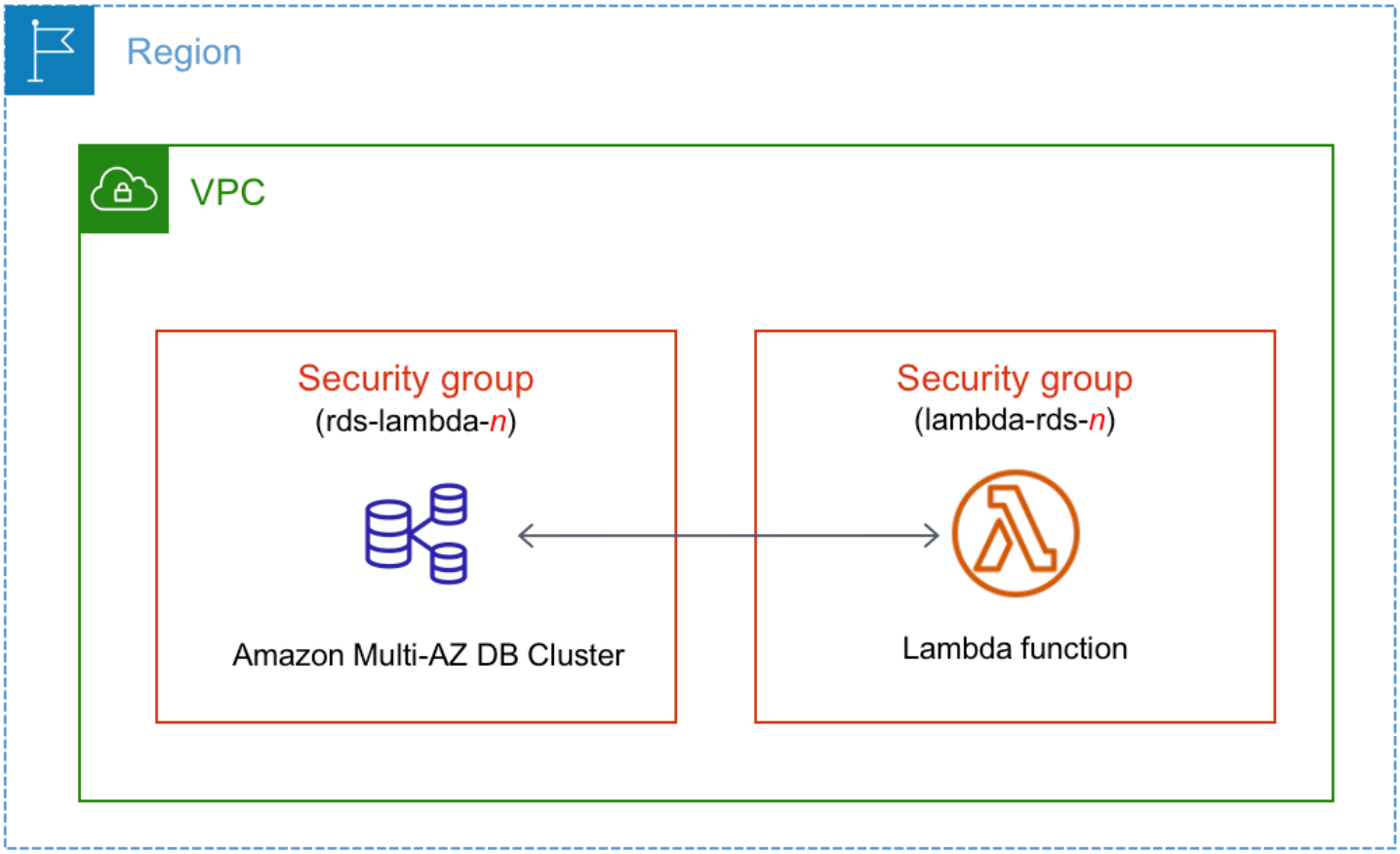
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the name of the RDS database.
3. On the **Connectivity & security** tab, view the compute resources in the **Connected compute resources**.



Automatically connecting a Lambda function and a Multi-AZ DB cluster

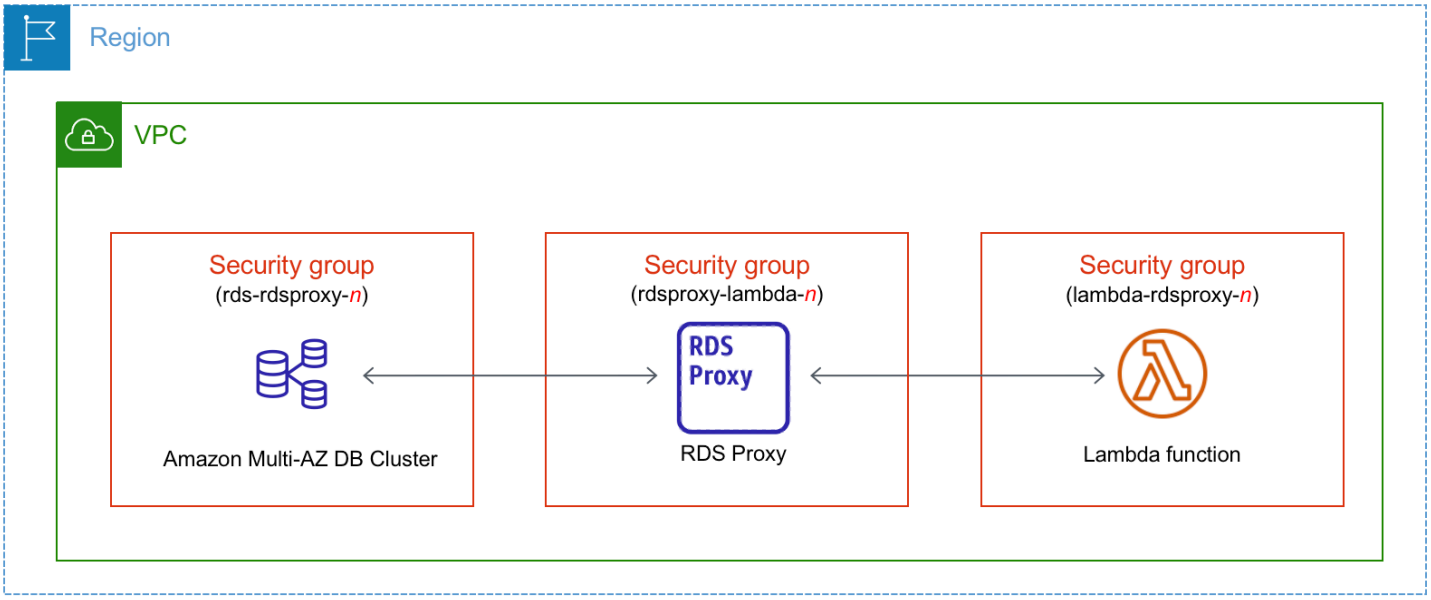
You can use the RDS console to simplify setting up a connection between a Lambda function and a Multi-AZ DB cluster. You can use the RDS console to simplify setting up a connection between a Lambda function and a Multi-AZ DB cluster. Often, your Multi-AZ DB cluster is in a private subnet within a VPC. The Lambda function can be used by applications to access your private Multi-AZ DB cluster.

The following image shows a direct connection between your Multi-AZ DB cluster and your Lambda function.



You can set up the connection between your Lambda function and your database through RDS Proxy to improve your database performance and resiliency. Often, Lambda functions make frequent, short database connections that benefit from connection pooling that RDS Proxy offers. You can take advantage of any IAM authentication that you already have for Lambda functions, instead of managing database credentials in your Lambda application code. For more information, see Amazon RDS Proxy.

You can use the console to automatically create a proxy for your connection. You can also select existing proxies. The console updates the proxy security group to allow connections from your database and Lambda function. You can input your database credentials or select the Secrets Manager secret you require to access the database.



Topics

- Overview of automatic connectivity with a Lambda function
- Automatically connecting a Lambda function and a Multi-AZ DB cluster
- Viewing connected compute resources

Overview of automatic connectivity with a Lambda function

When you set up a connection between a Lambda function and a Multi-AZ DB cluster automatically, Amazon RDS configures the VPC security group for your Lambda function and for your DB cluster.

The following are requirements for connecting a Lambda function with a Multi-AZ DB cluster:

- The Lambda function must exist in the same VPC as the Multi-AZ DB cluster.

  If no Lambda function exists in the same VPC, the console provides a link to create one.
- The user who sets up connectivity must have permissions to perform the following Amazon RDS, Amazon EC2, Lambda, Secrets Manager, and IAM operations:
  - Amazon RDS
    - `rds:CreateDBProxies`
    - `rds:DescribeDBInstances`
    - `rds:DescribeDBProxies`

    - rds:ModifyDBInstance
    - rds:ModifyDBProxy
    - rds:RegisterProxyTargets
- Amazon EC2
    - ec2:AuthorizeSecurityGroupEgress
    - ec2:AuthorizeSecurityGroupIngress
    - ec2:CreateSecurityGroup
    - ec2:DeleteSecurityGroup
    - ec2:DescribeSecurityGroups
    - ec2:RevokeSecurityGroupEgress
    - ec2:RevokeSecurityGroupIngress
- Lambda
    - lambda:CreateFunctions
    - lambda:ListFunctions
    - lambda:UpdateFunctionConfiguration
- Secrets Manager
    - sercetsmanager:CreateSecret
    - secretsmanager:DescribeSecret
- IAM
    - iam:AttachPolicy
    - iam:CreateRole
    - iam:CreatePolicy
- AWS KMS
    - kms:describeKey

When you set up a connection between a Lambda function and a Multi-AZ DB cluster, Amazon RDS configures the VPC security group for your function and for your Multi-AZ DB cluster. If you use RDS Proxy, then Amazon RDS also configures the VPC security group for the proxy. Amazon RDS acts according to the current configuration of the security groups associated with the Multi-AZ DB cluster and Lambda function, and proxy, as described in the following table.

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
Amazon RDS takes no action because security groups of all resources follow the correct naming pattern and have the right inbound and outbound rules.	There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-lambda-` *n* (where *n* is a number) or if the `TargetHealth` of an associated proxy is `AVAILABLE`. A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the Lambda function or proxy as the source.	There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-` *n* or `lambda-rdsproxy-` *n* (where *n* is a number). A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with either the VPC security group of the Multi-AZ DB cluster or the proxy as the destination.	There are one or more security groups associated with the proxy with a name that matches the pattern `rdsproxy-lambda-` *n* (where *n* is a number). A security group that matches the pattern hasn't been modified. This security group has inbound and outbound rules with the VPC security groups of the Lambda function and the Multi-AZ DB cluster.
Either of the following conditions apply: • There is no security group associated with the Multi-AZ	Either of the following conditions apply: • There is no security group associated with the Lambda	Either of the following conditions apply: • There is no security group associated with the proxy with a name	RDS action: create new security groups

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
DB cluster with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. • There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. However, Amazon RDS can't use any of these security groups for the connection with the Lambda function. Amazon RDS can't use a security group that doesn't have one inbound rule with	function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. • There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the Multi-AZ DB cluster. Amazon RDS can't use a security group if it doesn't have one outbound rule with the VPC security group of the Multi-AZ	that matches the pattern `rdsproxy-lambda-`*n*. • There are one or more security groups associated with the proxy with a name that matches `rdsproxy-lambda-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the Multi-AZ DB cluster or Lambda function. Amazon RDS can't use a security group that doesn't have inbound and outbound rules with the VPC security group of the Multi-AZ DB cluster and the Lambda function. Amazon RDS also	

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
the VPC security group of the Lambda function or proxy as the source. Amazon RDS also can't use a security group that has been modi fied. Examples of modifications include adding a rule or changing the port of an existing rule.	DB cluster or proxy as the source. Amazon RDS also can't use a security group that has been modified.	can't use a security group that has been modified.	

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the Lambda function or proxy as the source.	There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the Multi-AZ DB cluster. Amazon RDS can't use a security group that doesn't have one outbound rule with the VPC security group of the Multi-AZ DB cluster or proxy as the destination. Amazon RDS also can't use a security group that has been modified.	There are one or more security groups associated with the proxy with a name that matches the pattern `rdsproxy-lambda-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the Multi-AZ DB cluster or Lambda function. Amazon RDS can't use a security group that doesn't have inbound and outbound rules with the VPC security group of the Multi-AZ DB cluster and the Lambda function. Amazon RDS also can't use a security group that has been modified.	RDS action: create new security groups

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE` . A security group that matches the pattern hasn't been modified. This security group has only one inbound rule with the VPC security group of the Lambda function or proxy as the source.	A valid Lambda security group for the connection exists, but it is not associated with the Lambda function. This security group has a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. It hasn't been modified. It has only one outbound rule with the VPC security group of the Multi-AZ DB cluster or proxy as the destination.	A valid proxy security group for the connection exists, but it is not associated with the proxy. This security group has a name that matches the pattern `rdsproxy-lambda-`*n*. It hasn't been modified. It has inbound and outbound rules with the VPC security group of the Multi-AZ DB cluster and the Lambda function.	RDS action: associate Lambda security group

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
Either of the following conditions apply: • There is no security group associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. • There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-lambda-`*n* or if the `TargetHealth` of an associated proxy is `AVAILABLE`. However, Amazon RDS can'can't	There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. A security group that matches the pattern hasn't been modified. This security group has only one outbound rule with the VPC security group of the Multi-AZ DB cluster or proxy as the destination.	There are one or more security groups associated with the proxy with a name that matches the pattern `rdsproxy-lambda-`*n*. A security group that matches the pattern hasn't been modified. This security group has inbound and outbound rules with the VPC security group of the Multi-AZ DB cluster and the Lambda function.	RDS action: create new security groups

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
use any of these security groups for the connection with the Lambda function or proxy. Amazon RDS can't use a security group that doesn't have one inbound rule with the VPC security group of the Lambda function or proxy as the source. Amazon RDS also can't use a security group that has been modified.			

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
There are one or more security groups associated with the Multi-AZ DB cluster with a name that matches the pattern `rds-rdsproxy-`*n* (where *n* is a number).	Either of the following conditions apply: • There is no security group associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. • There are one or more security groups associated with the Lambda function with a name that matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n*. However, Amazon RDS can't use any of these security groups for the connection with	Either of the following conditions apply: • There is no security group associated with the proxy with a name that matches the pattern `rdsproxy-lambda-`*n*. • There are one or more security groups associated with the proxy with a name that matches `rdsproxy-lambda-`*n*. However, Amazon RDS can't use any of these security groups for the connection with the Multi-AZ DB cluster or Lambda function. Amazon RDS can't use a security group	RDS action: create new security groups

Current RDS security group configuration	Current Lambda security group configuration	Current proxy security group configuration	RDS action
	the Multi-AZ DB cluster. Amazon RDS can't use a security group that doesn't have one outbound rule with the VPC security group of the Multi-AZ DB cluster or proxy as the destination. Amazon RDS also can't use a security group that has been modified.	that doesn't have inbound and outbound rules with the VPC security group of the Multi-AZ DB cluster and the Lambda function. Amazon RDS also can't use a security group that has been modified.	

RDS action: create new security groups

Amazon RDS takes the following actions:

- Creates a new security group that matches the pattern `rds-lambda-`*n*.This security group has an inbound rule with the VPC security group of the Lambda function or proxy as the source. This security group is associated with the Multi-AZ DB cluster and allows the function or proxy to access the Multi-AZ DB cluster.
- Creates a new security group that matches the pattern `lambda-rds-`*n*. This security group has an outbound rule with the VPC security group of the Multi-AZ DB cluster or proxy as the destination. This security group is associated with the Lambda function and allows the Lambda function to send traffic to the Multi-AZ DB cluster or send traffic through a proxy.
- Creates a new security group that matches the pattern `rdsproxy-lambda-`*n*. This security group has inbound and outbound rules with the VPC security group of the Multi-AZ DB cluster and the Lambda function.

RDS action: associate Lambda security group

Amazon RDS associates the valid, existing Lambda security group with the Lambda function. This security group allows the function to send traffic to the Multi-AZ DB cluster or send traffic through a proxy.

Automatically connecting a Lambda function and a Multi-AZ DB cluster

You can use the Amazon RDS console to automatically connect a Lambda function to your Multi-AZ DB cluster. This simplifies the process of setting up a connection between these resources.

You can also use RDS Proxy to include a proxy in your connection. Lambda functions make frequent short database connections that benefit from the connection pooling that RDS Proxy offers. You can also use any IAM authentication that you've already set up for your Lambda function, instead of managing database credentials in your Lambda application code.

You can connect an existing Multi-AZ DB cluster to new and existing Lambda functions using the **Set up Lambda connection** page. The setup process automatically sets up the required security groups for you.

Before setting up a connection between a Lambda function and a Multi-AZ DB cluster, make sure that:

- Your Lambda function and Multi-AZ DB cluster are in the same VPC.
- You have the right permissions for your user account. For more information about the requirements, see Overview of automatic connectivity with a Lambda function.

If you change security groups after you configure connectivity, the changes might affect the connection between the Lambda function and the Multi-AZ DB cluster.

> **Note**
>
> You can automatically set up a connection between a Multi-AZ DB cluster and a Lambda function only in the AWS Management Console. To connect a Lambda function, all instances in the Multi-AZ DB cluster must be in the **Available** state.

To automatically connect a Lambda function and a Multi-AZ DB cluster

<result>

After you confirm the setup, Amazon RDS begins the process of connecting your Lambda function, RDS Proxy (if you used a proxy), and Multi-AZ DB cluster. The console shows the **Connection details** dialog box, which lists the security group changes that allow connections between your resources.
</result>

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the Multi-AZ DB cluster that you want to connect to a Lambda function.
3. For **Actions**, choose **Set up Lambda connection**.
4. On the **Set up Lambda connection** page, under **Select Lambda function**, do either of the following:
   - If you have an existing Lambda function in the same VPC as your Multi-AZ DB cluster, choose **Choose existing function**, and then choose the function.
   - If you don't have a Lambda function in the same VPC, choose **Create new function**, and then enter a **Function name**. The default runtime is set to Nodejs.18. You can modify the settings for your new Lambda function in the Lambda console after you complete the connection setup.
5. (Optional) Under **RDS Proxy**, select **Connect using RDS Proxy**, and then do any of the following:
   - If you have an existing proxy that you want to use, choose **Choose existing proxy**, and then choose the proxy.
   - If you don't have a proxy, and you want Amazon RDS to automatically create one for you, choose **Create new proxy**. Then, for **Database credentials**, do either of the following:
     a. Choose **Database username and password**, and then enter the **Username** and **Password** for your Multi-AZ DB cluster.
     b. Choose **Secrets Manager secret**. Then, for **Select secret**, choose an AWS Secrets Manager secret. If you don't have a Secrets Manager secret, choose **Create new Secrets Manager secret** to create a new secret. After you create the secret, for **Select secret**, choose the new secret.

     After you create the new proxy, choose **Choose existing proxy**, and then choose the proxy. Note that it might take some time for your proxy to be available for connection.

6. (Optional) Expand **Connection summary** and verify the highlighted updates for your resources.
7. Choose **Set up**.

Viewing connected compute resources

You can use the AWS Management Console to view the compute resources that are connected to your Multi-AZ DB cluster. The resources shown include compute resource connections that Amazon RDS set up automatically.

The listed compute resources don't include those that are manually connected to the Multi-AZ DB cluster. For example, you can allow a compute resource to access your Multi-AZ DB cluster manually by adding a rule to your VPC security group associated with the cluster.

For the console to list a Lambda function, the following conditions must apply:

- The name of the security group associated with the compute resource matches the pattern `lambda-rds-`*n* or `lambda-rdsproxy-`*n* (where *n* is a number).
- The security group associated with the compute resource has an outbound rule with the port range set to the port of the Multi-AZ DB cluster or an associated proxy. The destination for the outbound rule must be set to a security group associated with the Multi-AZ DB cluster or an associated proxy.
- The name of the security group attached to the proxy associated with your database matches the pattern `rds-rdsproxy-`*n* (where *n* is a number).
- The security group associated with the function has an outbound rule with the port set to the port that the Multi-AZ DB cluster or associated proxy uses. The destination must be set to a security group associated with the Multi-AZ DB cluster or associated proxy.

To view compute resources automatically connected to a Multi-AZ DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the Multi-AZ DB cluster.
3. On the **Connectivity & security** tab, view the compute resources under **Connected compute resources**.

Modifying a Multi-AZ DB cluster for Amazon RDS

A Multi-AZ DB cluster has a writer DB instance and two reader DB instances in three separate Availability Zones. Multi-AZ DB clusters provide high availability, increased capacity for read workloads, and lower latency when compared to Multi-AZ deployments. For more information about Multi-AZ DB clusters, see Multi-AZ DB cluster deployments for Amazon RDS.

You can modify a Multi-AZ DB cluster to change its settings. You can also perform operations on a Multi-AZ DB cluster, such as taking a snapshot of it.

> **Important**
>
> You can't modify the DB instances within a Multi-AZ DB cluster. All modifications must be done at the DB cluster level. The only operation you can perform on a DB instance within a Multi-AZ DB cluster is rebooting it.

You can modify a Multi-AZ DB cluster using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To modify a Multi-AZ DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the Multi-AZ DB cluster that you want to modify.
3. Choose **Modify**. The **Modify DB cluster** page appears.
4. Change any of the settings that you want. For information about each setting, see Settings for modifying Multi-AZ DB clusters.
5. When all the changes are as you want them, choose **Continue** and check the summary of modifications.
6. (Optional) Choose **Apply immediately** to apply the changes immediately. Choosing this option can cause downtime in some cases. For more information, see Applying changes immediately.
7. On the confirmation page, review your changes. If they're correct, choose **Modify DB cluster** to save your changes.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To modify a Multi-AZ DB cluster by using the AWS CLI, call the modify-db-cluster command. Specify the DB cluster identifier and the values for the options that you want to modify. For information about each option, see Settings for modifying Multi-AZ DB clusters.

Example

The following code modifies `my-multi-az-dbcluster` by setting the backup retention period to 1 week (7 days). The code turns on deletion protection by using `--deletion-protection`. To turn off deletion protection, use `--no-deletion-protection`. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately. For more information, see Applying changes immediately.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-multi-az-dbcluster \
    --backup-retention-period 7 \
    --deletion-protection \
    --no-apply-immediately
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier my-multi-az-dbcluster ^
    --backup-retention-period 7 ^
    --deletion-protection ^
    --no-apply-immediately
```

RDS API

To modify a Multi-AZ DB cluster by using the Amazon RDS API, call the ModifyDBCluster operation. Specify the DB cluster identifier, and the parameters for the settings that you want to modify. For information about each parameter, see Settings for modifying Multi-AZ DB clusters.

Applying changes immediately

When you modify a Multi-AZ DB cluster, you can apply the changes immediately. To apply changes immediately, you choose the **Apply Immediately** option in the AWS Management Console. Or you use the `--apply-immediately` option when calling the AWS CLI or set the `ApplyImmediately` parameter to `true` when using the Amazon RDS API.

If you don't choose to apply changes immediately, the changes are put into the pending modifications queue. During the next maintenance window, any pending changes in the queue are applied. If you choose to apply changes immediately, your new changes and any changes in the pending modifications queue are applied.

> **Important**
>
> If any of the pending modifications require the DB cluster to be temporarily unavailable (*downtime*), choosing the apply immediately option can cause unexpected downtime. When you choose to apply a change immediately, any pending modifications are also applied immediately, instead of during the next maintenance window.
> If you don't want a pending change to be applied in the next maintenance window, you can modify the DB instance to revert the change. You can do this by using the AWS CLI and specifying the `--apply-immediately` option.

Changes to some database settings are applied immediately, even if you choose to defer your changes. To see how the different database settings interact with the apply immediately setting, see Settings for modifying Multi-AZ DB clusters.

Settings for modifying Multi-AZ DB clusters

For details about settings that you can use to modify a Multi-AZ DB cluster, see the following table. For more information about the AWS CLI options, see modify-db-cluster. For more information about the RDS API parameters, see ModifyDBCluster.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes
Allocated storage	The amount of storage to allocate for each	**CLI option:**	If you choose to apply the change immediately,	Downtime doesn't occur during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

	DB instance in your DB cluster (in gibibyte). For more information, see Amazon RDS DB instance storage.	`--allocated-storage`
		
		RDS API parameter:
		
		`AllocatedStorage`
			it occurs immediately.
			
			If you don't choose to apply the change immediately, it occurs during the next maintenance window.	

Auto minor version upgrade	**Enable auto minor version upgrade** to have your DB cluster receive preferred minor DB engine version upgrades automatically when they become available. Amazon RDS performs automatic minor version upgrades in the maintenance window.	**CLI option:**
		
		`--auto-minor-version-upgrade`
		
		`--no-auto-minor-version-upgrade`
		
		RDS API parameter:
		
		`AutoMinorVersionUpgrade`
			The change occurs immediately. This setting ignores the apply immediately setting.	Downtime occurs during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

Backup retention period	The number of days that automatic backups of your DB cluster are retained. This value must be greater than zero.
	
	For more information, see Introduction to backups.
		CLI option:
		
		`--backup-retention-period`
		
		RDS API parameter:
		
		`BackupRetentionPeriod`
			If you choose to apply the change immediately, it occurs immediately.
			
			If you don't choose to apply the change immediately, it occurs during the next maintenance window.
				Downtime doesn't occur during this change.

Backup window	The time period during which Amazon RDS automatically takes a backup of your DB cluster. Unless you have a specific time that you want to have your database backed up, use the default of **No preference**.
	
	For more information, see Introduction to backups.
		CLI option:
		
		`--preferred-backup-window`
		
		RDS API parameter:
		
		`PreferredBackupWindow`
			The change is applied asynchronously, as soon as possible.	Downtime doesn't occur during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

Certificate authority	The certificate authority (CA) for the server certificate used by the DB cluster.
	
	For more information, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
		CLI option:
		
		`--ca-certificate-identifier`
		
		RDS API parameter:
		
		`CACertificateIdentifier`
			If you choose to apply the change immediately, it occurs immediately.
			
			If you don't choose to apply the change immediately, it occurs during the next maintenance window.
				Downtime only occurs if the DB engine doesn't support rotation without restart. You can use the describe-db-engine-versions AWS CLI command to determine whether the DB engine supports rotation without restart.

Copy tags to snapshot	This option copies any DB cluster tags to a DB snapshot when you create a snapshot.
	
	For more information, see Tagging Amazon RDS resources.
		CLI option:
		
		`-copy-tags-to-snapshot`
		
		`-no-copy-tags-to-snapshot`
		
		RDS API parameter:
		
		`CopyTagsToSnapshot`
			The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes
Database authentication	For Multi-AZ DB clusters, only **Password authentication** is supported.	None because password authentication is the default.	If you choose to apply the change immediately, it occurs immediately. If you don't choose to apply the change immediately, it occurs during the next maintenance window.	Downtime doesn't occur during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

DB cluster identifier	The DB cluster identifier. This value is stored as a lowercase string.
	
	When you change the DB cluster identifier, the DB cluster endpoint changes. The identifiers and endpoints of the DB instances in the DB cluster also change. The new DB cluster name must be unique. The maximum length is 63 characters.
	
	The names of the DB instances in the DB cluster are changed to correspond with the new name of the DB cluster. A new DB instance name can't be the same as the name
		CLI option:
		
		`--new-db-cluster-identifier`
		
		RDS API parameter:
		
		`NewDBClusterIdentifier`
			If you choose to apply the change immediately, it occurs immediately.
			
			If you don't choose to apply the change immediately, it occurs during the next maintenance window.
				Downtime doesn't occur during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes
	of an existing DB instance. For example, if you change the DB cluster name to `maz`, a DB instance name might be changed to `maz-instance-1` . In this case, there can't be an existing DB instance named `maz-instance-1` . For more information, see Renaming a Multi-AZ DB cluster for Amazon RDS.			

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

DB cluster instance class	The compute and memory capacity of each DB instance in the Multi-AZ DB cluster, for example `db.r6gd.xlarge`.
	
	If possible, choose a DB instance class large enough that a typical query working set can be held in memory. When working sets are held in memory, the system can avoid writing to disk, which improves performance.
	
	For more information, see the section called "Instance class availability for Multi-AZ DB clusters".
		CLI option:
		
		`--db-cluster-instance-class`
		
		RDS API parameter:
		
		`DBClusterInstanceClass`
			If you choose to apply the change immediately, it occurs immediately.
			
			If you don't choose to apply the change immediately, it occurs during the next maintenance window.
				Downtime occurs during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

DB cluster parameter group	The DB cluster parameter group that you want associated with the DB cluster.
	
	For more information, see Parameter groups for Multi-AZ DB clusters.
		CLI option:
		
		`--db-cluster-parameter-group-name`
		
		RDS API parameter:
		
		`DBClusterParameterGroupName`
			The parameter group change occurs immediately.	Downtime doesn't occur during this change. When you change the parameter group, changes to some parameters are applied to the DB instances in the Multi-AZ DB cluster immediately without a reboot. Changes to other parameters are applied only after the DB instances are rebooted.

DB engine version	The version of database engine that you want to use.	**CLI option:**
		
		`--engine-version`
		
		RDS API parameter:
		
		`EngineVersion`
			If you choose to apply the change immediately, it occurs immediately.
			
			If you don't choose to apply the change immediately, it occurs during the next maintenance window.
				Downtime occurs during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

Deletion protection	**Enable deletion protection** to prevent your DB cluster from being deleted.
	
	For more information, see Deleting a DB instance.
		CLI option:
		
		`--deletion-protection`
		
		`--no-deletion-protection`
		
		RDS API parameter:
		
		`DeletionProtection`
			The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.

Maintenance window	The 30-minute window in which pending modifications to your DB cluster are applied. If the time period doesn't matter, choose **No preference**.
	
	For more information, see Amazon RDS maintenance window.
		CLI option:
		
		`--preferred-maintenance-window`
		
		RDS API parameter:
		
		`PreferredMaintenanceWindow`
			The change occurs immediately. This setting ignores the apply immediately setting.	If there are one or more pending actions that cause downtime, and the maintenance window is changed to include the current time, those pending actions are applied immediately and downtime occurs.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

Manage master credentials in AWS Secrets Manager	Select **Manage master credentials in AWS Secrets Manager** to manage the master user password in a secret in Secrets Manager.
	
	Optionally, choose a KMS key to use to protect the secret. Choose from the KMS keys in your account, or enter the key from a different account.
	
	If RDS is already managing the master user password for the DB cluster, you can rotate the master user password by choosing **Rotate secret immediately**.
		CLI option:
		
		`--manage-master-user-password | --no-manage-master-user-password`
		
		`--master-user-secret-kms-key-id`
		
		`--rotate-master-user-password | --no-rotate-master-user-password`
		
		RDS API parameter:
		
		`ManageMasterUserPassword`
		
		`MasterUserSecretKmsKeyId`
		
		`RotateMasterUserPassword`
			If you are turning on or turning off automatic master user password management, the change occurs immediately. This change ignores the apply immediately setting.
			
			If you are rotating the master user password, you must specify that the change is applied immediately.
				Downtime doesn't occur during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes
	For more information, see Password management with Amazon RDS and AWS Secrets Manager.			

New master password	The password for your master user account.	**CLI option:**
		`--master-user-password`
		RDS API parameter:
		`MasterUserPassword`
			The change is applied asynchronously, as soon as possible. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.

Provisioned IOPS	The amount of Provisioned IOPS (input/output operations per second) to be initially allocated for the DB cluster.	**CLI option:**
		`--iops`
		RDS API parameter:
		`Iops`
			If you choose to apply the change immediately, it occurs immediately.
			If you don't choose to apply the change immediately, it occurs during the next maintenance window.
				Downtime doesn't occur during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes
Public access	**Publicly accessible** to give the DB cluster a public IP address, meaning that it's accessible outside its virtual private cloud (VPC). To be publicly accessible, the DB cluster also has to be in a public subnet in the VPC. **Not publicly accessible** to make the DB cluster accessible only from inside the VPC. For more information, see Hiding a DB instance in a VPC from the internet. To connect to a DB cluster from outside of its VPC, the	Not available when modifying a DB cluster.	The change occurs immediately. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes
	DB cluster must be publicly accessible. Also, access must be granted using the inbound rules of the DB cluster's security group, and other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance. If your DB cluster isn't publicly accessible, you can use an AWS Site-to-Site VPN connection or an Direct Connect connection to access it from a private network. For more information, see Internetwork traffic privacy.			

Console setting	Setting description	CLI option and RDS API parameter	When the change occurs	Downtime notes

Storage type	The storage type for your DB cluster.
	
	Only General Purpose SSD (gp3), Provisioned IOPS (io1), and Provisioned IOPS SSD (io2) storage are supported.
	
	For more information, see Amazon RDS storage types.
		CLI option:
		
		`--storage-type`
		
		RDS API parameter:
		
		`StorageType`
			If you choose to apply the change immediately, it occurs immediately.
			
			If you don't choose to apply the change immediately, it occurs during the next maintenance window.
				Downtime doesn't occur during this change.

VPC security group	The security groups to associate with the DB cluster.
	
	For more information, see Overview of VPC security groups.
		CLI option:
		
		`--vpc-security-group-ids`
		
		RDS API parameter:
		
		`VpcSecurityGroupIds`
			The change is applied asynchronously, as soon as possible. This setting ignores the apply immediately setting.	Downtime doesn't occur during this change.

Settings that don't apply when modifying Multi-AZ DB clusters

The following settings in the AWS CLI command modify-db-cluster and the RDS API operation ModifyDBCluster don't apply to Multi-AZ DB clusters.

You also can't modify these settings for Multi-AZ DB clusters in the console.

AWS CLI setting	RDS API setting
--backtrack-window	BacktrackWindow
--cloudwatch-logs-export-configuration	CloudwatchLogsExportConfiguration
--copy-tags-to-snapshot | --no-copy-tags-to-snapshot	CopyTagsToSnapshot
--db-instance-parameter-group-name	DBInstanceParameterGroupName
--domain	Domain
--domain-iam-role-name	DomainIAMRoleName
--enable-global-write-forwarding | --no-enable-global-write-forwarding	EnableGlobalWriteForwarding
--enable-http-endpoint | --no-enable-http-endpoint	EnableHttpEndpoint
--option-group-name	OptionGroupName
--port	Port
--scaling-configuration	ScalingConfiguration
--storage-type	StorageType

Upgrading the engine version of a Multi-AZ DB cluster for Amazon RDS

Amazon RDS provides newer versions of each supported database engine so that you can keep your Multi-AZ DB cluster up to date. This topic explains the process of upgrading a Multi-AZ DB cluster to newer versions.

Upgrading a Multi-AZ DB cluster involves selecting a new compatible engine version and planning for potential downtime. The process ensures minimal disruption by utilizing the failover capabilities of the Multi-AZ architecture. Best practices include performing upgrades during low-traffic periods, testing in non-production environments, and verifying application compatibility with the new version.

Topics

- Minor version upgrades
- Major version upgrades
- Upgrading a Multi-AZ DB cluster
- Upgrading Multi-AZ DB cluster read replicas
- Monitoring Multi-AZ DB cluster upgrades with events

Minor version upgrades

A minor version upgrade includes only changes that are backward-compatible with existing applications. When you initiate a minor version upgrade, Amazon RDS first upgrades the reader DB instances one at a time. Then, one of the reader DB instances switches to be the new writer DB instance. Amazon RDS then upgrades the old writer instance (which is now a reader instance).

Downtime during the upgrade is limited to the time it takes for one of the reader DB instances to become the new writer DB instance. This downtime acts like an automatic failover. For more information, see the section called “Failing over a Multi-AZ DB cluster”. Note that the replica lag of your Multi-AZ DB cluster might affect the downtime. For more information, see the section called “Replica lag and Multi-AZ DB clusters”.

For RDS for PostgreSQL Multi-AZ DB cluster read replicas, Amazon RDS upgrades the cluster member instances one at a time. The reader and writer cluster roles don't switch during the upgrade. Therefore, your DB cluster might experience downtime while Amazon RDS upgrades the cluster writer instance.

Note

The downtime for a Multi-AZ DB cluster minor version upgrade is typically 35 seconds. When used with RDS Proxy, you can further reduce downtime to one second or less. For more information, see *Amazon RDS Proxy*. Alternately, you can use an open source database proxy such as ProxySQL, PgBouncer, or the AWS Advanced JDBC Wrapper Driver.

Major version upgrades

A major version upgrade can introduce changes that aren't compatible with existing applications.

When you initiate a major version upgrade of an RDS for PostgreSQL Multi-AZ DB cluster, Amazon RDS simultaneously upgrades the reader and writer instances. Therefore, your DB cluster might not be available until the upgrade completes.

When you initiate a major version upgrade of an RDS for MySQL Multi-AZ DB cluster, Amazon RDS upgrades the cluster member instances one at a time, so replication occurs from a lower engine version to a higher one. It's important to ensure that your workload is compatible with both the source and target engine versions during a major version upgrade, as engine versions might differ in syntax and features.

Note

Like minor version upgrades, the downtime for an RDS for MySQL major version upgrade is typically 35 seconds. When used with RDS Proxy, you can further reduce downtime to one second or less. For more information, see *Amazon RDS Proxy*.

Upgrading a Multi-AZ DB cluster

The process for upgrading the engine version of a Multi-AZ DB cluster is the same as the process for upgrading a DB instance engine version. For instructions, see the section called "Upgrading the engine version". The only difference is that when using the AWS Command Line Interface (AWS CLI), you use the modify-db-cluster command and specify the `--db-cluster-identifier` parameter (along with the `--allow-major-version-upgrade` parameter).

For more information about major and minor version upgrades, see the following documentation for your DB engine:

- the section called “Upgrades of the PostgreSQL DB engine”
- the section called “Upgrades of the MySQL DB engine”

Upgrading Multi-AZ DB cluster read replicas

Amazon RDS doesn't automatically upgrade Multi-AZ DB cluster read replicas. For *minor* version upgrades, you must first manually upgrade all read replicas and then upgrade the cluster. Otherwise, the upgrade is blocked. When you perform a *major* version upgrade of a cluster, the replication state of all read replicas changes to **terminated**. You must delete and recreate the read replicas after the upgrade completes. For more information, see the section called “Monitoring read replication”.

Monitoring Multi-AZ DB cluster upgrades with events

When you upgrade the engine version of a Multi-AZ DB cluster, Amazon RDS emits a specific event during each phase of the process. To track the progress of an upgrade, you can view or subscribe to these events.

For more information about RDS events, see Monitoring Amazon RDS events.

For detailed information about a specific Amazon RDS event that occurs during your engine upgrade, see Amazon RDS event categories and event messages.

Renaming a Multi-AZ DB cluster for Amazon RDS

You can rename a Multi-AZ DB cluster by using the AWS Management Console, the AWS CLI `modify-db-cluster` command, or the Amazon RDS API `ModifyDBCluster` operation. Renaming a Multi-AZ DB cluster can have significant effects. The following is a list of considerations before you rename a Multi-AZ DB cluster.

- When you rename a Multi-AZ DB cluster, the cluster endpoints for the Multi-AZ DB cluster change. These endpoints change because they include the name you assigned to the Multi-AZ DB cluster. You can redirect traffic from an old endpoint to a new one. For more information about Multi-AZ DB cluster endpoints, see Connecting to a Multi-AZ DB cluster for Amazon RDS.
- When you rename a Multi-AZ DB cluster, the old DNS name that was used by the Multi-AZ DB cluster is deleted, although it could remain cached for a few minutes. The new DNS name for the renamed Multi-AZ DB cluster becomes effective in about two minutes. The renamed Multi-AZ DB cluster isn't available until the new name becomes effective.
- You can't use an existing Multi-AZ DB cluster name when renaming a cluster.
- Metrics and events associated with the name of a Multi-AZ DB cluster are maintained if you reuse a DB cluster name.
- Multi-AZ DB cluster tags remain with the Multi-AZ DB cluster, regardless of renaming.
- DB cluster snapshots are retained for a renamed Multi-AZ DB cluster.

> **Note**
>
> A Multi-AZ DB cluster is an isolated database environment running in the cloud. A Multi-AZ DB cluster can host multiple databases. For information about changing a database name, see the documentation for your DB engine.

Renaming to replace an existing Multi-AZ DB cluster

The most common scenarios for renaming a Multi-AZ DB cluster include restoring data from a DB cluster snapshot or performing point-in-time recovery (PITR). By renaming the Multi-AZ DB cluster, you can replace the Multi-AZ DB cluster without changing any application code that references the Multi-AZ DB cluster. In these cases, complete the following steps:

1. Stop all traffic going to the Multi-AZ DB cluster. You can redirect traffic from accessing the databases on the Multi-AZ DB cluster, or choose another way to prevent traffic from accessing your databases on the Multi-AZ DB cluster.
2. Rename the existing Multi-AZ DB cluster.
3. Create a new Multi-AZ DB cluster by restoring from a DB cluster snapshot or recovering to a point in time. Then, give the new Multi-AZ DB cluster the name of the previous Multi-AZ DB cluster.

If you delete the old Multi-AZ DB cluster, you are responsible for deleting any unwanted DB cluster snapshots of the old Multi-AZ DB cluster.

Console

To rename a Multi-AZ DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the Multi-AZ DB cluster that you want to rename.
4. Choose **Modify**.
5. In **Settings**, enter a new name for **DB cluster identifier**.
6. Choose **Continue**.
7. To apply the changes immediately, choose **Apply immediately**. Choosing this option can cause an outage in some cases. For more information, see Applying changes immediately.
8. On the confirmation page, review your changes. If they are correct, choose **Modify cluster** to save your changes.

   Alternatively, choose **Back** to edit your changes, or choose **Cancel** to discard your changes.

AWS CLI

To rename a Multi-AZ DB cluster, use the AWS CLI command modify-db-cluster. Provide the current `--db-cluster-identifier` value and `--new-db-cluster-identifier` parameter with the new name of the Multi-AZ DB cluster.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier DBClusterIdentifier \
    --new-db-cluster-identifier NewDBClusterIdentifier
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier DBClusterIdentifier ^
    --new-db-cluster-identifier NewDBClusterIdentifier
```

RDS API

To rename a Multi-AZ DB cluster, call the Amazon RDS API operation ModifyDBCluster with the following parameters:

- `DBClusterIdentifier` – The existing name of the DB cluster.
- `NewDBClusterIdentifier` – The new name of the DB cluster.

Rebooting a Multi-AZ DB cluster and reader DB instances for Amazon RDS

You might need to reboot your Multi-AZ DB cluster, usually for maintenance reasons. For example, if you make certain modifications or change the DB cluster parameter group associated with a DB cluster, you reboot the DB cluster. Doing so causes the changes to take effect.

If a DB cluster isn't using the latest changes to its associated DB cluster parameter group, the AWS Management Console shows the DB cluster parameter group with a status of **pending-reboot**. The **pending-reboot** parameter groups status doesn't result in an automatic reboot during the next maintenance window. To apply the latest parameter changes to that DB cluster, manually reboot the DB cluster. For more information about parameter groups, see Parameter groups for Multi-AZ DB clusters.

Rebooting a DB cluster restarts the database engine service. Rebooting a DB cluster results in a momentary outage, during which the DB cluster status is set to **rebooting**.

You can't reboot your DB cluster if it isn't in the **Available** state. Your database can be unavailable for several reasons, such as an in-progress backup, a previously requested modification, or a maintenance-window action.

The time required to reboot your DB cluster depends on the crash recovery process, the database activity at the time of reboot, and the behavior of your specific DB cluster. To improve the reboot time, we recommend that you reduce database activity as much as possible during the reboot process. Reducing database activity reduces rollback activity for in-transit transactions.

Important

Multi-AZ DB clusters don't support reboot with a failover. When you reboot the writer instance of a Multi-AZ DB cluster, it doesn't affect the reader DB instances in that DB cluster and no failover occurs. When you reboot a reader DB instance, no failover occurs. To fail over a Multi-AZ DB cluster, choose **Failover** in the console, call the AWS CLI command `failover-db-cluster`, or call the API operation `FailoverDBCluster`.

Console

To reboot a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the Multi-AZ DB cluster that you want to reboot.
3. For **Actions**, choose **Reboot**.

   The **Reboot DB cluster** page appears.
4. Choose **Reboot** to reboot your DB cluster.

   Or choose **Cancel**.

AWS CLI

To reboot a Multi-AZ DB cluster by using the AWS CLI, call the reboot-db-cluster command.

```
aws rds reboot-db-cluster --db-cluster-identifier mymultiazdbcluster
```

RDS API

To reboot a Multi-AZ DB cluster by using the Amazon RDS API, call the RebootDBCluster operation.

Failing over a Multi-AZ DB cluster for Amazon RDS

If there is a planned or unplanned outage of your writer DB instance in a Multi-AZ DB cluster, Amazon RDS automatically fails over to a reader DB instance in different Availability Zone. This ensures high availability with minimal disruption. Failovers can occur during hardware failures, network issues, or manual requests. The topic outlines the automatic detection of failures, the sequence of events during failover, and its impact on read and write operations. It also provides best practices for monitoring and minimizing failover times.

The time it takes for the failover to complete depends on the database activity and other conditions when the writer DB instance became unavailable. Failover times are typically under 35 seconds. Failover completes when both reader DB instances have applied outstanding transactions from the failed writer. When the failover is complete, it can take additional time for the RDS console to reflect the new Availability Zone.

Topics

- Automatic failovers
- Manually failing over a Multi-AZ DB cluster
- Determining whether a Multi-AZ DB cluster has failed over
- Setting the JVM TTL for DNS name lookups

Automatic failovers

Amazon RDS handles failovers automatically so you can resume database operations as quickly as possible without administrative intervention. To fail over, the writer DB instance switches automatically to a reader DB instance.

Manually failing over a Multi-AZ DB cluster

If you manually fail over a Multi-AZ DB cluster, RDS first terminates the primary DB instance. Then, the internal monitoring system detects that the primary DB instance is unhealthy and promotes a readable replica DB instance. Failover times are typically under 35 seconds.

You can fail over a Multi-AZ DB cluster manually using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To fail over a Multi-AZ DB cluster manually

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the Multi-AZ DB cluster that you want to fail over.
4. For **Actions**, choose **Failover**.

   The **Failover DB cluster** page appears.
5. Choose **Failover** to confirm the manual failover.

AWS CLI

To fail over a Multi-AZ DB cluster manually, use the AWS CLI command failover-db-cluster.

Example

```
aws rds failover-db-cluster --db-cluster-identifier mymultiazdbcluster
```

RDS API

To fail over a Multi-AZ DB cluster manually, call the Amazon RDS API FailoverDBCluster and specify the `DBClusterIdentifier`.

Determining whether a Multi-AZ DB cluster has failed over

To determine if your Multi-AZ DB cluster has failed over, you can do the following:

- Set up DB event subscriptions to notify you by email or SMS that a failover has been initiated. For more information about events, see Working with Amazon RDS event notification.
- View your DB events by using the Amazon RDS console or API operations.
- View the current state of your Multi-AZ DB cluster by using the Amazon RDS console, the AWS CLI, and the RDS API.

For information on how you can respond to failovers, reduce recovery time, and other best practices for Amazon RDS, see Best practices for Amazon RDS.

Setting the JVM TTL for DNS name lookups

The failover mechanism automatically changes the Domain Name System (DNS) record of the DB instance to point to the reader DB instance. As a result, you need to re-establish any existing connections to your DB instance. In a Java virtual machine (JVM) environment, due to how the Java DNS caching mechanism works, you might need to reconfigure JVM settings.

The JVM caches DNS name lookups. When the JVM resolves a host name to an IP address, it caches the IP address for a specified period of time, known as the *time-to-live* (TTL).

Because AWS resources use DNS name entries that occasionally change, we recommend that you configure your JVM with a TTL value of no more than 60 seconds. Doing this makes sure that when a resource's IP address changes, your application can receive and use the resource's new IP address by requerying the DNS.

On some Java configurations, the JVM default TTL is set so that it never refreshes DNS entries until the JVM is restarted. Thus, if the IP address for an AWS resource changes while your application is still running, it can't use that resource until you manually restart the JVM and the cached IP information is refreshed. In this case, it's crucial to set the JVM's TTL so that it periodically refreshes its cached IP information.

> **Note**
>
> The default TTL can vary according to the version of your JVM and whether a security manager is installed. Many JVMs provide a default TTL less than 60 seconds. If you're using such a JVM and not using a security manager, you can ignore the rest of this topic. For more information on security managers in Oracle, see The security manager in the Oracle documentation.

To modify the JVM's TTL, set the `networkaddress.cache.ttl` property value. Use one of the following methods, depending on your needs:

- To set the property value globally for all applications that use the JVM, set `networkaddress.cache.ttl` in the `$JAVA_HOME/jre/lib/security/java.security` file.

```
networkaddress.cache.ttl=60
```

- To set the property locally for your application only, set `networkaddress.cache.ttl` in your application's initialization code before any network connections are established.

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "60");
```

Setting up PostgreSQL logical replication with Multi-AZ DB clusters for Amazon RDS

By using PostgreSQL logical replication with your Multi-AZ DB cluster, you can replicate and synchronize individual tables rather than the entire database instance. Logical replication uses a publish and subscribe model to replicate changes from a source to one or more recipients. It works by using change records from the PostgreSQL write-ahead log (WAL). For more information, see the section called “Performing logical replication”.

When you create a new logical replication slot on the writer DB instance of a Multi-AZ DB cluster, the slot is asynchronously copied to each reader DB instance in the cluster. The slots on the reader DB instances are continuously synchronized with those on the writer DB instance.

Logical replication is supported for Multi-AZ DB clusters running RDS for PostgreSQL version 14.8-R2 and higher, and 15.3-R2 and higher.

> **Note**
>
> In addition to the native PostgreSQL logical replication feature, Multi-AZ DB clusters running RDS for PostgreSQL also support the `pglogical` extension.

For more information about PostgreSQL logical replication, see Logical replication in the PostgreSQL documentation.

Topics

- Prerequisites
- Setting up logical replication
- Limitations and recommendations

Prerequisites

To configure PostgreSQL logical replication for Multi-AZ DB clusters, you must meet the following prerequisites.

- Your user account must be a member of the `rds_superuser` group and have `rds_superuser` privileges. For more information, see the section called “Understanding PostgreSQL roles and permissions”.

- Your Multi-AZ DB cluster must be associated with a custom DB cluster parameter group so that you can configure the parameter values described in the following procedure. For more information, see the section called "DB cluster parameter groups".

Setting up logical replication

To set up logical replication for a Multi-AZ DB cluster, you enable specific parameters within the associated DB cluster parameter group, then create logical replication slots.

> **Note**
>
> Starting with PostgreSQL version 16, you can use reader DB instances of the Multi-AZ DB cluster for logical replication.

To set up logical replication for an RDS for PostgreSQL Multi-AZ DB cluster

1. Open the custom DB cluster parameter group associated with your RDS for PostgreSQL Multi-AZ DB cluster.
2. In the **Parameters** search field, locate the `rds.logical_replication` static parameter and set its value to 1. This parameter change can increase WAL generation, so enable it only when you're using logical slots.
3. As part of this change, configure the following DB cluster parameters.

   - `max_wal_senders`
   - `max_replication_slots`
   - `max_connections`

   Depending on your expected usage, you might also need to change the values of the following parameters. However, in many cases, the default values are sufficient.

   - `max_logical_replication_workers`
   - `max_sync_workers_per_subscription`
4. Reboot the Multi-AZ DB cluster for the parameter values to take effect. For instructions, see the section called "Rebooting a Multi-AZ DB cluster".

5. Create a logical replication slot on the writer DB instance of the Multi-AZ DB cluster as explained in the section called “Working with logical replication slots”. This process requires that you specify a decoding plugin. Currently, RDS for PostgreSQL supports the `test_decoding`, `wal2json`, and `pgoutput` plugins that ship with PostgreSQL.

   The slot is asynchronously copied to each reader DB instance in the cluster.

6. Verify the state of the slot on all reader DB instances of the Multi-AZ DB cluster. To do so, inspect the `pg_replication_slots` view on all reader DB instances and make sure that the `confirmed_flush_lsn` state is making progress while the application is actively consuming logical changes.

   The following commands demonstrate how to inspect the replication state on the reader DB instances.

```
% psql -h test-postgres-instance-2.abcdefabcdef.us-west-2.rds.amazonaws.com

postgres=> select slot_name, slot_type, confirmed_flush_lsn from
 pg_replication_slots;
  slot_name   | slot_type | confirmed_flush_lsn
--------------+-----------+---------------------
 logical_slot | logical   | 32/D0001700
(1 row)

postgres=> select slot_name, slot_type, confirmed_flush_lsn from
 pg_replication_slots;
  slot_name   | slot_type | confirmed_flush_lsn
--------------+-----------+---------------------
 logical_slot | logical   | 32/D8003628
(1 row)

% psql -h test-postgres-instance-3.abcdefabcdef.us-west-2.rds.amazonaws.com

postgres=> select slot_name, slot_type, confirmed_flush_lsn from
 pg_replication_slots;
  slot_name   | slot_type | confirmed_flush_lsn
--------------+-----------+---------------------
 logical_slot | logical   | 32/D0001700
(1 row)

postgres=> select slot_name, slot_type, confirmed_flush_lsn from
 pg_replication_slots;
  slot_name   | slot_type | confirmed_flush_lsn
```

```
--------------+-----------+--------------------
 logical_slot | logical   | 32/D8003628
(1 row)
```

After you complete your replication tasks, stop the replication process, drop replication slots, and turn off logical replication. To turn off logical replication, modify your DB cluster parameter group and set the value of `rds.logical_replication` back to `0`. Reboot the cluster for the parameter change to take effect.

Limitations and recommendations

The following limitations and recommendations apply to using logical replication with Multi-AZ DB clusters running PostgreSQL version 16:

- You can use only writer DB instances to create or drop logical replication slots. For example, the `CREATE SUBSCRIPTION` command must use the cluster writer endpoint in the host connection string.
- You must use the cluster writer endpoint during any table synchronization or resynchronization. For example, you can use the following commands to resynchronize a newly added table:

  ```
  Postgres=>ALTER SUBSCRIPTION subscription-name CONNECTION host=writer-endpoint
  Postgres=>ALTER SUBSCRIPTION subscription-name REFRESH PUBLICATION
  ```

- You must wait for table synchronization to complete before using the reader DB instances for logical replication. You can use the pg_subscription_rel catalog table to monitor table synchronization. Table synchronization is complete when the `srsubstate` column is set to ready (`r`).
- We recommend using instance endpoints for the logical replication connection once initial table synchronization is complete. The following command reduces load on the writer DB instance by offloading replication to one of the reader DB instances:

  ```
  Postgres=>ALTER SUBSCRIPTION subscription-name CONNECTION host=reader-instance-
  endpoint
  ```

  You can't use the same slot on more than one DB instance at a time. When two or more applications are replicating logical changes from different DB instances in the cluster, some changes might be lost due to a cluster failover or network issue. In these situations, you can use

instance endpoints for logical replication in the host connection string. The other application using the same configuration will show the following error message:

```
replication slot slot_name is already active for PID x providing immediate feedback.
```

- While using the `pglogical` extension, you can only use the cluster writer endpoint. The extension has known limitations that can create unused logical replication slots during table synchronization. Stale replication slots reserve write-ahead log (WAL) files and can lead to disk space problems.

Working with Multi-AZ DB cluster read replicas for Amazon RDS

A DB cluster read replica is a special type of cluster that you create from a source DB instance. After you create a read replica, any updates made to the primary DB instance are asynchronously copied to the Multi-AZ DB cluster read replica. You can reduce the load on your primary DB instance by routing read queries from your applications to the read replica. Using read replicas, you can elastically scale out beyond the capacity constraints of a single DB instance for read-heavy database workloads.

You can also create one or more DB instance read replicas from a Multi-AZ DB cluster. DB instance read replicas let you scale beyond the compute or I/O capacity of the source Multi-AZ DB cluster by directing excess read traffic to the read replicas. Currently, you can't create a Multi-AZ DB cluster read replica from an existing Multi-AZ DB cluster.

When choosing between migrating to a Multi-AZ DB cluster using a read replica or creating a DB instance read replica from a Multi-AZ DB cluster, consider your specific use case and performance requirements.

Migrating to a Multi-AZ DB cluster using a read replica

This approach is ideal when you need to enhance the availability and durability of your database while minimizing downtime. By using a read replica to transition to a Multi-AZ DB cluster, you can ensure continuous operation and data consistency. This method is particularly useful for production environments where maintaining availability and reducing impact on live workloads are critical.

Creating a DB instance read replica from a Multi-AZ DB cluster

This method is suitable when you want to scale read operations or offload read traffic from your primary database instance. By creating a read replica from an existing Multi-AZ DB cluster, you can distribute read-heavy workloads and improve performance without affecting the primary instance's stability.

Choosing the right approach depends on whether your priority is to ensure high availability and durability or to scale read performance. Evaluate your workload characteristics and operational requirements to make an informed decision.

Topics

- Migrating to a Multi-AZ DB cluster using a read replica

- Creating a DB instance read replica from a Multi-AZ DB cluster

Migrating to a Multi-AZ DB cluster using a read replica

To migrate a Single-AZ deployment or Multi-AZ DB instance deployment to a Multi-AZ DB cluster deployment with reduced downtime, you can create a Multi-AZ DB cluster read replica. For the source, you specify the DB instance in the Single-AZ deployment or the primary DB instance in the Multi-AZ DB instance deployment. The DB instance can process write transactions during the migration to a Multi-AZ DB cluster.

Consider the following before you create a Multi-AZ DB cluster read replica:

- The source DB instance must be on a version that supports Multi-AZ DB clusters. For more information, see Supported Regions and DB engines for Multi-AZ DB clusters in Amazon RDS.
- The Multi-AZ DB cluster read replica must be on the same major version as its source, and the same or higher minor version.
- You must turn on automatic backups on the source DB instance by setting the backup retention period to a value other than 0.
- The allocated storage of the source DB instance must be 100 GiB or higher.
- For RDS for MySQL, both the `gtid-mode` and `enforce_gtid_consistency` parameters must be set to `ON` for the source DB instance. You must use a custom parameter group, not the default parameter group. For more information, see the section called “DB parameter groups”.
- An active, long-running transaction can slow the process of creating the read replica. We recommend that you wait for long-running transactions to complete before creating a read replica.
- If you delete the source DB instance for a Multi-AZ DB cluster read replica, the read replica is promoted to a standalone Multi-AZ DB cluster.

Creating and promoting the Multi-AZ DB cluster read replica

You can create and promote a Multi-AZ DB cluster read replica using the AWS Management Console, AWS CLI, or RDS API.

> **Note**
>
> We strongly recommend that you create all read replicas in the same virtual private cloud (VPC) based on Amazon VPC of the source DB instance.

If you create a read replica in a different VPC from the source DB instance, Classless Inter-Domain Routing (CIDR) ranges can overlap between the replica and the Amazon RDS system. CIDR overlap makes the replica unstable, which can negatively impact applications connecting to it. If you receive an error when creating the read replica, choose a different destination DB subnet group. For more information, see Working with a DB instance in a VPC.

Console

To migrate a Single-AZ deployment or Multi-AZ DB instance deployment to a Multi-AZ DB cluster using a read replica, complete the following steps using the AWS Management Console.

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Create the Multi-AZ DB cluster read replica.
   a. In the navigation pane, choose **Databases**.
   b. Choose the DB instance that you want to use as the source for a read replica.
   c. For **Actions**, choose **Create read replica**.
   d. For **Availability and durability**, choose **Multi-AZ DB cluster**.
   e. For **DB instance identifier**, enter a name for the read replica.
   f. For the remaining sections, specify your DB cluster settings. For information about a setting, see Settings for creating Multi-AZ DB clusters.
   g. Choose **Create read replica**.
3. When you are ready, promote the read replica to be a standalone Multi-AZ DB cluster:
   a. Stop any transactions from being written to the source DB instance, and then wait for all updates to be made to the read replica.

      Database updates occur on the read replica after they have occurred on the primary DB instance. This replication lag can vary significantly. Use the `ReplicaLag` metric to determine when all updates have been made to the read replica. For more information about replica lag, see Monitoring read replication.
   b. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

c. In the Amazon RDS console, choose **Databases**.

   The **Databases** pane appears. Each read replica shows **Replica** in the **Role** column.

d. Choose the Multi-AZ DB cluster read replica that you want to promote.

e. For **Actions**, choose **Promote**.

f. On the **Promote read replica** page, enter the backup retention period and the backup window for the newly promoted Multi-AZ DB cluster.

g. When the settings are as you want them, choose **Promote read replica**.

h. Wait for the status of the promoted Multi-AZ DB cluster to be `Available`.

i. Direct your applications to use the promoted Multi-AZ DB cluster.

Optionally, delete the Single-AZ deployment or Multi-AZ DB instance deployment if it is no longer needed. For instructions, see Deleting a DB instance.

AWS CLI

To migrate a Single-AZ deployment or Multi-AZ DB instance deployment to a Multi-AZ DB cluster using a read replica, complete the following steps using the AWS CLI.

1. Create the Multi-AZ DB cluster read replica.

   To create a read replica from the source DB instance, use the AWS CLI command `create-db-cluster`. For `--replication-source-identifier`, specify the Amazon Resource Name (ARN) of the source DB instance.

   For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier mymultiazdbcluster \
  --replication-source-identifier arn:aws:rds:us-
east-2:123456789012:db:mydbinstance
  --engine postgres \
  --db-cluster-instance-class db.m5d.large \
  --storage-type io1 \
  --iops 1000 \
  --db-subnet-group-name defaultvpc \
  --backup-retention-period 1
```

For Windows:

```
aws rds create-db-cluster ^
  --db-cluster-identifier mymultiazdbcluster ^
  --replication-source-identifier arn:aws:rds:us-
east-2:123456789012:db:mydbinstance
  --engine postgres ^
  --db-cluster-instance-class db.m5d.large ^
  --storage-type io1 ^
  --iops 1000 ^
  --db-subnet-group-name defaultvpc ^
  --backup-retention-period 1
```

2. Stop any transactions from being written to the source DB instance, and then wait for all updates to be made to the read replica.

   Database updates occur on the read replica after they have occurred on the primary DB instance. This replication lag can vary significantly. Use the `Replica Lag` metric to determine when all updates have been made to the read replica. For more information about replica lag, see Monitoring read replication.

3. When you are ready, promote the read replica to be a standalone Multi-AZ DB cluster.

   To promote a Multi-AZ DB cluster read replica, use the AWS CLI command `promote-read-replica-db-cluster`. For `--db-cluster-identifier`, specify the identifier of the Multi-AZ DB cluster read replica.

   ```
   aws rds promote-read-replica-db-cluster --db-cluster-identifier mymultiazdbcluster
   ```

4. Wait for the status of the promoted Multi-AZ DB cluster to be `Available`.
5. Direct your applications to use the promoted Multi-AZ DB cluster.

Optionally, delete the Single-AZ deployment or Multi-AZ DB instance deployment if it is no longer needed. For instructions, see Deleting a DB instance.

RDS API

To migrate a Single-AZ deployment or Multi-AZ DB instance deployment to a Multi-AZ DB cluster using a read replica, complete the following steps using the RDS API.

1. Create the Multi-AZ DB cluster read replica.

   To create a Multi-AZ DB cluster read replica, use the `CreateDBCluster` operation with the required parameter `DBClusterIdentifier`. For `ReplicationSourceIdentifier`, specify the Amazon Resource Name (ARN) of the source DB instance.
2. Stop any transactions from being written to the source DB instance, and then wait for all updates to be made to the read replica.

   Database updates occur on the read replica after they have occurred on the primary DB instance. This replication lag can vary significantly. Use the `Replica Lag` metric to determine when all updates have been made to the read replica. For more information about replica lag, see Monitoring read replication.
3. When you are ready, promote read replica to be a standalone Multi-AZ DB cluster.

   To promote a Multi-AZ DB cluster read replica, use the `PromoteReadReplicaDBCluster` operation with the required parameter `DBClusterIdentifier`. Specify the identifier of the Multi-AZ DB cluster read replica.
4. Wait for the status of the promoted Multi-AZ DB cluster to be `Available`.
5. Direct your applications to use the promoted Multi-AZ DB cluster.

Optionally, delete the Single-AZ deployment or Multi-AZ DB instance deployment if it is no longer needed. For instructions, see Deleting a DB instance.

Limitations for creating a Multi-AZ DB cluster read replica

The following limitations apply to creating a Multi-AZ DB cluster read replica from a Single-AZ deployment or Multi-AZ DB instance deployment.

- You can't create a Multi-AZ DB cluster read replica in an AWS account that is different from the AWS account that owns the source DB instance.
- You can't create a Multi-AZ DB cluster read replica in a different AWS Region from the source DB instance.
- You can't recover a Multi-AZ DB cluster read replica to a point in time.
- Storage encryption must have the same settings on the source DB instance and Multi-AZ DB cluster.
- If the source DB instance is encrypted, the Multi-AZ DB cluster read replica must be encrypted using the same KMS key.

- If the source DB instance uses General Purpose SSD (gp3) storage and has less than 400 GiB of allocated storage, you can't modify the provisioned IOPS for the Multi-AZ DB cluster read replica.
- To perform a minor version upgrade on the source DB instance, you must first perform the minor version upgrade on the Multi-AZ DB cluster read replica.
- When you perform a minor version upgrade on an RDS for PostgreSQL Multi-AZ DB cluster read replica, the reader DB instance doesn't switch to the writer DB instance after the upgrade. Therefore, your DB cluster might experience downtime while Amazon RDS upgrades the writer instance.
- You can't perform a major version upgrade on a Multi-AZ DB cluster read replica.
- You can perform a major version upgrade on the source DB instance of a Multi-AZ DB cluster read replica, but replication to the read replica stops and can't be restarted.
- The Multi-AZ DB cluster read replica doesn't support cascading read replicas.
- For RDS for PostgreSQL, Multi-AZ DB cluster read replicas can't fail over.

Creating a DB instance read replica from a Multi-AZ DB cluster

You can create a DB instance read replica from a Multi-AZ DB cluster in order to scale beyond the compute or I/O capacity of the cluster for read-heavy database workloads. You can direct this excess read traffic to one or more DB instance read replicas. You can also use read replicas to migrate from a Multi-AZ DB cluster to a DB instance.

To create a read replica, specify a Multi-AZ DB cluster as the replication source. One of the reader instances of the Multi-AZ DB cluster is always the source of replication, not the writer instance. This condition ensures that the replica is always in sync with the source cluster, even in cases of failover.

Topics

- Comparing reader DB instances and DB instance read replicas
- Considerations
- Creating a DB instance read replica
- Promoting the DB instance read replica
- Limitations for creating a DB instance read replica from a Multi-AZ DB cluster

Comparing reader DB instances and DB instance read replicas

A *DB instance read replica* of a Multi-AZ DB cluster is different than the *reader DB instances* of the Multi-AZ DB cluster in the following ways:

- The reader DB instances act as automatic failover targets, while DB instance read replicas do not.
- Reader DB instances must acknowledge a change from the writer DB instance before the change can be committed. For DB instance read replicas, however, updates are asynchronously copied to the read replica without requiring acknowledgement.
- Reader DB instances always share the same instance class, storage type, and engine version as the writer DB instance of the Multi-AZ DB cluster. DB instance read replicas, however, don't necessarily have to share the same configurations as the source cluster.
- You can promote a DB instance read replica to a standalone DB instance. You can't promote a reader DB instance of a Multi-AZ DB cluster to a standalone instance.
- The reader endpoint only routes requests to the reader DB instances of the Multi-AZ DB cluster. It never routes requests to a DB instance read replica.

For more information about reader and writer DB instances, see the section called "Multi-AZ DB cluster architecture".

Considerations

Consider the following before you create a DB instance read replica from a Multi-AZ DB cluster:

- When you create the DB instance read replica, it must be on the same major version as its source cluster, and the same or higher minor version. After you create it, you can optionally upgrade the read replica to a higher minor version than the source cluster.
- When you create the DB instance read replica, the allocated storage must be the same as the allocated storage of the source Multi-AZ DB cluster. You can change the allocated storage after the read replica is created.
- For RDS for MySQL, the `gtid-mode` parameter must be set to `ON` for the source Multi-AZ DB cluster. For more information, see the section called "DB cluster parameter groups".
- An active, long-running transaction can slow the process of creating the read replica. We recommend that you wait for long-running transactions to complete before creating a read replica.
- If you delete the source Multi-AZ DB cluster for a DB instance read replica, any read replicas that it's writing to are promoted to standalone DB instances.

Creating a DB instance read replica

You can create a DB instance read replica from a Multi-AZ DB cluster using the AWS Management Console, AWS CLI, or RDS API.

> **Note**
>
> We strongly recommend that you create all read replicas in the same virtual private cloud (VPC) based on Amazon VPC of the source Multi-AZ DB cluster.
> If you create a read replica in a different VPC from the source Multi-AZ DB cluster, Classless Inter-Domain Routing (CIDR) ranges can overlap between the replica and the RDS system. CIDR overlap makes the replica unstable, which can negatively impact applications connecting to it. If you receive an error when creating the read replica, choose a different destination DB subnet group. For more information, see the section called "Working with a DB instance in a VPC".

Console

To create a DB instance read replica from a Multi-AZ DB cluster, complete the following steps using the AWS Management Console.

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the Multi-AZ DB cluster that you want to use as the source for a read replica.
4. For **Actions**, choose **Create read replica**.
5. For **Replica source**, make sure that the correct Multi-AZ DB cluster is selected.
6. For **DB identifier**, enter a name for the read replica.
7. For the remaining sections, specify your DB instance settings. For information about a setting, see the section called "Available settings".

   > **Note**
   >
   > The allocated storage for the DB instance read replica must be the same as the allocated storage for the source Multi-AZ DB cluster.

8. Choose **Create read replica**.

AWS CLI

To create a DB instance read replica from a Multi-AZ DB cluster, use the AWS CLI command `create-db-instance-read-replica`. For `--source-db-cluster-identifier`, specify the identifier of the Multi-AZ DB cluster.

For Linux, macOS, or Unix:

```
aws rds create-db-instance-read-replica \
  --db-instance-identifier myreadreplica \
  --source-db-cluster-identifier mymultiazdbcluster
```

For Windows:

```
aws rds create-db-instance-read-replica ^
  --db-instance-identifier myreadreplica ^
  --source-db-cluster-identifier mymultiazdbcluster
```

RDS API

To create a DB instance read replica from a Multi-AZ DB cluster, use the `CreateDBInstanceReadReplica` operation.

Promoting the DB instance read replica

If you no longer need the DB instance read replica, you can promote it into a standalone DB instance. When you promote a read replica, the DB instance is rebooted before it becomes available. For instructions, see the section called “Promoting a read replica”.

If you're using the read replica to migrate a Multi-AZ DB cluster deployment to a Single-AZ or Multi-AZ DB instance deployment, make sure to stop any transactions that are being written to the source DB cluster. Then, wait for all updates to be made to the read replica. Database updates occur on the read replica after they occur on one of the reader DB instances of the Multi-AZ DB cluster. This replication lag can vary significantly. Use the `ReplicaLag` metric to determine when all updates have been made to the read replica. For more information about replica lag, see the section called “Monitoring read replication”.

After you promote the read replica, wait for the status of the promoted DB instance to be `Available` before you direct your applications to use the promoted DB instance. Optionally,

delete the Multi-AZ DB cluster deployment if you no longer need it. For instructions, see the section called “Deleting a Multi-AZ DB cluster”.

Limitations for creating a DB instance read replica from a Multi-AZ DB cluster

The following limitations apply to creating a DB instance read replica from a Multi-AZ DB cluster deployment.

- You can't create a DB instance read replica in an AWS account that's different from the AWS account that owns the source Multi-AZ DB cluster.
- You can't create a DB instance read replica in a different AWS Region from the source Multi-AZ DB cluster.
- You can't recover a DB instance read replica to a point in time.
- Storage encryption must have the same settings on the source Multi-AZ DB cluster and DB instance read replica.
- If the source Multi-AZ DB cluster is encrypted, the DB instance read replica must be encrypted using the same KMS key.
- To perform a minor version upgrade on the source Multi-AZ DB cluster, you must first perform the minor version upgrade on the DB instance read replica.
- The DB instance read replica doesn't support cascading read replicas.
- For RDS for PostgreSQL, the source Multi-AZ DB cluster must be running PostgreSQL version 13.11, 14.8, or 15.2.R2 or higher in order to create a DB instance read replica.
- You can perform a major version upgrade on the source Multi-AZ DB cluster of a DB instance read replica, but replication to the read replica stops and can't be restarted.

Setting up external replication from Multi-AZ DB clusters for Amazon RDS

You can set up replication between a Multi-AZ DB cluster and a database that is external to Amazon RDS.

External replication allows Multi-AZ DB clusters to replicate data between an RDS DB instance and an external database, either on-premises or in another cloud environment. It's beneficial for disaster recovery, data migration, and maintaining consistency between systems in different locations. The section covers the prerequisites for setting up replication, how to configure the process, and key considerations like replication latency, bandwidth, and compatibility with different database engines.

RDS for MySQL

To set up external replication for an RDS for MySQL Multi-AZ DB cluster, you must retain binary log files on the DB instances within the cluster for long enough to ensure that the changes are applied to the replica before Amazon RDS deletes the binlog file. To do so, configure binary log retention by calling the `mysql.rds_set_configuration` stored procedure and specifying the `binlog retention hours` parameter. For more information, see the section called “binlog retention hours”.

The default value for `binlog retention hours` is `NULL`, which means that binary logs aren't retained (0 hours). If you want to set up external replication for a Multi-AZ DB cluster, you must set the parameter to a value other than `NULL`.

You can only configure binary log retention from the writer DB instance of the Multi-AZ DB cluster, and the setting is propagated to all reader DB instances asynchronously.

In addition, we highly recommend enabling GTID-based replication on your external replica. Then, if one of the DB instances fails, you can resume replication from another healthy DB instance within the cluster. For more information, see Replication with Global Transaction Identifiers in the MySQL documentation.

RDS for PostgreSQL

To set up external replication for an RDS for PostgreSQL Multi-AZ DB cluster, you must enable logical replication. For instructions, see the section called “PostgreSQL logical replication with Multi-AZ DB clusters”.

Deleting a Multi-AZ DB cluster for Amazon RDS

You can delete a DB Multi-AZ DB cluster using the AWS Management Console, the AWS CLI, or the RDS API.

The time required to delete a Multi-AZ DB cluster can vary depending on the following factors:

- Thee backup retention period (that is, how many backups to delete).
- How much data is deleted.
- Whether a final snapshot is taken.

Deletion protection must be disabled on the Multi-AZ DB cluster before you can delete it. For more information, see the section called “Prerequisites for deleting a DB instance”. You can disable deletion protection by modifying the Multi-AZ DB cluster. For more information, see the section called “Modifying a Multi-AZ DB cluster”.

Console

To delete a Multi-AZ DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the Multi-AZ DB cluster that you want to delete.
3. For **Actions**, choose **Delete**.
4. Choose **Create final snapshot?** to create a final DB snapshot for the Multi-AZ DB cluster.

   If you create a final snapshot, enter a name for **Final snapshot name**.
5. Choose **Retain automated backups** to retain automated backups.
6. Enter **`delete me`** in the box.
7. Choose **Delete**.

AWS CLI

To delete a Multi-AZ DB cluster by using the AWS CLI, call the delete-db-cluster command with the following options:

- `--db-cluster-identifier`
- `--final-db-snapshot-identifier` or `--skip-final-snapshot`

Example With a final snapshot

For Linux, macOS, or Unix:

```
aws rds delete-db-cluster \
    --db-cluster-identifier mymultiazdbcluster \
    --final-db-snapshot-identifier mymultiazdbclusterfinalsnapshot
```

For Windows:

```
aws rds delete-db-cluster ^
    --db-cluster-identifier mymultiazdbcluster ^
    --final-db-snapshot-identifier mymultiazdbclusterfinalsnapshot
```

Example With no final snapshot

For Linux, macOS, or Unix:

```
aws rds delete-db-cluster \
    --db-cluster-identifier mymultiazdbcluster \
    --skip-final-snapshot
```

For Windows:

```
aws rds delete-db-cluster ^
    --db-cluster-identifier mymultiazdbcluster ^
    --skip-final-snapshot
```

RDS API

To delete a Multi-AZ DB cluster by using the Amazon RDS API, call the DeleteDBCluster operation with the following parameters:

- `DBClusterIdentifier`
- `FinalDBSnapshotIdentifier` or `SkipFinalSnapshot`

Limitations of Multi-AZ DB clusters for Amazon RDS

A Multi-AZ DB cluster has a writer DB instance and two reader DB instances in three separate Availability Zones. Multi-AZ DB clusters provide high availability, increased capacity for read workloads, and lower latency when compared to Multi-AZ deployments. For more information about Multi-AZ DB clusters, see Multi-AZ DB cluster deployments for Amazon RDS.

The following limitations apply to Multi-AZ DB clusters.

- Multi-AZ DB clusters don't support the following features:
  - IPv6 connections (dual-stack mode)
  - Cross-Region automated backups
  - Kerberos authentication
  - Modifying the port. As an alternative, you can restore a Multi-AZ DB cluster to a point in time and specify a different port.
  - Option groups
  - Point-in-time-recovery (PITR) for deleted clusters
  - Storage autoscaling by setting the maximum allocated storage. As an alternative, you can scale storage manually.
  - Stopping and starting the Multi-AZ DB cluster
  - Copying a snapshot of a Multi-AZ DB cluster
  - Encrypting an unencrypted Multi-AZ DB cluster
- RDS for MySQL Multi-AZ DB clusters support only the following system stored procedures:
  - `mysql.rds_rotate_general_log`
  - `mysql.rds_rotate_slow_log`
  - `mysql.rds_show_configuration`
  - `mysql.rds_set_external_master_with_auto_position`
  - `mysql.rds_set_configuration`
- RDS for PostgreSQL Multi-AZ DB clusters don't support the following extensions: `aws_s3` and `pg_transport`.
- RDS for PostgreSQL Multi-AZ DB clusters don't support using a custom DNS server for outbound network access.

Amazon RDS Extended Support with Amazon RDS

RDS Extended Support allows you to continue running a database on a major engine version past the RDS end of standard support date for an additional cost.

You can only enroll a database in RDS Extended Support by enabling RDS Extended Support when you first create or restore a DB instance. You can't update your RDS Extended Support enrollment status on existing DB instances unless you are restoring them.

If you enabled RDS Extended Support during the creation or restoration of a DB instance, then after the RDS end of standard support date, Amazon RDS will automatically enroll the DB instance in RDS Extended Support. Automatic enrollment into RDS Extended Support doesn't change the database engine and doesn't impact the uptime or performance of your DB instance.

RDS Extended Support provides the following updates and technical support:

- Security updates for critical and high CVEs for your DB instance or DB cluster, including the database engine
- Bug fixes and patches for critical issues
- The ability to open support cases and receive troubleshooting help within the standard Amazon RDS service level agreement

This paid offering gives you more time to upgrade to a supported major engine version. For example, the RDS end of standard support date for RDS for MySQL version 5.7 is February 29, 2024. However, you aren't ready to manually upgrade to RDS for MySQL version 8.0 before that date. In this case, Amazon RDS automatically enrolls your databases in RDS Extended Support on February 29, 2024, and you can continue to run RDS for MySQL version 5.7. Starting March 1, 2024, Amazon RDS automatically charges you for RDS Extended Support.

RDS Extended Support is available for up to 3 years past the RDS end of standard support date for a major engine version. After this time, if you haven't upgraded your major engine version to a supported version, then Amazon RDS will automatically upgrade your major engine version. We recommend that you upgrade to a supported major engine version as soon as possible.

For more information about the RDS end of standard support dates and the RDS end of Extended Support dates, see Supported MySQL major versions on Amazon RDS and Release calendar for Amazon RDS for PostgreSQL.

Topics

- Overview of Amazon RDS Extended Support
- Amazon RDS Extended Support charges
- Versions with Amazon RDS Extended Support
- Amazon RDS and customer responsibilities with Amazon RDS Extended Support
- Creating a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support
- Viewing the enrollment of your DB instances or Multi-AZ DB clusters in Amazon RDS Extended Support
- Viewing support dates for engine versions in Amazon RDS Extended Support
- Restoring a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support

Overview of Amazon RDS Extended Support

After the RDS end of standard support date, if you didn't disable RDS Extended Support during the creation or restoration of your DB instances, then Amazon RDS will automatically enroll them in RDS Extended Support. Amazon RDS automatically upgrades your DB instance to the last minor version released before the RDS end of standard support date, if you aren't already running that version. Amazon RDS won't upgrade your minor version until *after* the RDS end of standard support date for your major engine version.

You can create new databases with major engine versions that have reached the RDS end of standard support date. RDS automatically enrolls these new databases in RDS Extended Support and charges you for this offering.

If you upgrade to an engine that's still under RDS standard support *before* the RDS end of standard support date, Amazon RDS won't enroll your engine in RDS Extended Support.

If you attempt to restore a snapshot of a database compatible with an engine that's past the RDS end of standard support date but isn't enrolled in RDS Extended Support, then Amazon RDS will attempt to upgrade the snapshot to be compatible with the latest engine version that is still under RDS standard support. If the restore fails, then Amazon RDS will automatically enroll your engine in RDS Extended Support with a version that's compatible with the snapshot.

You can end enrollment in RDS Extended Support at any time. To end enrollment, upgrade each enrolled engine to a newer engine version that's still under RDS standard support. The end of RDS

Extended Support enrollment will be effective the day that you complete an upgrade to a newer engine version that's still under RDS standard support.

For more information about the RDS end of standard support dates and the RDS end of Extended Support dates, see Supported MySQL major versions on Amazon RDS and Release calendar for Amazon RDS for PostgreSQL.

Amazon RDS Extended Support charges

You will incur charges for all engines enrolled in RDS Extended Support beginning the day after the RDS end of standard support date. For the RDS end of standard support date, see Major versions and Release calendar for Amazon RDS for PostgreSQL. RDS Extended Support charges apply to standby instances in Multi-AZ deployments.

The additional charge for RDS Extended Support automatically stops when you take one of the following actions:

- Upgrade to an engine version that's covered under standard support.
- Delete the database that's running a major version past the RDS end of standard support date.

The charges will restart if your target engine version enters RDS Extended Support in the future.

For example, RDS for PostgreSQL 11 enters Extended Support on March 1, 2024, but charges don't start until April 1, 2024. You upgrade your RDS for PostgreSQL 11 database to RDS for PostgreSQL 12 on April 30, 2024. You will only be charged for 30 days of Extended Support on RDS for PostgreSQL 11. You continue running RDS for PostgreSQL 12 on this DB instance past the RDS end of standard support date of February 28, 2025. Your database will again incur RDS Extended Support charges starting on March 1, 2025.

For more information, see Amazon RDS for MySQL pricing and Amazon RDS for PostgreSQL pricing.

Avoiding charges for Amazon RDS Extended Support

You can avoid being charged for RDS Extended Support by preventing RDS from creating or restoring a DB instance or a Multi-AZ DB cluster past the RDS end of standard support date. To do this, use the AWS CLI or the RDS API.

In the AWS CLI, specify `open-source-rds-extended-support-disabled` for the `--engine-lifecycle-support` option. In the RDS API, specify `open-source-rds-extended-support-disabled` for the `LifeCycleSupport` parameter. For more information, see Creating a DB instance or a Multi-AZ DB cluster or Restoring a DB instance or a Multi-AZ DB cluster.

Versions with Amazon RDS Extended Support

RDS Extended Support is only available for major versions. It isn't available for minor versions.

RDS Extended Support is available for RDS for MySQL and for RDS for PostgreSQL. For more information, see Major versions and Release calendar for Amazon RDS for PostgreSQL in the *Amazon RDS for PostgreSQL Release Notes*.

You can also view information about support dates for engine versions by using the AWS CLI or the RDS API. For more information, see Viewing support dates for engine versions in Amazon RDS Extended Support.

Amazon RDS Extended Support version naming

Amazon RDS will release new minor versions with fixes and CVE patches for engines on RDS Extended Support. For more information, see Amazon RDS Extended Support versions for RDS for MySQL and Amazon RDS Extended Support updates for RDS for PostgreSQL in the *Amazon RDS for PostgreSQL Release Notes*.

The names of these minor releases will be in the form *major.minor-RDS.YYYYMMDD.patch.YYYYMMDD*, for example, 5.7.44-RDS.20240208.R2.20240210 (for RDS for MySQL) or 11.22-RDS.20240208.R2.20240210 (for RDS for PostgreSQL).

major

For MySQL, the major version number is both the integer and the first fractional part of the version number, for example, 8.0. A major version upgrade increases the major part of the version number. For example, an upgrade from 5.7.44 to 8.0.33 is a major version upgrade, where *5.7* and *8.0* are the major version numbers.

For PostgreSQL, the major version number is the integer, for example, 11.

minor-RDS.YYYYMMDD

For MySQL, the minor version number is the third part of the version number, for example, the 44-RDS.20240208 in 5.7.44-RDS.20240208.

For PostgeSQL, the minor version number is the second part of version number, for example, the 22-RDS.20240208in 11.22-RDS.20240208.

The date is when Amazon RDS created the Amazon RDS minor version.

patch

The patch version is what follows the date when Amazon RDS created the Amazon RDS minor version, for example, the R2 in 5.7.44-RDS.20240208.R2 or 11.22-RDS.20240208.R2.

An Amazon RDS patch version includes important bug fixes added to an Amazon RDS minor version after its release.

YYYYMMDD

The date is when Amazon RDS created the patch version, for example, the 20240210 in 5.7.44-RDS.20240208.R2.20240210 or 11.22-RDS.20240208.R2.20240210.

An Amazon RDS dated version is a security patch that includes important security fixes added to a minor version after its release. It doesn't include any fixes that might change an engine's behavior.

Amazon RDS and customer responsibilities with Amazon RDS Extended Support

The following content describes the responsibilities of Amazon RDS and your responsibilities with RDS Extended Support.

Topics

- Amazon RDS responsibilities
- Your responsibilities

Amazon RDS responsibilities

After the RDS end of standard support date, Amazon RDS will supply patches, bug fixes, and upgrades for engines that are enrolled in RDS Extended Support. This will occur for up to 3 years, or until you stop using the engines, whichever happens first.

The patches will be for Critical and High CVEs as defined by the National Vulnerability Database (NVD) CVSS severity ratings. For more information, see Vulnerability Metrics.

Your responsibilities

You're responsible for applying the patches, bug fixes, and upgrades given for DB instances or Multi-AZ DB clusters enrolled in RDS Extended Support. Amazon RDS reserves the right to change, replace, or withdraw such patches, bug fixes, and upgrades at any time. If a patch is necessary to address security or critical stability issues, Amazon RDS reserves the right to update your DB instances or Multi-AZ DB clusters with the patch, or to require that you install the patch.

You're also responsible for upgrading your engine to a newer engine version *before* the RDS end of Extended Support date. The RDS end of Extended Support date is typically 3 years after the RDS standard support date. For the RDS end of Extended Support date for your database major engine version, see Major versions and Release calendar for Amazon RDS for PostgreSQL.

If you don't upgrade your engine, then after the RDS end of Extended Support date, Amazon RDS will attempt to upgrade your engine to a newer engine version that's supported under RDS standard support. If the upgrade fails, then Amazon RDS reserves the right to delete the DB instance or Multi-AZ DB cluster that's running the engine past the RDS end of standard support date. However, before doing so, Amazon RDS will preserve your data from that engine.

Creating a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support

When you create a DB instance or a Multi-AZ DB cluster, select **Enable RDS Extended Support** in the console, or use the Extended Support option in the AWS CLI or the parameter in the RDS API. When you enroll a DB instance or a Multi-AZ DB cluster in Amazon RDS Extended Support, it is permanently enrolled in RDS Extended Support for the life of the DB instance or Multi-AZ DB cluster.

If you use the console, you must select **Enable RDS Extended Support**. The setting isn't selected by default.

If you use the AWS CLI or the RDS API and don't specify the RDS Extended Support setting, Amazon RDS defaults to enabling RDS Extended Support. When you automate by using CloudFormation or other services, this default behavior maintains the availability of your database past the RDS end of standard support date.

You can prevent enrollment in RDS Extended Support by using the AWS CLI or the RDS API to create a DB instance or a Multi-AZ DB cluster.

Topics

- RDS Extended Support behavior
- Considerations for RDS Extended Support
- Create a DB instance or a Multi-AZ DB cluster with RDS Extended Support

RDS Extended Support behavior

The following table summarizes what happens when a major engine version reaches the RDS end of standard support.

RDS Extended Support status*	Behavior
Enabled	Amazon RDS charges you for RDS Extended Support.
Disabled	Amazon RDS upgrades your DB instance or Multi-AZ DB cluster to a supported engine version. This upgrade takes place on or shortly after the RDS end of standard support date.

* In the RDS console, the RDS Extended Support status appears as Yes or No. In the AWS CLI or RDS API, the RDS Extended Support status appears as `open-source-rds-extended-support` or `open-source-rds-extended-support-disabled`.

Considerations for RDS Extended Support

Before creating a DB instance or a Multi-AZ DB cluster, consider the following items:

- *After* the RDS end of standard support date has passed, you can prevent the creation of a new DB instance or a new Multi-AZ DB cluster and avoid RDS Extended Support charges. To do this, use the AWS CLI or the RDS API. In the AWS CLI, specify `open-source-rds-extended-support-disabled` for the `--engine-lifecycle-support` option. In the RDS API, specify `open-source-rds-extended-support-disabled` for the `LifeCycleSupport` parameter. If you specify `open-source-rds-extended-support-disabled` and the RDS end of standard support date has passed, creating a DB instance or a Multi-AZ DB cluster will always fail.

- RDS Extended Support is set at the cluster level. Members of a cluster will always have the same setting for RDS Extended Support in the RDS console, `--engine-lifecycle-support` in the AWS CLI, and `EngineLifecycleSupport` in the RDS API.

For more information, see MySQL versions and Release calendars for Amazon RDS for PostgreSQL.

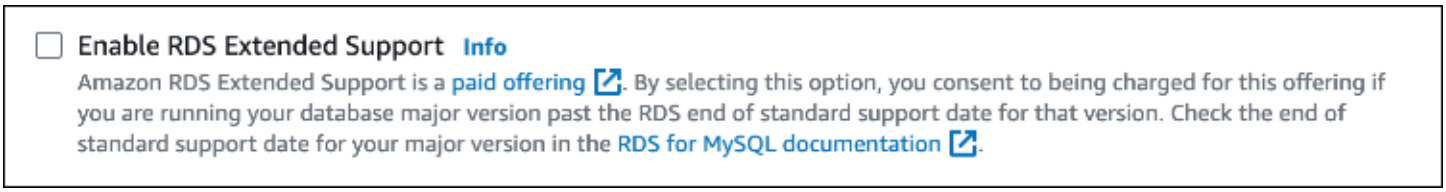
Create a DB instance or a Multi-AZ DB cluster with RDS Extended Support

You can create a DB instance or a Multi-AZ DB cluster with an RDS Extended Support version using the AWS Management Console, the AWS CLI, or the RDS API.

Console

When you create a DB instance or a Multi-AZ DB cluster, in the **Engine options** section, select **Enable RDS Extended Support**. This setting isn't selected by default.

The following image shows the **Enable RDS Extended Support** setting:



AWS CLI

When you run the create-db-instance or create-db-cluster (Multi-AZ DB cluster) AWS CLI command, select RDS Extended Support by specifying `open-source-rds-extended-support` for the `--engine-lifecycle-support` option. By default, this option is set to `open-source-rds-extended-support`.

To prevent the creation of a new DB instance or a Multi-AZ DB cluster after the RDS end of standard support date, specify `open-source-rds-extended-support-disabled` for the `--engine-lifecycle-support` option. By doing so, you will avoid any associated RDS Extended Support charges.

RDS API

When you use the CreateDBInstance or CreateDBCluster (Multi-AZ DB cluster) Amazon RDS API operation, select RDS Extended Support by setting the `EngineLifecycleSupport` parameter

to `open-source-rds-extended-support`. By default, this parameter is set to `open-source-rds-extended-support`.

To prevent the creation of a new DB instance or a Multi-AZ DB cluster after the RDS end of standard support date, specify `open-source-rds-extended-support-disabled` for the `EngineLifecycleSupport` parameter. By doing so, you will avoid any associated RDS Extended Support charges.

For more information, see the following topics:

- To create a DB instance, follow the instructions for your DB engine in Creating an Amazon RDS DB instance.
- To create a Multi-AZ DB cluster, follow the instructions for your DB engine in Creating a Multi-AZ DB cluster for Amazon RDS.

Viewing the enrollment of your DB instances or Multi-AZ DB clusters in Amazon RDS Extended Support

You can view the enrollment of your DB instances or Multi-AZ DB clusters in RDS Extended Support using the AWS Management Console, the AWS CLI, or the RDS API.

> **Note**
>
> The **RDS Extended Support** column in the console, the `--engine-lifecycle-support` option in the AWS CLI, and the `EngineLifecycleSupport` parameter in the RDS API only indicate enrollment in RDS Extended Support. Charges for RDS Extended Support only start when your DB engine version has reached the RDS end of standard support. For more information, see Major versions and Release calendar for Amazon RDS for PostgreSQL in the *Amazon RDS for PostgreSQL Release Notes*.
> For example, you have an RDS for MySQL 5.7 database that is enrolled in RDS Extended Support. On March 1, 2024, Amazon RDS started charging you for RDS Extended Support for this database. On July 31, 2024, you upgraded this database to RDS for MySQL 8.0. The RDS Extended Support status for this database remains enabled. However, the RDS Extended Support charges for this database stopped because RDS for MySQL 8.0 hadn't reached RDS end of standard support yet. Amazon RDS won't charge you for RDS Extended

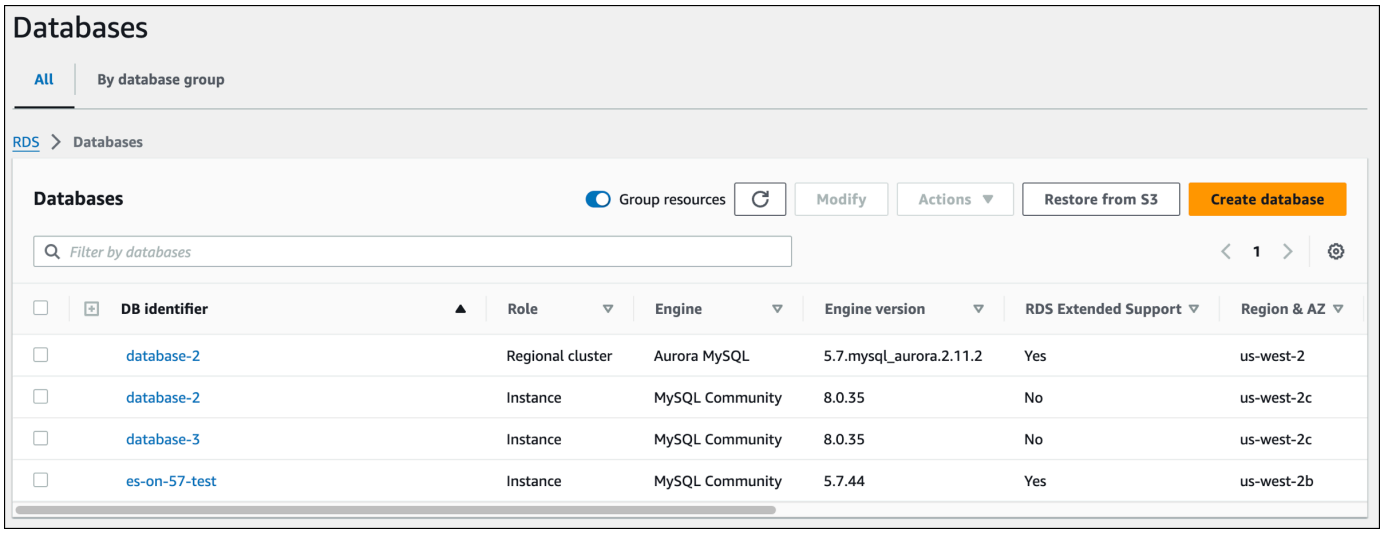
Support for this database until August 1, 2026, which is when RDS standard support ends for RDS for MySQL 8.0.

Console

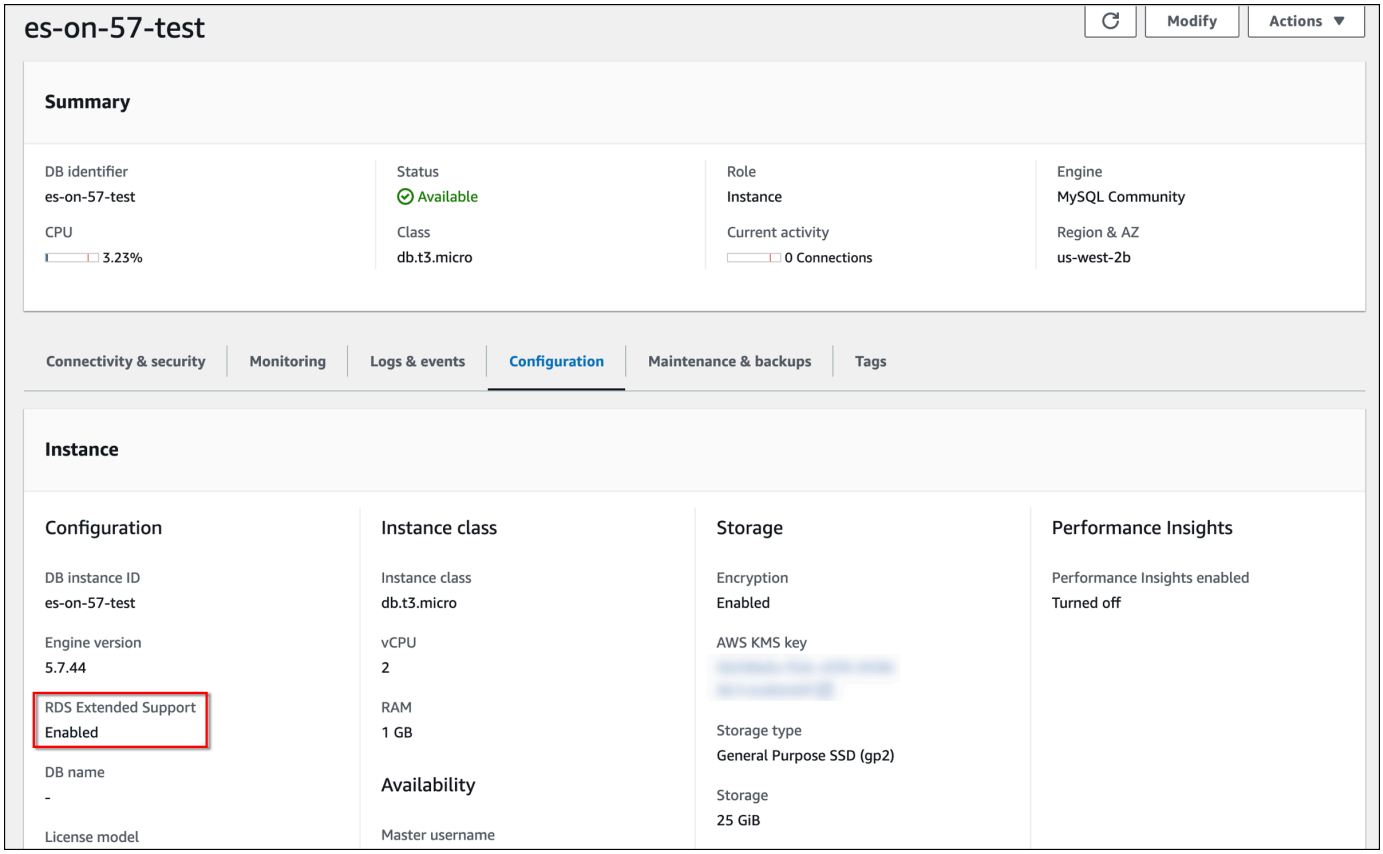
To view the enrollment of your DB instances or Multi-AZ DB clusters in RDS Extended Support

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**. The value under **RDS Extended Support** indicates if a DB instance or Multi-AZ DB cluster is enrolled in RDS Extended Support. If no value appears, then RDS Extended Support isn't available for your database.

> **Tip**
>
> If the **RDS Extended Support** column doesn't appear, choose the **Preferences** icon, and then turn on **RDS Extended Support**.



3. You can also view the enrollment on the **Configuration** tab for each database. Choose a database under **DB identifier**. On the **Configuration** tab, look under **Extended Support** to see if the database is enrolled or not.



AWS CLI

To view the enrollment of your databases in RDS Extended Support by using the AWS CLI, run the describe-db-instances or describe-db-clusters (Multi-AZ DB clusters) command.

If RDS Extended Support is available for a database, then the response includes the parameter `EngineLifecycleSupport`. The value `open-source-rds-extended-support` indicates that a DB instance or Multi-AZ DB cluster is enrolled in RDS Extended Support. The value `open-source-rds-extended-support-disabled` indicates that enrollment of the DB instance or Multi-AZ DB cluster in RDS Extended Support was disabled.

Example

The following command returns information for all of your DB instances:

```
aws rds describe-db-instances
```

The following response shows that a PostgreSQL engine running on the DB instance `database-1` is enrolled in RDS Extended Support:

```
{
    "DBInstanceIdentifier": "database-1",
    "DBInstanceClass": "db.t3.large",
    "Engine": "postgres",
    ...
    "EngineLifecycleSupport": "open-source-rds-extended-support"
}
```

RDS API

To view the enrollment of your databases in RDS Extended Support by using the Amazon RDS API, use the DescribeDBInstances or DescribeDBClusters operation.

If RDS Extended Support is available for a database, then the response includes the parameter `EngineLifecycleSupport`. The value `open-source-rds-extended-support` indicates that a DB instance or Multi-AZ DB cluster is enrolled in RDS Extended Support. The value `open-source-rds-extended-support-disabled` indicates that enrollment of the DB instance or Multi-AZ DB cluster in RDS Extended Support was disabled.

Viewing support dates for engine versions in Amazon RDS Extended Support

You can view information about support dates for engine versions for your DB instances or Multi-AZ DB clusters in Amazon RDS Extended Support by using the AWS CLI or the RDS API. This information can help you plan for upgrades.

AWS CLI commands and RDS API operations return start and end dates for RDS standard support and RDS Extended Support. These dates can also be found in the major engine version tables. For more information, see Supported MySQL major versions on Amazon RDS and Release calendar for Amazon RDS for PostgreSQL in the *Amazon RDS for PostgreSQL Release Notes*.

AWS CLI

To view the start and end dates for RDS standard support and RDS Extended Support for your major engine versions by using the AWS CLI, run the describe-db-major-engine-versions command.

This command returns the following relevant parameters for open source engines (MariaDB, MySQL, and PostgreSQL). It doesn't return these parameters for commercial engines (Db2, SQL Server, and Oracle).

- `SupportedEngineLifecycles` – This parameter is an array of objects that include `LifecycleSupportName`, `LifecycleSupportStartDate`, and `LifecycleSupportEndDate`.
- `LifecycleSupportName` – This parameter indicates the type of support the engine version is in: RDS standard support (`open-source-rds-standard-support`) or RDS Extended Support (`open-source-rds-extended-support`). For MariaDB, this parameter only returns RDS standard support (`open-source-rds-standard-support`).
- `LifecycleSupportStartDate` – This parameter lists the start date for either RDS standard support or RDS Extended Support for the major engine version, depending on the value of `LifecycleSupportName`.
- `LifecycleSupportEndDate` – This parameter lists the end date for either RDS standard support or RDS Extended Support for the major engine version, depending on the value of `LifecycleSupportName`.

Example

The response example shows the following information:

- The start and end dates for the supported engine lifecycles `open-source-rds-standard-support` and `open-source-rds-extended-support` for MySQL 5.7. RDS Extended Support is available for MySQL 5.7.
- The start and end dates for the supported engine lifecycle `open-source-rds-standard-support` for MariaDB 10.6. RDS Extended Support isn't available for MariaDB 10.6.
- No information about supported engine lifecycles for SQL Server Enterprise Edition 13 because SQL Server is a commercial engine.

```
{
    "DBMajorEngineVersions": [
        {
            "Engine": "mysql",
            "MajorEngineVersion": "5.7",
            "SupportedEngineLifecycles": [
                {
                    "LifecycleSupportName": "open-source-rds-standard-support",
                    "LifecycleSupportStartDate": "2016-02-22T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2024-02-29T23:59:59.999000+00:00"
                },
```

```
                {
                    "LifecycleSupportName": "open-source-rds-extended-support",
                    "LifecycleSupportStartDate": "2024-03-01T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2027-02-28T23:59:59.999000+00:00"
                }
            ]
        },
        {
            "Engine": "mariadb",
            "MajorEngineVersion": "10.6",
            "SupportedEngineLifecycles": [
                {
                    "LifecycleSupportName": "open-source-rds-standard-support",
                    "LifecycleSupportStartDate": "2022-02-03T00:00:00+00:00",
                    "LifecycleSupportEndDate": "2026-07-31T23:59:59.999000+00:00"
                }
            ]
        },
        {
            "Engine": "sqlserver-ee",
            "MajorEngineVersion": "13.00"
        },
        ...
    ]
}
```

RDS API

To view the start and end dates for RDS standard support and RDS Extended Support for your major engine versions by using the RDS API, use the DescribeDBMajorEngineVersions operation.

If RDS Extended Support is available for an engine version, then the response includes the parameter `SupportedEngineLifeCycles` as an array with two objects. One object includes the start and end dates for RDS standard support. The second object includes the start and end dates for RDS Extended Support.

If RDS Extended Support isn't available for an open source engine version (MariaDB, MySQL, and PostgreSQL), then the response only includes the parameter `SupportedEngineLifeCycles` as an array with a single object. This object includes the start and end dates for RDS standard support.

If the engine version is for a commercial engine (Db2, SQL Server, and Oracle), then the response doesn't include the parameter `SupportedEngineLifeCycles`.

Restoring a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support

When you restore a DB instance or a Multi-AZ DB cluster, select **Enable RDS Extended Support** in the console, or use the Extended Support option in the AWS CLI or the parameter in the RDS API. When you enroll a DB instance or Multi-AZ DB cluster in RDS Extended Support, it is permanently enrolled in RDS Extended Support for the life of the DB instance or Multi-AZ DB cluster.

The default for the RDS Extended Support setting depends on whether you use the console, the AWS CLI, or the RDS API to restore your database. If you use the console, you don't select **Enable RDS Extended Support**, and the major engine version you are restoring is past the RDS end of standard support, then Amazon RDS automatically upgrades your DB instance to a newer engine version. If you use the AWS CLI or the RDS API and you don't specify the RDS Extended Support setting, then Amazon RDS defaults to enabling RDS Extended Support. When you automate by using CloudFormation or other services, this default behavior maintains the availability of your database past the RDS end of standard support date. You can disable RDS Extended Support by using the AWS CLI or the RDS API.

Topics

- RDS Extended Support behavior
- Considerations for RDS Extended Support
- Restore a DB instance or a Multi-AZ DB cluster with RDS Extended Support

RDS Extended Support behavior

The following table summarizes what happens when a major engine version of a DB instance or a Multi-AZ DB cluster that you are restoring has reached the RDS end of standard support.

RDS Extended Support status*	Behavior
Enabled	Amazon RDS charges you for RDS Extended Support.
Disabled**	After the restore finishes, Amazon RDS automatically upgrades your DB instance or Multi-AZ DB cluster to a newer engine version (in a future maintenance window).

* In the RDS console, the RDS Extended Support status appears as Yes or No. In the AWS CLI or RDS API, the RDS Extended Support status appears as `open-source-rds-extended-support` or `open-source-rds-extended-support-disabled`.

** This option is only available when restoring a DB instance or a Multi-AZ DB cluster running PostgreSQL 12 and higher or MySQL 8 and higher.

Considerations for RDS Extended Support

Before restoring a DB instance or a Multi-AZ DB cluster, consider the following items:

- *After* the RDS end of standard support date has passed, if you want to restore a DB instance or a Multi-AZ DB cluster from Amazon S3, you can only do so by using the AWS CLI or the RDS API. Use the `--engine-lifecycle-support` option in the restore-db-cluster-from-s3 AWS CLI command or the `EngineLifecycleSupport` parameter in the RestoreDBClusterFromS3 RDS API operation.
- If you want to prevent RDS from restoring your databases to RDS Extended Support versions, specify `open-source-rds-extended-support-disabled` in the AWS CLI or the RDS API. By doing so, you will avoid any associated RDS Extended Support charges.

  If you specify this setting, Amazon RDS will automatically upgrade your restored database to a newer, supported major version. If the upgrade fails pre-upgrade checks, Amazon RDS will safely roll back to the RDS Extended Support engine version. This database will remain in RDS Extended Support mode, and Amazon RDS will charge you for RDS Extended Support until you manually upgrade your database.

  For example, if you restore a MySQL 5.7 snapshot without using RDS Extended Support, Amazon RDS will attempt to automatically upgrade your database to MySQL 8.0. If this upgrade fails because of an issue that you need to resolve, Amazon RDS will roll back the database to MySQL 5.7. Amazon RDS will keep the database on RDS Extended Support until you can fix the issue. For example, an upgrade might fail because of insufficient storage space. After you fix the issue, you must initiate the upgrade. After the first attempt to upgrade your database, Amazon RDS won't attempt to upgrade it again.
- RDS Extended Support is set at the cluster level. Members of a cluster will always have the same setting for RDS Extended Support in the RDS console, `--engine-lifecycle-support` in the AWS CLI, and `EngineLifecycleSupport` in the RDS API.

For more information, see MySQL versions and Release calendars for Amazon RDS for PostgreSQL.

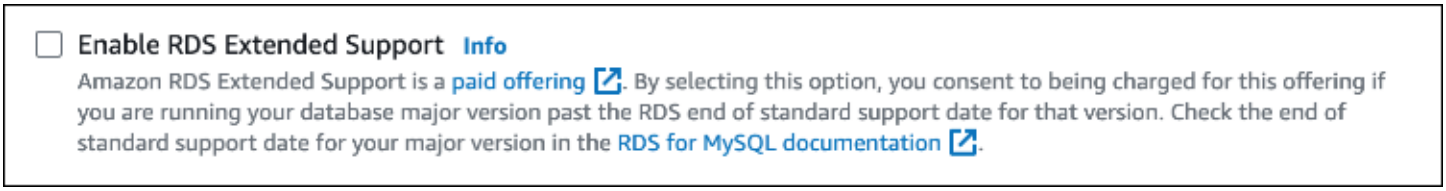
Restore a DB instance or a Multi-AZ DB cluster with RDS Extended Support

You can restore a DB instance or a Multi-AZ DB cluster with an RDS Extended Support version using the AWS Management Console, the AWS CLI, or the RDS API.

Console

When you restore a DB instance or a Multi-AZ DB cluster, select **Enable RDS Extended Support** in the **Engine options** section. If you don't select this setting and the major engine version that you are restoring is past the RDS end of standard support, then Amazon RDS automatically upgrades your DB instance or Multi-AZ DB cluster to a version under RDS standard support.

The following image shows the **Enable RDS Extended Support** setting:



AWS CLI

When you run the restore-db-instance-from-db-snapshot or restore-db-cluster-from-snapshot AWS CLI command, select RDS Extended Support by specifying `open-source-rds-extended-support` for the `--engine-lifecycle-support` option.

If you want to avoid charges associated with RDS Extended Support, set the `--engine-lifecycle-support` option to `open-source-rds-extended-support-disabled`. By default, this option is set to `open-source-rds-extended-support`.

You can also specify this value using the following AWS CLI commands:

- restore-db-cluster-from-s3
- restore-db-cluster-to-point-in-time
- restore-db-instance-from-s3
- restore-db-instance-to-point-in-time

RDS API

When you use the RestoreDBInstanceFromDBSnapshot or RestoreDBClusterFromSnapshot Amazon RDS API operation, select RDS Extended Support by setting the `EngineLifecycleSupport` parameter to `open-source-rds-extended-support`.

If you want to avoid charges associated with RDS Extended Support, set the `EngineLifecycleSupport` parameter to `open-source-rds-extended-support-disabled`. By default, this parameter is set to `open-source-rds-extended-support`.

You can also specify this value using the following RDS API operations:

- RestoreDBClusterFromS3
- RestoreDBClusterToPointInTime
- RestoreDBInstanceFromS3
- RestoreDBInstanceToPointInTime

For more information about restoring a DB instance or a Multi-AZ DB cluster, follow the instructions for your DB engine in Restoring to a DB instance.

Using Amazon RDS Blue/Green Deployments for database updates

A blue/green deployment copies a production database environment to a separate, synchronized staging environment. By using Amazon RDS Blue/Green Deployments, you can make changes to the database in the staging environment without affecting the production environment. For example, you can upgrade the major or minor DB engine version or change database parameters in the staging environment. When you're ready, you can promote the staging environment to be the new production database environment, with downtime typically under one minute.

> **Note**
>
> Currently, Blue/Green Deployments are supported for RDS for MariaDB, RDS for MySQL, and RDS for PostgreSQL only. For Amazon Aurora availability, see Using Amazon Aurora Blue/Green Deployments for database updates in the *Amazon Aurora User Guide*.

Topics

- Overview of Amazon RDS Blue/Green Deployments
- Creating a blue/green deployment in Amazon RDS
- Viewing a blue/green deployment in Amazon RDS
- Switching a blue/green deployment in Amazon RDS
- Deleting a blue/green deployment in Amazon RDS

Overview of Amazon RDS Blue/Green Deployments

By using Amazon RDS Blue/Green Deployments, you can make and test database changes before implementing them in a production environment. A *blue/green deployment* creates a staging environment that copies the production environment. In a blue/green deployment, the *blue environment* is the current production environment. The *green environment* is the staging environment and stays in sync with the current production environment.

You can make changes to the RDS DB instances in the green environment without affecting production workloads. For example, you can upgrade the major or minor DB engine version, upgrade the underlying file system configuration, or change database parameters in the staging environment. You can thoroughly test changes in the green environment. When ready, you can *switch over* the environments to transition the green environment to be the new production environment. The switchover typically takes under a minute with no data loss and no need for application changes.

Because the green environment is a copy of the topology of the production environment, the green environment includes the features used by the DB instance. These features include the read replicas, the storage configuration, DB snapshots, automated backups, Performance Insights, and Enhanced Monitoring. If the blue DB instance is a Multi-AZ DB instance deployment, then the green DB instance is also a Multi-AZ DB instance deployment.

> **Note**
>
> Currently, blue/green deployments are supported only for RDS for MariaDB, RDS for MySQL, and RDS for PostgreSQL. For Amazon Aurora availability, see Overview of Amazon Aurora Blue/Green Deployments in the *Amazon Aurora User Guide*.
> Under certain conditions, RDS for PostgreSQL uses logical replication instead of physical replication to keep the green environment in sync with the blue environment. For more information, see the section called “PostgreSQL replication methods”.

Topics

- Region and version availability
- Benefits of using Amazon RDS Blue/Green Deployments
- Workflow of a blue/green deployment
- Authorizing access to Amazon RDS blue/green deployment operations

- Limitations and considerations for Amazon RDS blue/green deployments
- Best practices for Amazon RDS blue/green deployments

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information, see the section called “Blue/Green Deployments”.

Benefits of using Amazon RDS Blue/Green Deployments

By using Amazon RDS Blue/Green Deployments, you can stay current on security patches, improve database performance, and adopt newer database features with short, predictable downtime. Blue/green deployments reduce the risks and downtime for database updates, such as major or minor engine version upgrades.

Blue/green deployments provide the following benefits:

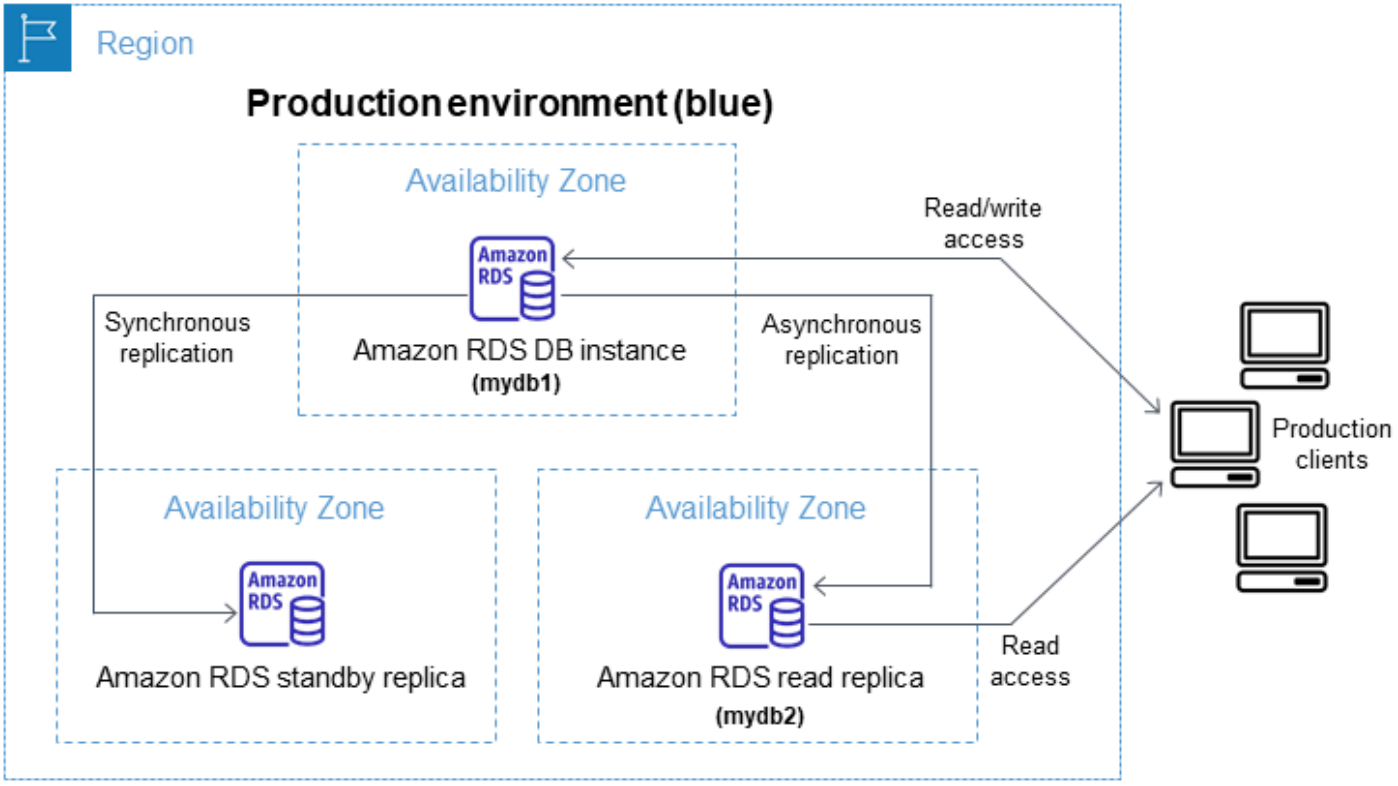
- Easily create a production-ready staging environment.
- Automatically replicate database changes from the production environment to the staging environment.
- Test database changes in a safe staging environment without affecting the production environment.
- Stay current with database patches and system updates.
- Implement and test newer database features.
- Switch over your staging environment to be the new production environment without changes to your application.
- Safely switch over through the use of built-in switchover guardrails.
- Eliminate data loss during switchover.
- Switch over quickly, typically under a minute depending on your workload.

Workflow of a blue/green deployment

Complete the following major steps when you use a blue/green deployment for database updates.

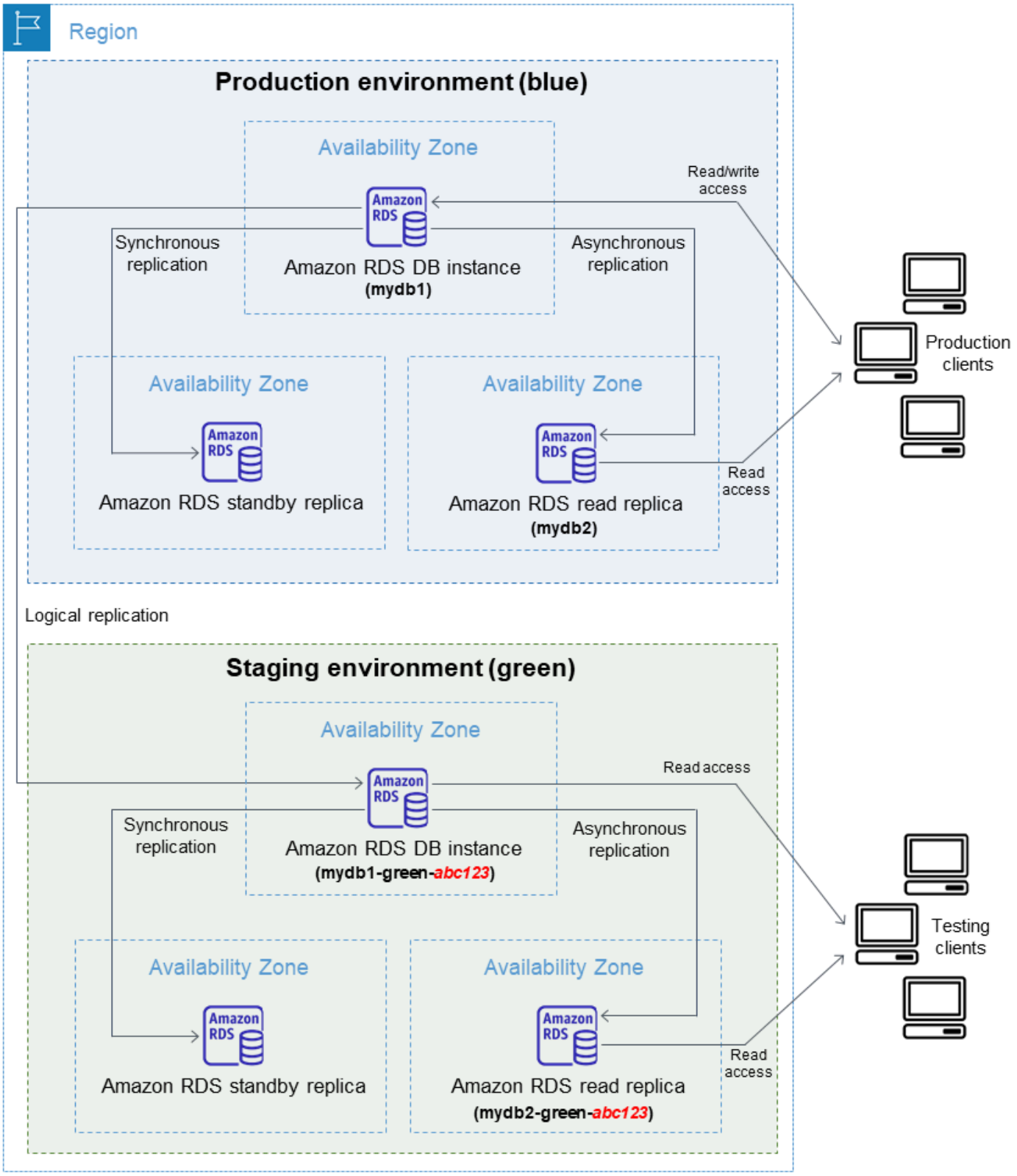
1. Identify a production environment that requires updates.

For example, the production environment in this image has a Multi-AZ DB instance deployment (mydb1) and a read replica (mydb2).



2. Create the blue/green deployment. For instructions, see Creating a blue/green deployment in Amazon RDS.

   The following image shows an example of a blue/green deployment of the production environment from step 1. While creating the blue/green deployment, RDS copies the complete topology and configuration of the primary DB instance to create the green environment. The copied DB instance names are appended with `-green-`*`random-characters`*. The staging environment in the image contains a Multi-AZ DB instance deployment (mydb1-green-*abc123*) and a read replica (mydb2-green-*abc123*).



When you create the blue/green deployment, you can upgrade your DB engine version and specify a different DB parameter group for the DB instances in the green environment. RDS also

configures replication from the primary DB instance in the blue environment to the primary DB instance in the green environment.

After you create the blue/green deployment, the DB instance in the green environment is read-only by default.

3. Make additional changes to the staging environment, if required. For example, you might change the DB instance class used by one or more DB instances in the green environment.

   For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

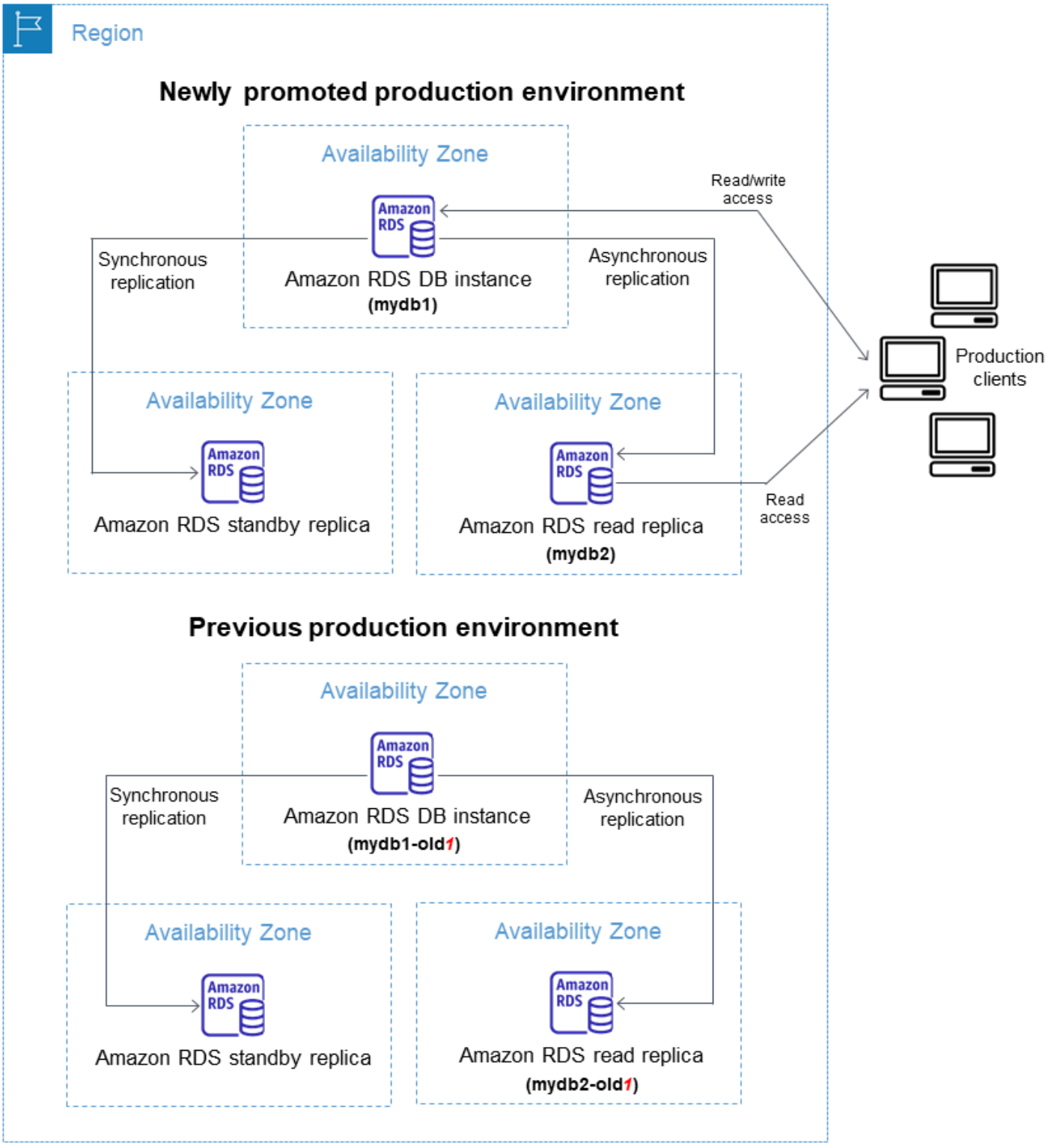
4. Test your staging environment.

   During testing, we recommend that you keep your databases in the green environment read only. Enable write operations on the green environment with caution because they can result in replication conflicts. They can also result in unintended data in the production databases after switchover. To enable write operations for RDS for MySQL, set the `read_only` parameter to `1`, and wait for the parameter group to synchronize. Since `read_only` is a dynamic parameter, so a reboot isn't required. Once synchronized, change `read_only` from `1` to `0`. For RDS for PostgreSQL deployments that use logical replication, set the `default_transaction_read_only` parameter to `off` at the session level. For those that use physical replication, you can't enable write operations on the green environment.

5. When ready, switch over to transition the staging environment to be the new production environment. For instructions, see Switching a blue/green deployment in Amazon RDS.

   The switchover results in downtime. The downtime is usually under one minute, but it can be longer depending on your workload.

   The following image shows the DB instances after the switchover.



After the switchover, the DB instances that were in the green environment become the new production DB instances. The names and endpoints in the current production environment are assigned to the newly switched over production environment, requiring no changes to your

application. As a result, your production traffic now flows to the new production environment. The DB instances in the previous blue environment are renamed by appending `-old`*n* to the current name, where *n* is a number. For example, assume the name of the DB instance in the blue environment is `mydb1`. After switchover, the DB instance name will be `mydb1-old1`.

In the example in the image, the following changes occur during switchover:

- The green environment Multi-AZ DB instance deployment named `mydb1-green-abc123` becomes the production Multi-AZ DB instance deployment named `mydb1`.
- The green environment read replica named `mydb2-green-abc123` becomes the production read replica `mydb2`.
- The blue environment Multi-AZ DB instance deployment named `mydb1` becomes `mydb1-old1`.
- The blue environment read replica named `mydb2` becomes `mydb2-old1`.

6. If you no longer need a blue/green deployment, you can delete it. For instructions, see Deleting a blue/green deployment in Amazon RDS.

   After switchover, the previous production environment isn't deleted so that you can use it for regression testing, if necessary.

Authorizing access to Amazon RDS blue/green deployment operations

Users must have the required permissions to perform operations related to blue/green deployments. You can create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. You can then attach those policies to the IAM permission sets or roles that require those permissions. For more information, see Identity and access management for Amazon RDS.

The user who creates a blue/green deployment must have permissions to perform the following RDS operations:

- `rds:CreateBlueGreenDeployment`
- `rds:AddTagsToResource`
- `rds:CreateDBInstanceReadReplica`

The user who switches over a blue/green deployment must have permissions to perform the following RDS operations:

- `rds:SwitchoverBlueGreenDeployment`
- `rds:ModifyDBInstance`
- `rds:PromoteReadReplica`

The user who deletes a blue/green deployment must have permissions to perform the following RDS operation:

- `rds:DeleteBlueGreenDeployment`
- `rds:DeleteDBInstance`

Amazon RDS provisions and modifies resources in the staging environment on your behalf. These resources include DB instances that use an internally defined naming convention. Therefore, attached IAM policies can't contain partial resource name patterns such as `my-db-prefix-*`. Only wildcards (*) are supported. In general, we recommend using resource tags and other supported attributes to control access to these resources, rather than wildcards. For more information, see Actions, resources, and condition keys for Amazon RDS.

Limitations and considerations for Amazon RDS blue/green deployments

Blue/green deployments in Amazon RDS require careful consideration of factors such as replication slots, resource management, instance sizing, and potential impacts on database performance. The following sections provide guidance to help you optimize your deployment strategy to ensure minimal downtime, seamless transitions, and effective management of your database environment.

Topics

- Limitations for blue/green deployments
- Considerations for blue/green deployments

Limitations for blue/green deployments

The following limitations apply to blue/green deployments.

Topics

- General limitations for blue/green deployments

- RDS for MySQL limitations for blue/green deployments
- RDS for PostgreSQL limitations for blue/green deployments with physical replication
- RDS for PostgreSQL limitations for blue/green deployments with logical replication

General limitations for blue/green deployments

The following general limitations apply to blue/green deployments:

- Blue/green deployments don't support managing master user passwords with AWS Secrets Manager.
- If dedicated log volume (DLV) is enabled on the blue database, it must be enabled on *all* DB instances, including read replicas.
- During switchover, the blue and green environments can't have zero-ETL integrations with Amazon Redshift. You must delete the integration first and switch over, then recreate the integration.
- The Event Scheduler (`event_scheduler` parameter) must be disabled on the green environment when you create a blue/green deployment. This prevents events from being generated in the green environment and causing inconsistencies.
- You can't change an unencrypted DB instance into an encrypted DB instance. In addition, you can't change an encrypted DB instance into an unencrypted DB instance.
- You can't change a blue DB instance to a higher engine version than its corresponding green DB instance.
- The resources in the blue environment and green environment must be in the same AWS account.
- Blue/green deployments aren't supported for the following features:
  - Amazon RDS Proxy
  - Cascading read replicas
  - Cross-Region read replicas
  - CloudFormation
  - Multi-AZ DB cluster deployments

    Blue/green deployments are supported for Multi-AZ DB instance deployments. For more information about Multi-AZ deployments, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

RDS for MySQL limitations for blue/green deployments

The following limitations apply to RDS for MySQL blue/green deployments:

- The blue DB instance can't be an external binlog replica.
- If the source database is associated with a custom option group, you can't specify a major version upgrade when you create the blue/green deployment.

  In this case, you can create a blue/green deployment without specifying a major version upgrade. Then, you can upgrade the database in the green environment. For more information, see Upgrading a DB instance engine version.
- Blue/green deployments don't support the AWS JDBC Driver for MySQL. For more information, see Known Limitations on GitHub.

RDS for PostgreSQL limitations for blue/green deployments with physical replication

The following limitations apply to RDS for PostgreSQL blue/green deployments that use physical replication. For an explanation of when blue/green deployments use physical replication instead of logical replication, see the section called "PostgreSQL replication methods".

- After the green environment is created, you can't perform a manual major version upgrade.
- Blue/green deployments that use physical replication don't support schema changes on the green environment, as it is strictly read-only.
- The blue DB instance can't be a logical source (publisher) or replica (subscriber).
- Blue/Green deployments have the following limitations when configuring delayed replication in RDS for PostgreSQL:
  - **Green source instance** — The `recovery_min_apply_delay parameter` is disregarded, even if configured in the parameter group. Any delay settings on the green source instance do not take effect.
  - **Green replica instance** — The `recovery_min_apply_delay parameter` is fully supported and applied to the PostgreSQL configuration file. Delay settings function as expected during the switchover workflow.
  - Delayed replication isn't compatible with RDS Blue/Green deployments for major version upgrades.

RDS for PostgreSQL limitations for blue/green deployments with logical replication

The following limitations apply to RDS for PostgreSQL blue/green deployments that use logical replication. For an explanation of when blue/green deployments use logical replication instead of physical replication, see the section called “PostgreSQL replication methods”.

- Unlogged tables aren't replicated to the green environment.
- The blue DB instance can't be a logical source (publisher) or replica (subscriber).
- If the blue DB instance is configured as the foreign server of a foreign data wrapper (FDW) extension, you must use the instance endpoint name instead of IP addresses. This allows the configuration to remain functional after switchover.
- In a blue/green deployment, each database requires a logical replication slot. As the number of databases grows, resource overhead increases and can potentially lead to replication lag, especially if the DB instance isn't sufficiently scaled. The impact depends on factors such as database workload and the number of connections. To mitigate this, consider scaling up your DB instance class or reducing the number of databases on the source instance.
- The logical replication apply process in the green environment is single-threaded. If the blue environment generates a high volume of write traffic, the green environment might not be able to keep up. This can lead to replication lag or failure, especially for workloads that produce continuous high write throughput. Make sure to test your workloads thoroughly. For scenarios that require major version upgrades and handling high-volume write workloads, consider alternative approaches such as using AWS Database Migration Service (AWS DMS).
- Blue/Green deployments have the following limitations when configuring delayed replication in RDS for PostgreSQL:
  - **Green source instance** — The `recovery_min_apply_delay parameter` is disregarded, even if configured in the parameter group. Any delay settings on the green source instance do not take effect.
  - **Green replica instance** — The `recovery_min_apply_delay parameter` is fully supported and applied to the PostgreSQL configuration file. Delay settings function as expected during the switchover workflow.
  - Delayed replication isn't compatible with RDS Blue/Green deployments for major version upgrades.
- Creating new partitions on partitioned tables isn't supported during blue/green deployments for RDS for PostgreSQL. Creating new partitions involves data definition language (DDL) operations such as `CREATE TABLE`, which aren't replicated from the blue environment to the green

environment. However, existing partitioned tables and their data will be replicated to the green environment.

- The following limitations apply to PostgreSQL extensions:
  - The `pg_partman` extension must be disabled in the blue environment when you create a blue/green deployment. The extension performs DDL operations such as `CREATE TABLE`, which break logical replication from the blue environment to the green environment.
  - The `pg_cron` extension must remain disabled on all green databases after the blue/green deployment is created. The extension has background workers that run as superuser and bypass the read-only setting of the green environment, which might cause replication conflicts.
  - The `pglogical` and `pgactive` extensions must be disabled on the blue environment when you create a blue/green deployment. After you switch over the green environment to be the new production environment, you can enable the extensions again. In addition, the blue database can't be a logical subscriber of an external instance.
  - If you're using the `pgAudit` extension, it must remain in the shared libraries (`shared_preload_libraries`) on the custom DB parameter groups for both the blue and the green DB instances. For more information, see the section called "Setting up the pgAudit extension".

Logical replication-specific limitations for blue/green deployments

PostgreSQL has certain restrictions related to logical replication, which translate to limitations when creating blue/green deployments for RDS for PostgreSQL DB instances.

The following table describes logical replication limitations that apply to blue/green deployments for RDS for PostgreSQL. For more information, see Restrictions in the PostgreSQL logical replication documentation.

Limitation	Explanation
Data definitio n language (DDL) statement s, such as `CREATE TABLE` and `CREATE`	If Amazon RDS detects a DDL change in the blue environment, your green databases enter a state of **Replication degraded**. You must delete the blue/green deployment and all green databases, then recreate it.

Limitation	Explanation
SCHEMA, aren't replicated from the blue environment to the green environment.	
Data control language (DCL) statements, such as GRANT and REVOKE, aren't replicated from the blue environment to the green environment.	If Amazon RDS PostgreSQL detects an attempt to execute a DCL statement in the blue environment, you will see a warning message. There is no configuration or API available to change this behavior, as it is a limitation of the blue/green deployment process.
NEXTVAL operations on sequence objects aren't synchronized between the blue environment and the green environment.	During switchover, Amazon RDS increments sequence values in the green environment to match those in the blue environment. If you have thousands of sequences, this can delay switchover.

Limitation	Explanation
Large objects in the blue environment aren't replicated to the green environment. This includes both existing large objects and any newly created or modified large objects during the blue/green deployment process.	If Amazon RDS detects the creation or modification of large objects in the blue environment that are stored in the `pg_largeobject` system table, your green databases enter a state of **Replication degraded**. You must delete the blue/green deployment and all green databases, then recreate it.
Materialized views aren't automatically refreshed in the green environment.	Refreshing materialized views in the blue environment doesn't refresh them in the green environment. After switchover, you can manually refresh them using the REFRESH MATERIALIZED VIEW command, or schedule a refresh.
UPDATE and DELETE operations aren't permitted on tables that don't have a primary key.	Before you create a blue/green deployment, make sure that all tables have a primary key or use `REPLICA IDENTITY FULL`. However, only use `REPLICA IDENTITY FULL` if no primary or unique key exists, as it affects replication performance. For more information, see the PostgreSQL documentation.

Considerations for blue/green deployments

Amazon RDS tracks resources in blue/green deployments with the `DbiResourceId` of each resource. This resource ID is an AWS Region-unique, immutable identifier for the resource.

The *resource* ID is separate from the DB *instance* ID. Each one is listed in the database configuration in the RDS console.

The name (instance ID) of a resource changes when you switch over a blue/green deployment, but each resource keeps the same resource ID. For example, a DB instance identifier might be `mydb` in the blue environment. After switchover, the same DB instance might be renamed to `mydb-old1`. However, the resource ID of the DB instance doesn't change during switchover. So, when you switch over the green resources to be the new production resources, their resource IDs don't match the blue resource IDs that were previously in production.

After you switch over a blue/green deployment, consider updating the resource IDs to those of the newly transitioned production resources for integrated features and services that you used with the production resources. Specifically, consider the following updates:

- If you perform filtering using the RDS API and resource IDs, adjust the resource IDs used in filtering after switchover.
- If you use CloudTrail for auditing resources, adjust the consumers of the CloudTrail to track the new resource IDs after switchover. For more information, see Monitoring Amazon RDS API calls in AWS CloudTrail.
- If you use the Performance Insights API, adjust the resource IDs in calls to the API after switchover. For more information, see Monitoring DB load with Performance Insights on Amazon RDS.

  You can monitor a database with the same name after switchover, but it doesn't contain the data from before the switchover.
- If you use resource IDs in IAM policies, make sure you add the resource IDs of the newly transitioned resources when necessary. For more information, see Identity and access management for Amazon RDS.
- If you have IAM roles associated with your DB instance, make sure to reassociate them after switchover. Attached roles aren't automatically copied to the green environment.
- If you authenticate to your DB instance using IAM database authentication, make sure that the IAM policy used for database access has both the blue and the green databases listed under the

`Resource` element of the policy. This is required in order to connect to the green database after switchover. For more information, see the section called "Creating and using an IAM policy for IAM database access".

- If you use AWS Backup to manage automated backups of resources in a blue/green deployment, adjust the resource IDs used by AWS Backup after switchover. For more information, see Using AWS Backup to manage automated backups for Amazon RDS.
- If you want to restore a manual or automated DB snapshot for a DB instance that was part of a blue/green deployment, make sure you restore the correct DB snapshot by examining the time when the snapshot was taken. For more information, see Restoring to a DB instance.
- If you want to describe a previous blue environment DB instance automated backup or restore it to a point in time, use the resource ID for the operation.

  Because the name of the DB instance changes during switchover, you can't use its previous name for `DescribeDBInstanceAutomatedBackups` or `RestoreDBInstanceToPointInTime` operations.

  For more information, see Restoring a DB instance to a specified time for Amazon RDS.
- When you add a read replica to a DB instance in the green environment of a blue/green deployment, the new read replica won't replace a read replica in the blue environment when you switch over. However, the new read replica is retained in the new production environment after switchover.
- After you switch over, AWS Database Migration Service (AWS DMS) replication tasks can't resume because the checkpoint from the blue environment is invalid in the green environment. You must recreate the DMS task with a new checkpoint to continue replication.
- When you delete a DB instance in the green environment of a blue/green deployment, you can't create a new DB instance to replace it in the blue/green deployment.

  If you create a new DB instance with the same name and Amazon Resource Name (ARN) as the deleted DB instance, it has a different `DbiResourceId`, so it isn't part of the green environment.

  The following behavior results if you delete a DB instance in the green environment:

  - If the DB instance in the blue environment with the same name exists, it won't be switched over to the DB instance in the green environment. This DB instance won't be renamed by adding `-old`*n* to the DB instance name.
  - Any application that points to the DB instance in the blue environment continues to use the same DB instance after switchover.

The same behavior applies to DB instances and read replicas.

- If you use resource tags for access control or operational management, you need to understand that tag changes aren't synchronized between blue and green environments until switchover. When you create a blue/green deployment, tags from the blue environment are copied to the green environment. After creation, any tag modifications that you make to either environment aren't automatically synchronized. During switchover, blue environment tags replace all tags in the green environment. Apply all necessary tags to the blue environment before you create the blue/green deployment, or reapply required tags to the new production environment after switchover. For more information about tags, see Tagging Amazon RDS resources.

Best practices for Amazon RDS blue/green deployments

The following are best practices for blue/green deployments.

Topics

- General best practices for blue/green deployments
- RDS for MySQL best practices for blue/green deployments
- RDS for MySQL best practices for blue/green deployments
- PostgreSQL replication methods for blue/green deployments

General best practices for blue/green deployments

Consider the following general best practices when you create a blue/green deployment.

- Thoroughly test the DB instances in the green environment before switching over.
- Keep your databases in the green environment read only. We recommend that you enable write operations on the green environment with caution because they can result in replication conflicts. They can also result in unintended data in the production databases after switchover.
- If you use a blue/green deployment to implement schema changes, make only replication-compatible changes.

  For example, you can add new columns at the end of a table without disrupting replication from the blue deployment to the green deployment. However, schema changes, such as renaming columns or renaming tables, break replication to the green deployment.

For more information about replication-compatible changes, see Replication with Differing Table Definitions on Source and Replica in the MySQL documentation and Restrictions in the PostgreSQL logical replication documentation.

> **Note**
>
> This limitation doesn't apply to RDS for PostgreSQL blue/green deployments that use physical replication. For more information, see the section called "RDS for PostgreSQL limitations for blue/green deployments with physical replication".

- After you create the blue/green deployment, handle lazy loading if necessary. Make sure data loading is complete before switching over. For more information, see Lazy loading and storage initialization for blue/green deployments.
- When you switch over a blue/green deployment, follow the switchover best practices. For more information, see the section called "Switchover best practices".

RDS for MySQL best practices for blue/green deployments

Consider the following best practices when you create a blue/green deployment from an RDS for MySQL DB instance.

- Avoid using non-transactional storage engines, such as MyISAM, that aren't optimized for replication.
- Optimize read replicas and the green environment for binary log replication. If supported by your DB engine, enable GTID, parallel, and crash-safe replication to ensure data consistency and durability before you create your blue/green deployment. For more information, see Using GTID-based replication.
- If the green environment experiences replica lag, consider the following:
  - Temporarily set the `innodb_flush_log_at_trx_commit` parameter to 2 in the green DB parameter group. After replication catches up, revert to the default value of 1 before switchover. If an unexpected shutdown or crash occurs with the temporary parameter value, rebuild the green environment to avoid undetected data corruption.
  - To reduce write latency and improve replication throughput, temporarily change green Multi-AZ DB instances to Single-AZ DB instances. Re-enable Multi-AZ right before switchover.

RDS for MySQL best practices for blue/green deployments

In addition to the above listed general and engine specific best practices, consider the following best practices for RDS for MySQL DB instance

- Monitor the following CloudWatch metrics to identify periods of low activity in your production environment:
  - `DatabaseConnections`
  - `ActiveTransactions`

  Schedule the blue/green switchover during your planned maintenance window or during a period of low activity.
- Blue/Green switchover duration varies based on your workload and the number of secondary regions. When you initiate a blue/green switchover, the service waits for replica lag to reach zero before proceeding. We recommend checking replica lag before initiating a switchover.
- If you intend to use a DB parameter or DB Cluster parameter group other than the default one for your green environment, create the desired parameter group with the same name in all secondary regions before initiating the blue/green deployment.

RDS for PostgreSQL best practices for blue/green deployments

Consider the following best practices when you create a blue/green deployment from an RDS for PostgreSQL DB instance.

Topics

- RDS for PostgreSQL general best practices for blue/green deployments
- RDS for PostgreSQL best practices for blue/green deployments with physical replication
- RDS for PostgreSQL best practices for blue/green deployments with logical replication

RDS for PostgreSQL general best practices for blue/green deployments

Consider the following general best practices when you create a blue/green deployment from an RDS for PostgreSQL DB instance.

- Update all of your PostgreSQL extensions to the latest version before you create a blue/green deployment. For more information, see the section called “Upgrading PostgreSQL extensions”.

- Long-running transactions can cause significant replica lag. To reduce replica lag, consider doing the following:
  - Reduce long-running transactions that can be delayed until after the green environment catches up to the blue environment.
  - Reduce bulk operations on the blue environment until after the green environment catches up to the blue environment.
  - Initiate a manual vacuum freeze operation on busy tables prior to creating the blue/green deployment.
  - For PostgreSQL version 12 and higher, disable the `index_cleanup` parameter on large or busy tables to increase the rate of normal maintenance on blue databases. For more information, see the section called "Vacuuming a table as quickly as possible".

    Note

    Regularly skipping index cleanup during vacuuming can lead to index bloat, which might degrade scan performance. As a best practice, use this approach only while using a blue/green deployment. Once the deployment is complete, we recommend resuming regular index maintenance and cleanup.
- Slow replication can cause senders and receivers to restart often, which delays synchronization. To ensure that they remain active, disable timeouts by setting the `wal_sender_timeout` parameter to `0` in the blue environment, and the `wal_receiver_timeout` parameter to `0` in the green environment.
- To prevent write-ahead log (WAL) segments from being removed from the blue environment, set the `wal_keep_segments` parameter to 15625 for PostgreSQL version 13 and lower. For version 14 and higher, set the `wal_keep_size` parameter too 1 TiB, if there's enough free storage space.

RDS for PostgreSQL best practices for blue/green deployments with physical replication

With physical replication, Amazon RDS creates a read replica of the source DB instance. For related parameters, monitoring, tuning, and troubleshooting, see the section called "RDS for PostgreSQL read replicas".

For an explanation of when blue/green deployments use physical replication instead of logical replication, see the section called "PostgreSQL replication methods".

RDS for PostgreSQL best practices for blue/green deployments with logical replication

Consider the following best practices when you create a blue/green deployment that uses logical replication. For an explanation of when blue/green deployments use logical replication instead of physical replication, see the section called "PostgreSQL replication methods".

- If your database has sufficient freeable memory, increase the value of the `logical_decoding_work_mem` DB parameter in the blue environment. Doing so allows for less decoding on disk and instead uses memory. For more information, see the PostgreSQL documentation.
  - You can monitor transaction overflow being written to disk using the `ReplicationSlotDiskUsage` CloudWatch metric. This metric offers insights into the disk usage of replication slots, helping identify when transaction data exceeds memory capacity and is stored on disk. You can monitor freeable memory with the `FreeableMemory` CloudWatch metric. For more information, see the section called "Amazon CloudWatch instance-level metrics for Amazon RDS".
  - In RDS for PostgreSQL version 14 and higher, you can monitor the size of logical overflow files using the `pg_stat_replication_slots` system view.
- If you're using the `aws_s3` extension, give the green DB instance access to Amazon S3 through an IAM role after the green environment is created. This allows the import and export commands to continue functioning after switchover. For instructions, see the section called "Setting up access to an Amazon S3 bucket".
- Review the performance of your UPDATE and DELETE statements and evaluate whether creating an index on the column used in the WHERE clause can optimize these queries. This can enhance performance when the operations are replayed in the green environment.
- If you're using triggers, make sure they don't interfere with the creating, updating, and dropping of `pg_catalog.pg_publication`, `pg_catalog.pg_subscription`, and `pg_catalog.pg_replication_slots` objects whose names start with 'rds'.
- If you specify a higher engine version for the green environment, run the `ANALYZE` operation on all databases to refresh the `pg_statistic` table. Optimizer statistics aren't transferred during a major version upgrade, so you must regenerate all statistics to avoid performance issues. For additional best practices during major version upgrades, see the section called "How to perform a major version upgrade".
- Avoid configuring triggers as `ENABLE REPLICA` or `ENABLE ALWAYS` if the trigger is used on the source to manipulate data. Otherwise, the replication system propagates changes and executes the trigger, which leads to duplication.

PostgreSQL replication methods for blue/green deployments

Amazon RDS for PostgreSQL primarily uses physical replication for blue/green deployments. However, if you request a major version upgrade when you create the blue/green deployment, and your source DB instance runs one of the PostgreSQL versions listed in the table below, Amazon RDS uses logical replication instead.

The following table outlines when Amazon RDS uses physical versus logical replication for PostgreSQL blue/green deployments.

Source PostgreSQL DB instance version	Upgrade action in blue/green deployment	Replication method
• 16.1 and all higher major and minor versions • 15.4 and higher 15 versions • 14.9 and higher 14 versions • 13.12 and higher 13 versions • 12.16 and higher 12 versions • 11.21 and higher 11 versions	Major version upgrade (green instance on higher major engine version than blue)	Logical replication
All supported versions	Minor version upgrade, or no upgrade (green instance on same major engine version as blue)	Physical replication

Note

Major version upgrades are not supported for blue/green deployments with source RDS for PostgreSQL versions 15.3 and lower, 14.8 and lower, 13.11 and lower, 12.15 and lower, or 11.20 and lower.

For information about the limitations of blue/green deployments that use physical and logical replication, see the following sections:

- the section called “RDS for PostgreSQL limitations for blue/green deployments with physical replication”
- the section called “RDS for PostgreSQL limitations for blue/green deployments with logical replication”

Creating a blue/green deployment in Amazon RDS

When you create a blue/green deployment, you specify the source DB instance to copy in the deployment. The DB instance you choose is the production DB instance, and it becomes the primary DB instance in the blue environment. This DB instance is copied to the green environment, and RDS configures replication from the DB instance in the blue environment to the DB instance in the green environment.

RDS copies the blue environment's topology and features to a staging area. If the blue DB instance has read replicas, they are copied as replicas of the green instance. The allocated storage of all green replicas matches the green primary instance, while other storage parameters are inherited from the blue replicas.

If the blue DB instance is a Multi-AZ DB instance deployment, then the green DB instance is created as a Multi-AZ DB instance deployment.

Topics

- Preparing for a blue/green deployment
- Specifying changes when creating a blue/green deployment
- Lazy loading and storage initialization for blue/green deployments
- Creating a blue/green deployment
- Settings for creating blue/green deployments

Preparing for a blue/green deployment

There are certain steps you must take before you create a blue/green deployment, depending on the engine that your DB instance is running.

Topics

- Preparing an RDS for MySQL or RDS for MariaDB DB instance for a blue/green deployment
- Preparing an RDS for PostgreSQL DB instance for a blue/green deployment with physical replication
- Preparing an RDS for PostgreSQL DB instance for a blue/green deployment with logical replication

Preparing an RDS for MySQL or RDS for MariaDB DB instance for a blue/green deployment

Before you create a blue/green deployment for an RDS for MySQL or RDS for MariaDB DB instance, you must enable automated backups. For instructions, see the section called "Enabling automated backups".

Preparing an RDS for PostgreSQL DB instance for a blue/green deployment with physical replication

Before you create an RDS for PostgreSQL blue/green deployment that uses physical replication, make sure to do the following. For a list of versions that use physical replication versus logical replication, see the section called "PostgreSQL replication methods".

- Enable automated backups on the DB instance. For instructions, see the section called "Enabling automated backups".
- Confirm that the DB instance isn't the source or target of external replication. For more information, see the section called "General limitations".

Preparing an RDS for PostgreSQL DB instance for a blue/green deployment with logical replication

Before you create an RDS for PostgreSQL blue/green deployment that uses logical replication, make sure to do the following. For a list of versions that use logical replication versus physical replication, see the section called "PostgreSQL replication methods".

- Associate the instance with a custom DB parameter group with logical replication (`rds.logical_replication`) turned on. Logical replication is required for replication from the blue environment to the green environment. For instructions, see the section called "Modifying parameters in a DB parameter group".

Because blue/green deployments require at least one background worker per database, make sure to tune the following configuration settings according to your workload. For instructions to tune each setting, see Configuration Settings in the PostgreSQL documentation.

- `max_replication_slots`
- `max_wal_senders`
- `max_logical_replication_workers`
- `max_worker_processes`

After you enable logical replication and set all configuration options, make sure to reboot the DB instance so that your changes take effect. Blue/green deployments *require* that the DB instance be in sync with the DB parameter group, otherwise creation fails. For more information, see the section called "Rebooting a DB instance".

- Confirm that the DB instance isn't the source or target of external replication. For more information, see the section called "General limitations".
- Make sure that all tables in the DB instance have a primary key. PostgreSQL logical replication doesn't allow `UPDATE` or `DELETE` operations on tables that don't have a primary key.
- RDS for PostgreSQL uses PostgreSQL's native logical replication, storing write-ahead logs (WAL) segments on the blue instance until they're replayed on the green environment. Before you create a blue/green deployment, verify that the blue instance has adequate capacity by checking the following metrics:
  - `FreeStorageSpace`
  - `TransactionLogsGeneration`
  - `TransactionLogsDiskUsage`
  - `OldestReplicationSlotLag`

  To estimate the additional storage required on the blue instance, monitor the `TransactionLogsGeneration` CloudWatch metric during peak workload periods. For example, if your workload generates 100 GB of WAL data over 24 hours, ensure you have at least 100 GB of extra storage to accommodate one day's worth of WAL segments. For more information, see *Monitoring metrics in a DB instance*.

Specifying changes when creating a blue/green deployment

You can make the following changes to the DB instance in the green environment when you create the blue/green deployment.

You can make other modifications to the DB instance in the green environment after it is deployed. For example, you might specify a higher engine version or a different parameter group.

For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

Topics

- Specify a higher engine version
- Specify a different DB parameter group
- Modify storage and performance settings
- Enable RDS Optimized Writes
- Upgrade the storage configuration

Specify a higher engine version

You can specify a higher engine version if you want to test a DB engine upgrade. Upon switchover, the database is upgraded to the major or minor DB engine version that you specify.

Specify a different DB parameter group

You can test how parameter changes affect the DB instances in the green environment or specify a parameter group for a new major DB engine version in the case of an upgrade.

If you specify a different DB parameter group, the specified DB parameter group is associated with all of the DB instances in the green environment. If you don't specify a different parameter group, each DB instance in the green environment is associated with the parameter group of its corresponding blue DB instance.

Modify storage and performance settings

Adjust storage and performance settings in the green environment to optimize resource allocation. These settings include allocated storage, provisioned IOPS, storage type, and storage throughput (for gp3 storage).

You can change the storage type of the green DB instance to gp2, gp3, io1, or io2. For gp3 storage, you can also adjust storage throughput to enhance data transfer performance for high-demand workloads, or to reduce costs for less intensive applications. For more information, see Amazon RDS DB instance storage.

You can also choose to increase or decrease allocated storage in the green environment. However, a storage reduction only occurs if the target allocated storage is at least 20% more than the current storage usage. If you decrease the allocated storage, Amazon RDS initiates a storage configuration upgrade. For more information, see the section called "Upgrade the storage configuration". The minimum target storage is calculated as:

```
Minimum Target Storage = Total Allocated Storage × Current Apparent Utilization × 1.2
```

1. Total Allocated Storage: The storage capacity provisioned for your DB instance, visible in the RDS console.
2. Current Apparent Utilization: The percentage of allocated storage in use. To estimate it, use the `os.fileSys.usedPercent` metric from Performance Insights (Database Insights) or Enhanced Monitoring.

> **Note**
>
> Since storage utilization fluctuates over time, we recommend setting the target storage slightly above the calculated minimum to account for potential increases during the reduction process.

If the blue DB instance uses magnetic storage, you must change the green DB instance to a General Purpose or Provisioned IOPS storage type in order to increase or decrease the allocated storage.

Enable RDS Optimized Writes

You can use a blue/green deployment to upgrade to a DB instance class that supports RDS Optimized Writes. You can only enable RDS Optimized Writes on a database that was *created* with a supported DB instance class. Thus, this option creates a green database that uses a supported DB instance class, which enables you to turn on RDS Optimized Writes on the green DB instance.

If you're upgrading from a DB instance class that doesn't support RDS Optimized Writes to one that does, you must also upgrade the storage configuration of the green DB instance. For more information, see the section called “Upgrade the storage configuration”.

You can only upgrade the DB instance class of the *primary* green DB instance. By default, read replicas in the green environment inherit the DB instance settings from the blue environment. After the green environment is successfully created, you must manually modify the DB instance class of the read replicas in the green environment.

Some instance class upgrades aren't supported depending on the engine version and instance class of the blue DB instance. For more information about DB instance classes, see the section called “DB instance classes”.

Upgrade the storage configuration

If your blue database isn't on the latest storage configuration, RDS can migrate the green DB instance from the older storage configuration (32-bit file system) to the preferred configuration. You can use RDS Blue/Green Deployments to overcome the scaling limitations on storage and file size for older 32-bit file systems. In addition, this setting changes the storage configuration to be compatible with RDS Optimized Writes if the specified DB instance class supports Optimized Writes.

> **Note**
>
> Upgrading the storage configuration is an I/O-intensive operation and leads to longer creation times for blue/green deployments. The storage upgrade process is faster if the blue DB instance uses Provisioned IOPS SSD (io1 or io2 Block Express) storage, and if you provisoned the green environment with an instance size of 4xlarge or larger. Storage upgrades involving General Purpose SSD (gp2) storage can deplete your I/O credit balance, resulting in longer upgrade times. For more information, see the section called “DB instance storage”.
>
> During the storage upgrade, the green DB instance is temporarily unavailable, while the blue DB instance remains available. Replication pauses during this time. Monitor storage on the blue instance, and consider scaling if storage reaches 90%, as the green instance automatically scales by 10% after the upgrade.

This option is only available if your blue database is *not* on the latest storage configuration, or if you're changing the DB instance class within the same request. You can only upgrade the storage configuration when initially creating a blue/green deployment.

Lazy loading and storage initialization for blue/green deployments

When you create a blue/green deployment, Amazon RDS creates the primary DB instance in the green environment by restoring from a DB snapshot. After it's created, the green DB instance and its read replicas continue to load data in the background through a process known as *lazy loading*.

Lazy loading only loads data blocks as applications request them. If you attempt to access data that hasn't been loaded yet, Amazon EBS immediately retrieves it from Amazon S3, while remaining data continues to load in the background. For more information, see Amazon EBS snapshots.

To accelerate full volume performance, Amazon RDS provides storage initialization, which reads all blocks in the green environment volume. Amazon EBS proactively downloads blocks from Amazon S3, providing maximum volume performance from the first use. Storage initialization occurs entirely in the background, ensuring no impact on your DB instance availability or ongoing activities, such as patching or upgrades.

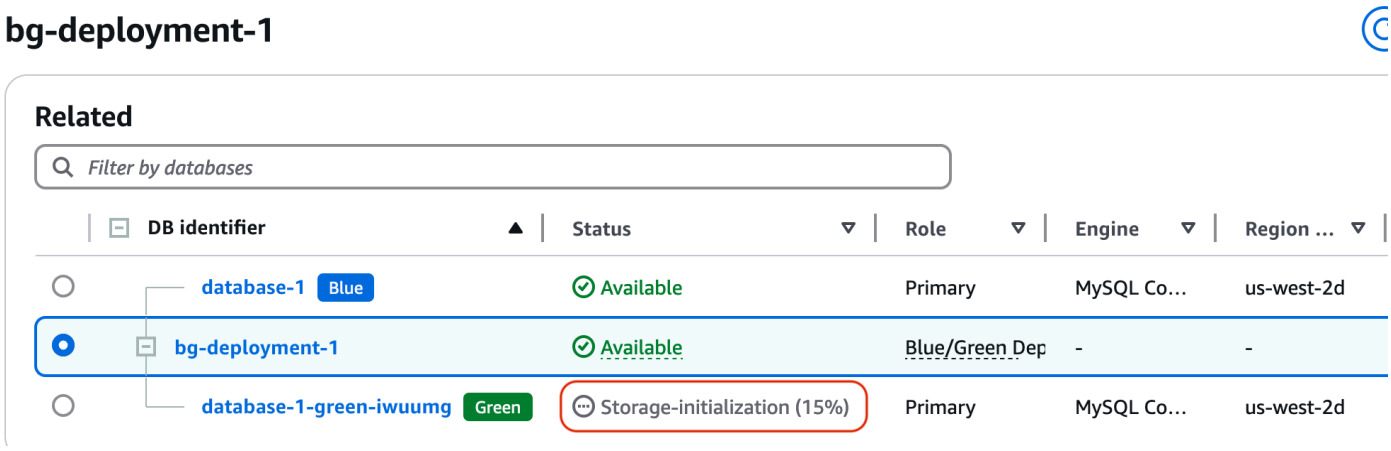
Storage initialization is available only for instances in blue/green deployments with `gp2`, `gp3`, `io1`, and `io2` volume types. It supports all instance classes except the t3 and t4 families. If you modify a green DB instance in a Single-AZ deployment to a Multi-AZ DB instance deployment, storage initialization includes the secondary node in the Multi-AZ configuration.

During storage initialization, the instance remains fully available and usable for database operations, though storage might not reach full performance until initialization completes. While the storage initialization is underway, the overall instance status changes to **Storage-initialization**, and the progress indicator reflects the minimum initialization level across all volumes of the DB instance.

Use the console, AWS CLI, or Amazon RDS API to monitor storage initialization.

Console

In the AWS Management Console, you see the progress of storage initialization with the DB instance status.



AWS CLI

With the AWS CLI, you can monitor storage initialization with the describe-db-instances command. The `PercentProgress` field in the response shows what percentage of data has been retrieved from Amazon S3.

```
aws rds describe-db-instances --db-instance-identifier my-db-instance

{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "my-db-instance",
            "DBInstanceClass": "db.m5.2xlarge",
            "Engine": "postgres",
            "DBInstanceStatus": "storage-initialization",
            ...
            "PercentProgress": "34"
        }
    ]
}
```

Amazon RDS API

With the Amazon RDS API, you retrieve the status of storage initialization by calling the DescribeDBInstances action.

The progress indicator updates as the background initialization job advances, allowing you to track storage readiness before full storage initialization completes. Storage initialization enables optimized performance as your green DB instance becomes fully operational.

Creating a blue/green deployment

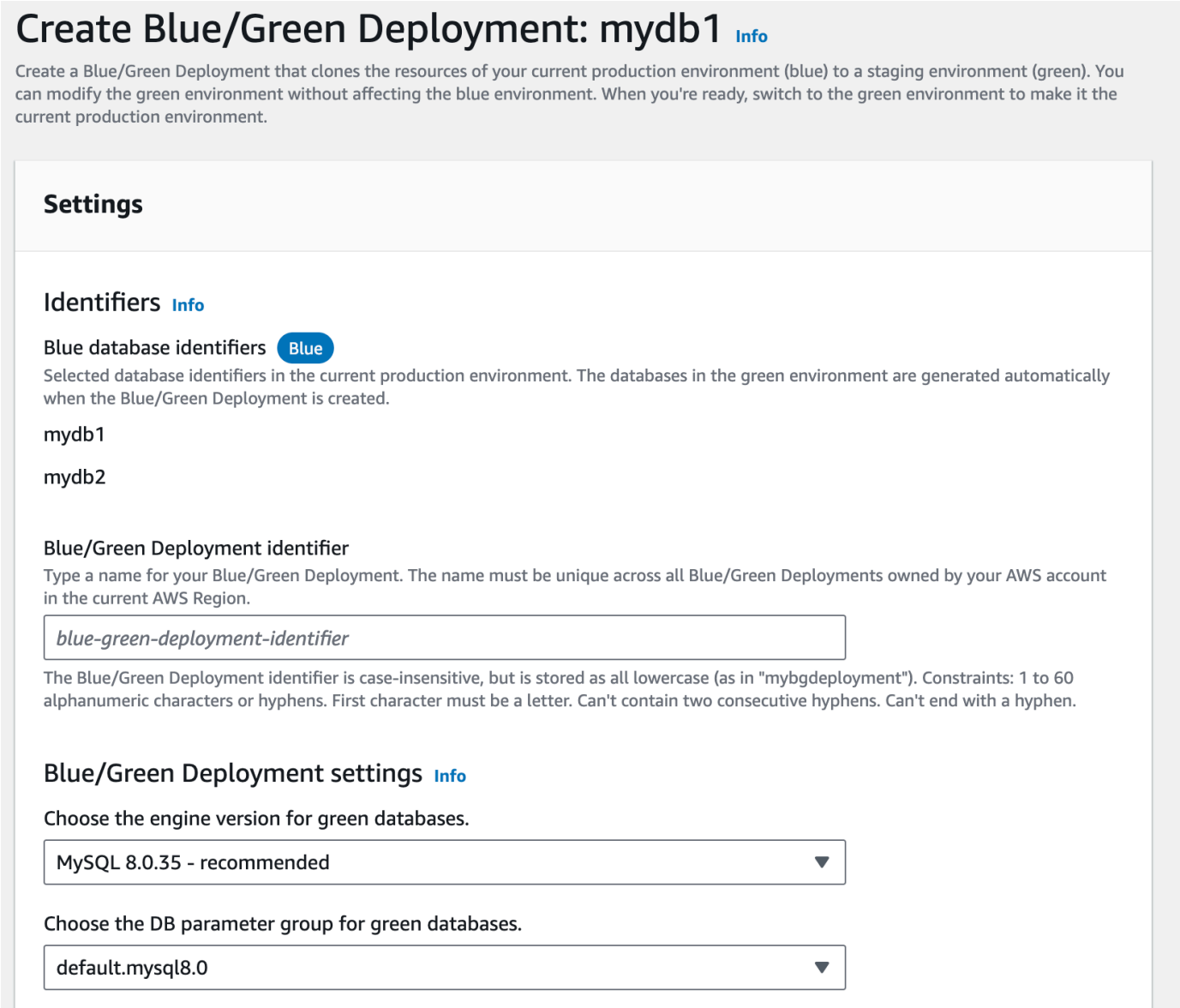
You can create a blue/green deployment using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To create a blue/green deployment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to copy to a green environment.
3. Choose **Actions**, **Create blue/green deployment**.

   The **Create blue/green deployment** page appears.



4. Review the blue database identifiers. Make sure that they match the DB instances that you expect in the blue environment. If they don't, choose **Cancel**.
5. For **Blue/green deployment name**, enter a name for your blue/green deployment.
6. In the remaining sections, specify the settings for the green environment. For information about each setting, see the section called “Available settings”.

   You can make other modifications to the databases in the green environment after it is deployed.
7. Choose **Create**.

AWS CLI

To create a blue/green deployment using the AWS CLI, use the create-blue-green-deployment command. For information about all available options, see the section called "Available settings".

Example

For Linux, macOS, or Unix:

```
aws rds create-blue-green-deployment \
    --blue-green-deployment-name my-blue-green-deployment \
    --source arn:aws:rds:us-east-2:123456789012:db:mydb1 \
    --target-engine-version 8.0.31 \
    --target-db-parameter-group-name mydbparametergroup
```

For Windows:

```
aws rds create-blue-green-deployment ^
    --blue-green-deployment-name my-blue-green-deployment ^
    --source arn:aws:rds:us-east-2:123456789012:db:mydb1 ^
    --target-engine-version 8.0.31 ^
    --target-db-parameter-group-name mydbparametergroup
```

RDS API

To create a blue/green deployment by using the Amazon RDS API, use the `CreateBlueGreenDeployment` operation. For information about each option, see the section called "Available settings".

Settings for creating blue/green deployments

The following table explains the settings that you can choose when you create a blue/green deployment. For more information about the AWS CLI options, see create-blue-green-deployment. For more information about the RDS API parameters, see CreateBlueGreenDeployment.

Console setting	Setting description	CLI option and RDS API parameter
Allocated storage	The amount of storage to allocate for your green DB instance (in gibibytes). You can choose to	**CLI option:** `--target-allocated-storage`

Console setting	Setting description	CLI option and RDS API parameter
	increase or decrease the allocated storage. If your blue DB instance uses magnetic (`standard`) storage, you must change the green DB instance to a General Purpose or Provisioned IOPS storage type in order to modify the allocated storage in the green environment. For more information, see Amazon RDS DB instance storage.	**API parameter:** `TargetAllocatedStorage`

Blue/Green Deployment identifier	A name for the blue/green deployment.	**CLI option:**
		
		`--blue-green-deployment-name`
		
		API parameter:
		
		`BlueGreenDeploymentName`

Blue database identifier	The identifier of the instance that you want to copy to the green environment. When using the CLI or API, specify the instance Amazon Resource Name (ARN).	**CLI option:**
		
		`--source`
		
		API parameter:
		
		`Source`

Console setting	Setting description	CLI option and RDS API parameter

DB parameter group for green databases	A parameter group to associate with the databases in the green environment.	**CLI option:**
		`--target-db-parameter-group-name`
		`--target-db-cluster-parameter-group-name`
		API parameter:
		`TargetDBParameterGroupName`
		`TargetDBClusterParameterGroupName`

Enable Optimized Writes for green database	Enable RDS Optimized Writes on the green primary DB instance. For more information, see the section called "Enable RDS Optimized Writes". If you're changing from a DB instance class that doesn't support Optimized Writes to one that does, you also need to perform a storage configuration upgrade. For more information, see the section called "Upgrade the storage configuration".	For the CLI and API, specifying a target DB instance class that supports RDS Optimized Writes automatically enables it on the green primary DB instance.

Console setting	Setting description	CLI option and RDS API parameter

Engine version for green databases	Upgrade the databases in the green environment to the specified DB engine version.
	
	If not specified, each database in the green environment is created with the same engine version as the corresponding DB instance in the blue environment.
	
	If you choose an RDS for PostgreSQL DB instance that uses logical replication, review and acknowledge the logical replicati on limitations. For more informati on, see the section called "Logical replication-specific limitations for blue/green deployments".
		CLI option:
		
		`--target-engine-version`
		
		RDS API parameter:
		
		`TargetEngineVersion`

Green DB instance class	The compute and memory capacity of each DB instance in the green environment, for example `db.m5d.xlarge` .
	
	This option is only visible when you enable RDS Optimized Writes for the green database.
		CLI option:
		
		`--target-db-instance-class`
		
		RDS API parameter:
		
		`TargetDBInstanceClass`

Provisioned IOPS	The amount of provisioned input/ output operations per second (IOPS) to be initially allocated for the green database.
	
	This value applies only to the green primary DB instance, not green replicas.
		CLI option:
		
		`--target-iops`
		
		RDS API parameter:
		
		`TargetIops`

Console setting	Setting description	CLI option and RDS API parameter

Storage configuration upgrade	Choose whether to upgrade your storage file system configuration. If you enable this setting, RDS migrates the green database from the old storage file system to the preferred configuration.
	
	This option is only available if your blue database is *not* on the latest storage configuration, or if you're enabling RDS Optimized Writes within the same request. You can only upgrade the storage configuration when initially creating a blue/green deployment.
	
	For more information, see the section called "Upgrading the storage file system".
		CLI option:
		
		`--upgrade-target-storage-config`
		
		RDS API parameter:
		
		`UpgradeTargetStorageConfig`

Storage throughput	The storage throughput value for the green database. This setting is visible only if you choose General Purpose SSD (gp3) for the storage type.
	
	This value applies only to the green primary DB instance, not green replicas.
	
	For more information, see gp3 storage (recommended).
		CLI option:
		
		`--target-storage-throughput`
		
		RDS API parameter:
		
		`TargetStorageThroughput`

Console setting	Setting description	CLI option and RDS API parameter

Storage type	The storage type for the green database. The following storage types are supported:
	• General Purpose SSD (gp2)
	• General Purpose SSD (gp3)
	• Provisioned IOPS (io1)
	• Provisioned IOPS SSD (io2)
	This value applies only to the green primary DB instance, not green replicas.
	For more information, see Amazon RDS storage types.
		CLI option:
		`--target-storage-type`
		RDS API parameter:
		`TargetStorageType`

Viewing a blue/green deployment in Amazon RDS

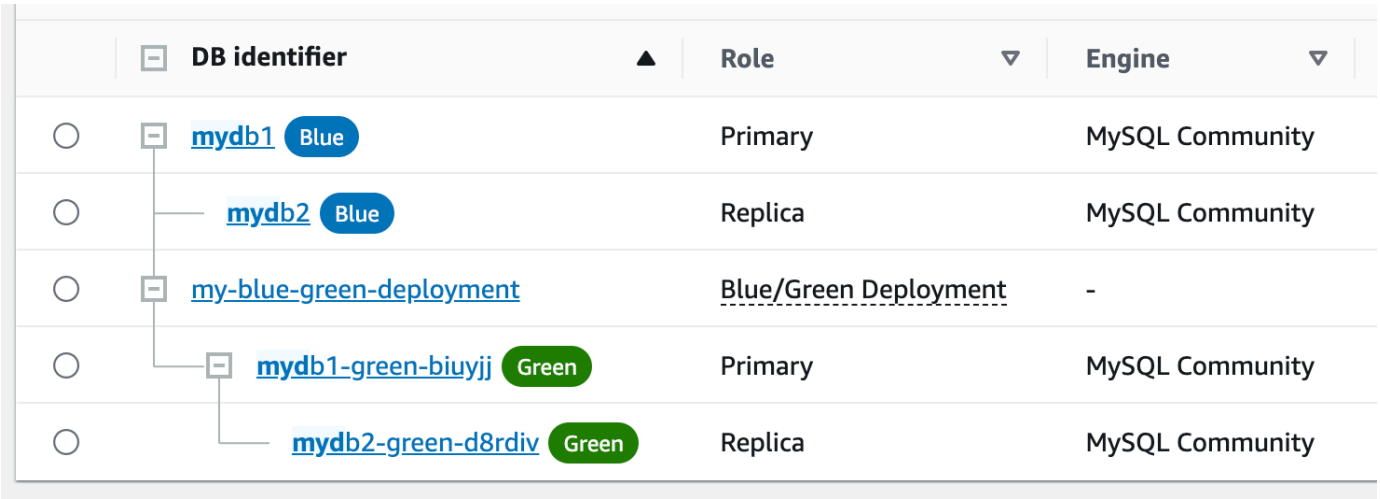
You can view the details about a blue/green deployment using the AWS Management Console, the AWS CLI, or the RDS API.

You can also view and subscribe to events for information about a blue/green deployment. For more information, see Blue/green deployment events.

Console

To view the details about a blue/green deployment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then find the blue/green deployment in the list.

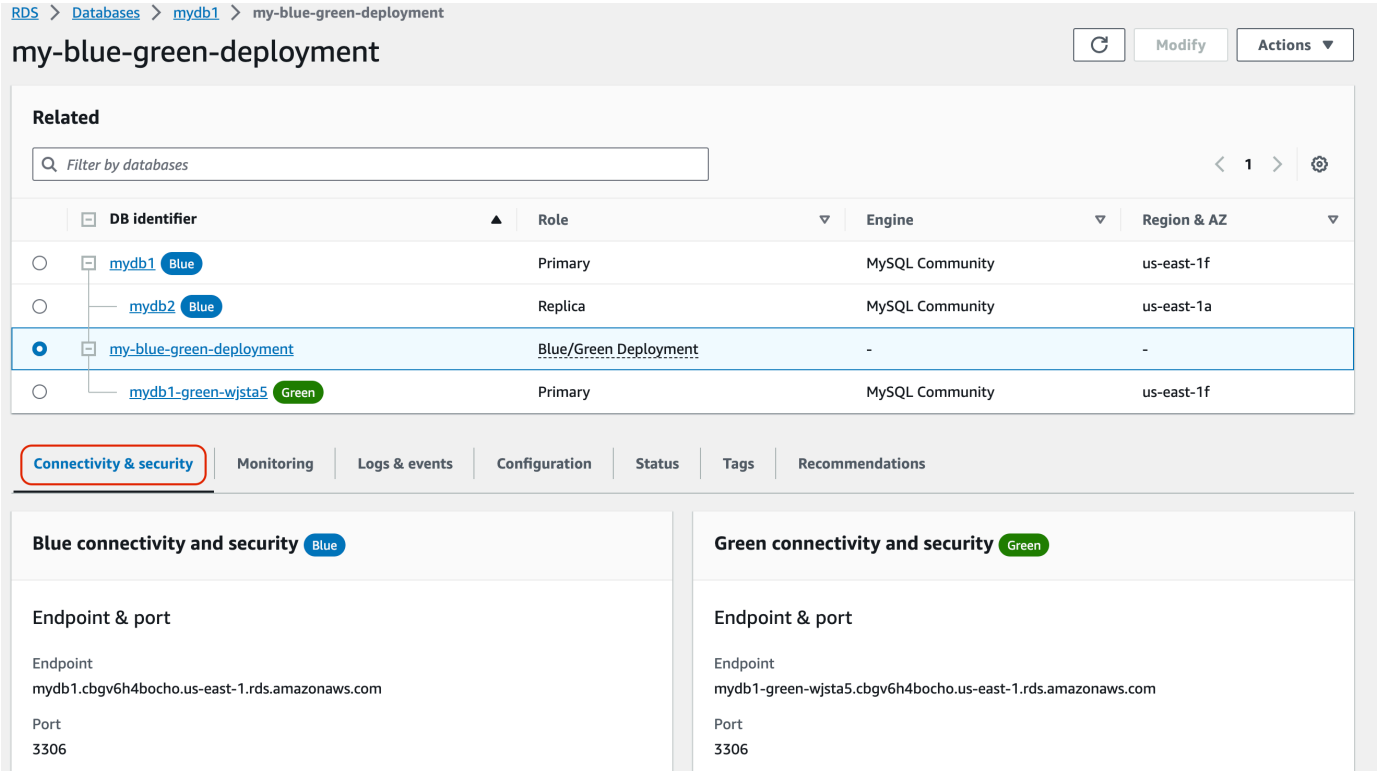


The **Role** value for the blue/green deployment is **Blue/Green Deployment**.

3. Choose the name of blue/green deployment that you want to view to display its details.

Each tab has a section for the blue deployment and a section for the green deployment. For example, on the **Configuration** tab, the DB engine version might be different in the blue environment and in the green environment if you're upgrading the DB engine version in the green environment.

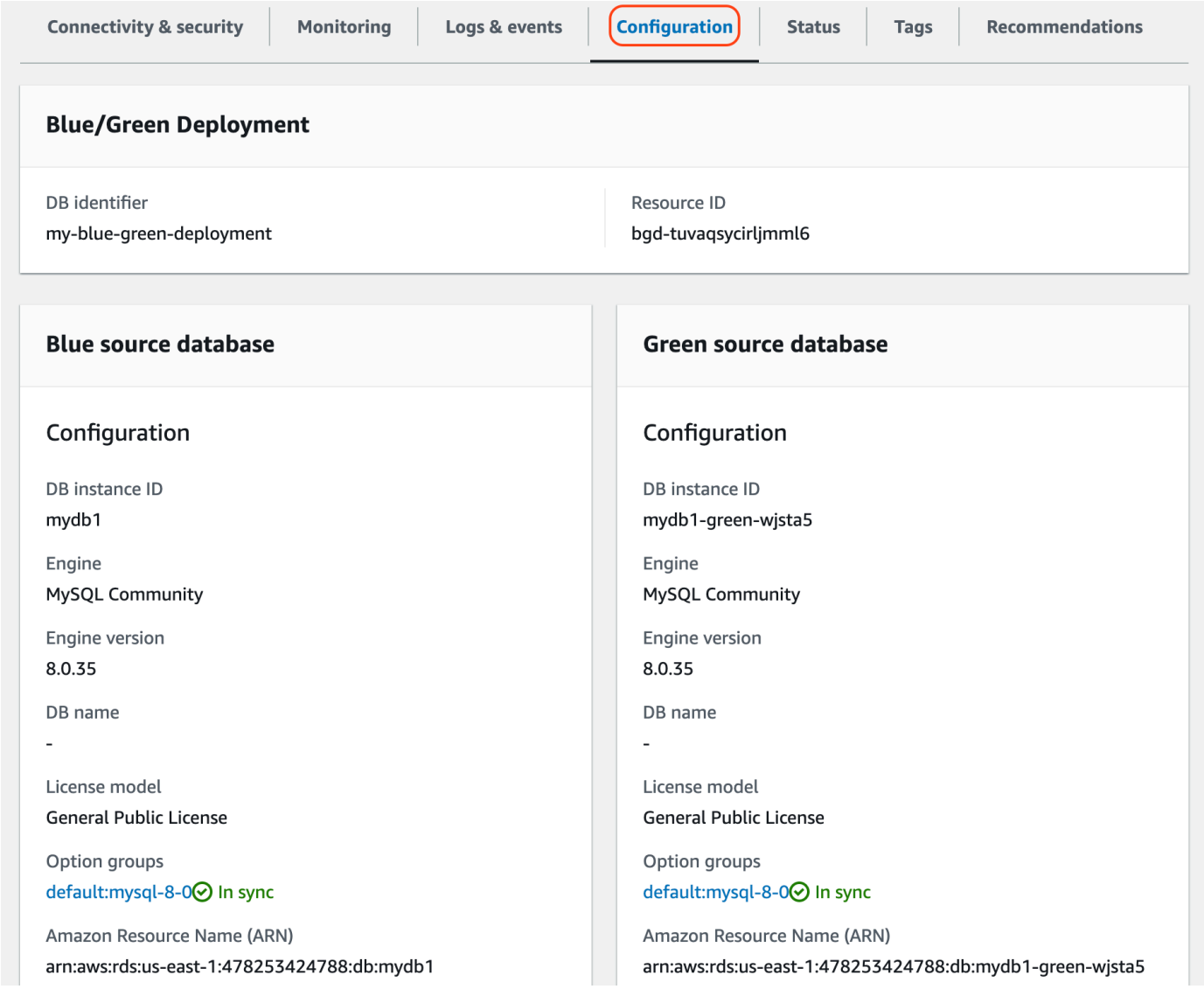
The following image shows an example of the **Connectivity & security** tab:



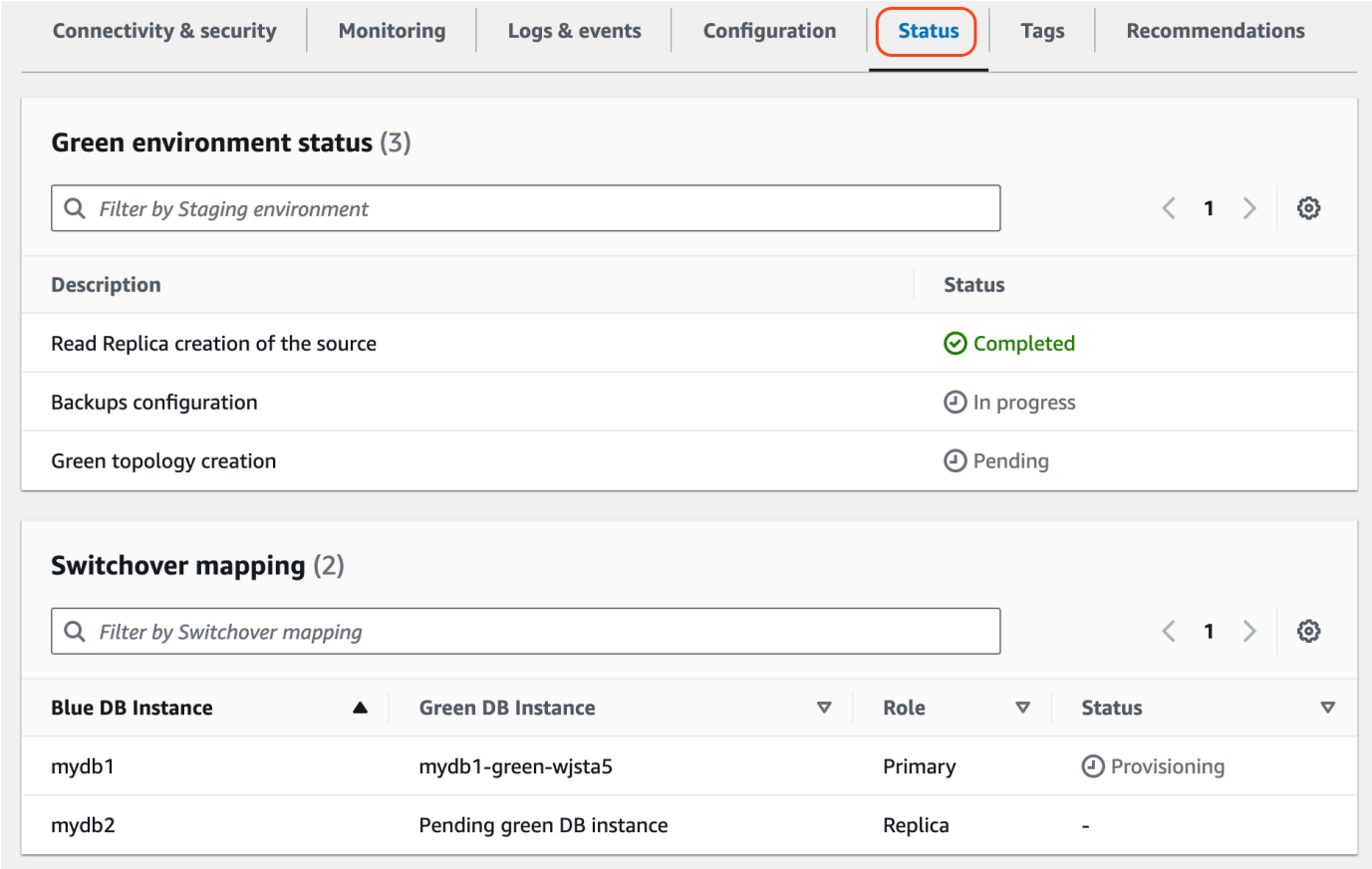
The **Connectivity & security** tab also includes a section called **Replication**, which shows the current state of replication and replica lag between the blue and green environments. If the replication state is `Replicating`, the blue/green deployment is replicating successfully.

For RDS for PostgreSQL blue/green deployments that use logical replication, the replication state can change to `Replication degraded` if you make unsupported DDL or large object changes in the blue environment. For more information, see the section called "Logical replication-specific limitations for blue/green deployments".

The following image shows an example of the **Configuration** tab:



The following image shows an example of the **Status** tab:



AWS CLI

To view the details about a blue/green deployment by using the AWS CLI, use the describe-blue-green-deployments command.

Example View the details about a blue/green deployment by filtering on its name

When you use the describe-blue-green-deployments command, you can filter on the `--blue-green-deployment-name`.

The following example shows the details for a blue/green deployment named *`my-blue-green-deployment`*.

```
aws rds describe-blue-green-deployments \
  --filters Name=blue-green-deployment-name,Values=my-blue-green-deployment
```

Example View the details about a blue/green deployment by specifying its identifier

When you use the describe-blue-green-deployments command, you can specify the `--blue-green-deployment-identifier` option.

The following example shows the details for a blue/green deployment with the identifier *bgd-1234567890abcdef*.

```
aws rds describe-blue-green-deployments \
  --blue-green-deployment-identifier bgd-1234567890abcdef
```

RDS API

To view the details about a blue/green deployment by using the Amazon RDS API, use the `DescribeBlueGreenDeployments` operation and specify the `BlueGreenDeploymentIdentifier`.

Switching a blue/green deployment in Amazon RDS

A *switchover* transitions the green environment to be the new production environment. When the green DB instance has read replicas, they are also transitioned. Before you switch over, production traffic is routed to the DB instance and read replicas in the blue environment. After you switch over, production traffic is routed to the DB instance and read replicas in the green environment.

Switching over a blue/green deployment is not the same as *promoting* the green DB instance within the blue/green deployment. If you manually promote the green DB instance by choosing **Promote** from the **Actions** menu, replication between the blue and green environments breaks and the blue/green deployment enters a state of **Invalid configuration**.

Topics

- Switchover timeout
- Switchover guardrails
- Switchover actions
- Switchover best practices
- Verifying CloudWatch metrics before switchover
- Monitoring replica lag prior to switchover

- Switching over a blue/green deployment
- After switchover

Switchover timeout

You can specify a switchover timeout period between 30 seconds and 3,600 seconds (one hour). If the switchover takes longer than the specified duration, then any changes are rolled back and no changes are made to either environment. The default timeout period is 300 seconds (five minutes).

Switchover guardrails

When you start a switchover, Amazon RDS runs some basic checks to test the readiness of the blue and green environments for switchover. These checks are known as *switchover guardrails*. These switchover guardrails prevent a switchover if the environments aren't ready for it. Therefore, they avoid longer than expected downtime and prevent the loss of data between the blue and green environments that might result if the switchover started.

Amazon RDS runs the following guardrail checks on the green environment:

- **Replication health** – Checks if green primary DB instance replication status is healthy. The green primary DB instance is a replica of the blue primary DB instance.
- **Replication lag** – Checks if the replica lag of the green primary DB instance is within allowable limits for switchover. The allowable limits are based on the specified timeout period. Replica lag indicates how far the green primary DB instance is lagging behind its blue primary DB instance. For more information, see the section called “Monitoring replica lag prior to switchover”.
- **Active writes** – Makes sure there are no active writes on the green primary DB instance.

Amazon RDS runs the following guardrail checks on the blue environment:

- **External replication** – For RDS for PostgreSQL, makes sure that the blue environment isn't a self-managed logical source (publisher) or replica (subscriber). If it is, we recommend that you drop the self-managed replication slots and subscriptions across all databases in the blue environment, proceed with switchover, then recreate them to resume replication. For RDS for MySQL and RDS for MariaDB, checks whether the blue database isn't an external binlog replica. If it is, make sure that it is not actively replicating.
- **Long-running active writes** – Makes sure there are no long-running active writes on the blue primary DB instance because they can increase replica lag.

- **Long-running DDL statements** – Makes sure there are no long-running DDL statements on the blue primary DB instance because they can increase replica lag.
- **Unsupported PostgreSQL changes** – For RDS for PostgreSQL blue/green deployments that use logical replication, makes sure that no DDL changes and no additions or modifications of large objects have been performed on the blue environment. For more information, see the section called “Logical replication-specific limitations for blue/green deployments”.

  If Amazon RDS detects unsupported PostgreSQL changes, it changes the replication state to `Replication degraded` and notifies you that switchover is not available for the blue/green deployment. To proceed with switchover, we recommend that you delete and recreate the blue/green deployment and all green databases. To do so, choose **Actions**, **Delete with green databases**.

Switchover actions

When you switch over a blue/green deployment, RDS performs the following actions:

1. Runs guardrail checks to verify if the blue and green environments are ready for switchover.
2. Stops new write operations on the primary DB instance in both environments.
3. Drops connections to the DB instances in both environments and doesn't allow new connections.
4. Waits for replication to catch up in the green environment so that the green environment is in sync with the blue environment.
5. Renames the DB instances in the both environments.

   RDS renames the DB instances in the green environment to match the corresponding DB instances in the blue environment. For example, assume the name of a DB instance in the blue environment is `mydb`. Also assume the name of the corresponding DB instance in the green environment is `mydb-green-abc123`. During switchover, the name of the DB instance in the green environment is changed to `mydb`.

   RDS renames the DB instances in the blue environment by appending `-old`*n* to the current name, where *n* is a number. For example, assume the name of a DB instance in the blue environment is `mydb`. After switchover, the DB instance name might be `mydb-old1`.

   RDS also renames the endpoints in the green environment to match the corresponding endpoints in the blue environment so that application changes aren't required.
6. Allows connections to databases in both environments.

7. Allows write operations on the primary DB instance in the new production environment.

   After switchover, the previous production primary DB instance only allows read operations until you set the `read_only` parameter (for RDS for MySQL) or the `default_transaction_read_only` parameter (for RDS for PostgreSQL) to `0` and reboot the DB instance.

You can monitor the status of a switchover using Amazon EventBridge. For more information, see the section called "Blue/green deployment events".

During switchover, tags from the blue environment replace all tags on resources in the green environment. Any tags that you added directly to green environment resources are overwritten during this process. For more information about tags, see Tagging Amazon RDS resources.

If the switchover starts and then stops before finishing for any reason, then any changes are rolled back, and no changes are made to either environment.

Switchover best practices

Before you switch over, we strongly recommend that you adhere to best practices by completing the following tasks:

- Thoroughly test the resources in the green environment. Make sure they function properly and efficiently.
- Monitor relevant Amazon CloudWatch metrics. For more information, see the section called "Verifying CloudWatch metrics before switchover".
- Identify the best time for the switchover.

  During the switchover, writes are cut off from databases in both environments. Identify a time when traffic is lowest on your production environment. Long-running transactions, such as active DDLs, can increase your switchover time, resulting in longer downtime for your production workloads.

  If there's a large number of connections on your DB instances, consider manually reducing them to the minimum amount necessary for your application before you switch over the blue/green deployment. One way to achieve this is to create a script that monitors the status of the blue/green deployment and starts cleaning up connections when it detects that the status has changed to `SWITCHOVER_IN_PROGRESS`.

- Make sure the DB instances in both environments are in `Available` state.
- Make sure the primary DB instance in the green environment is healthy and replicating.
- Make sure that your network and client configurations don't increase the DNS cache Time-To-Live (TTL) beyond five seconds, which is the default for RDS DNS zones. Otherwise, applications will continue to send write traffic to the blue environment after switchover.
- Make sure data loading is complete before switching over. For more information, see the section called "Lazy loading and storage initialization".
- For RDS for PostgreSQL blue/green deployments that use logical replication, do the following:
  - Review the logical replication limitations and take any required actions prior to switchover. For more information, see the section called "Logical replication-specific limitations for blue/green deployments".
  - Run the ANALYZE operation to refresh the `pg_statistics` table. This reduces the risk of performance issues after switchover.
  - Before initiating a blue/green deployment switchover, verify that your application does not override the `default_transaction_read_only` parameter at the session level. During switchover, this parameter is set to on on the green environment writer to prevent writes until promotion completes. If your application or transactions override this configuration to `off`, your application may write data to the green environment during the switchover process. In the event the switchover has to roll back, these writes are not available in the blue environment, requiring you to manually resolve the data inconsistencies. We strongly recommend auditing your application queries to ensure they respect the `default_transaction_read_only` setting before proceeding with switchover.

> **Note**
>
> During a switchover, you can't modify any DB instances included in the switchover.

Verifying CloudWatch metrics before switchover

Before you switch over a blue/green deployment, we recommend that you check the value of the following metric within Amazon CloudWatch.

- `DatabaseConnections` – Use this metric to estimate the level of activity on the blue/green deployment, and make sure that the value is at an acceptable level for your deployment before you switch over. If Performance Insights is turned on, `DBLoad` is a more accurate metric.

For more information, see the section called "CloudWatch metrics for RDS".

Monitoring replica lag prior to switchover

Before you switch over a blue/green deployment, make sure that replica lag is close to zero in order to reduce downtime.

RDS for MySQL and RDS for MariaDB

For MySQL and MariaDB blue/green deployments, check the `ReplicaLag` CloudWatch metric in the green environment to identify the current replica lag. For more information, see the section called "Diagnosing and resolving lag between read replicas".

RDS for PostgreSQL

For PostgreSQL blue/green deployments that use physical replication, see the section called "Monitoring and tuning the replication process" for instructions to identify the current replica lag.

For PostgreSQL blue/green deployments that use logical replication, check the `OldestReplicationSlotLag` CloudWatch metric in the blue environment to identify the current replica lag. For more information, see the section called "Amazon CloudWatch instance-level metrics for Amazon RDS".

In addition, you can run the following SQL query in the blue environment:

```
SELECT slot_name,
       confirmed_flush_lsn as flushed,
       pg_current_wal_lsn(),
       (pg_current_wal_lsn() - confirmed_flush_lsn) AS lsn_distance
FROM pg_catalog.pg_replication_slots
WHERE slot_type = 'logical';

slot_name        |    flushed    | pg_current_wal_lsn | lsn_distance
-----------------+---------------+--------------------+------------
logical_replica1 | 47D97/CF32980 | 47D97/CF3BAC8      | 37192
```

The `confirmed_flush_lsn` represents the last log sequence number (LSN) that was sent to the replica. The `pg_current_wal_lsn` represents where the database is now. An `lsn_distance` of 0 means that the replica is caught up.

For an explanation of when blue/green deployments use physical replication versus logical replication, see the section called “PostgreSQL replication methods”.

Switching over a blue/green deployment

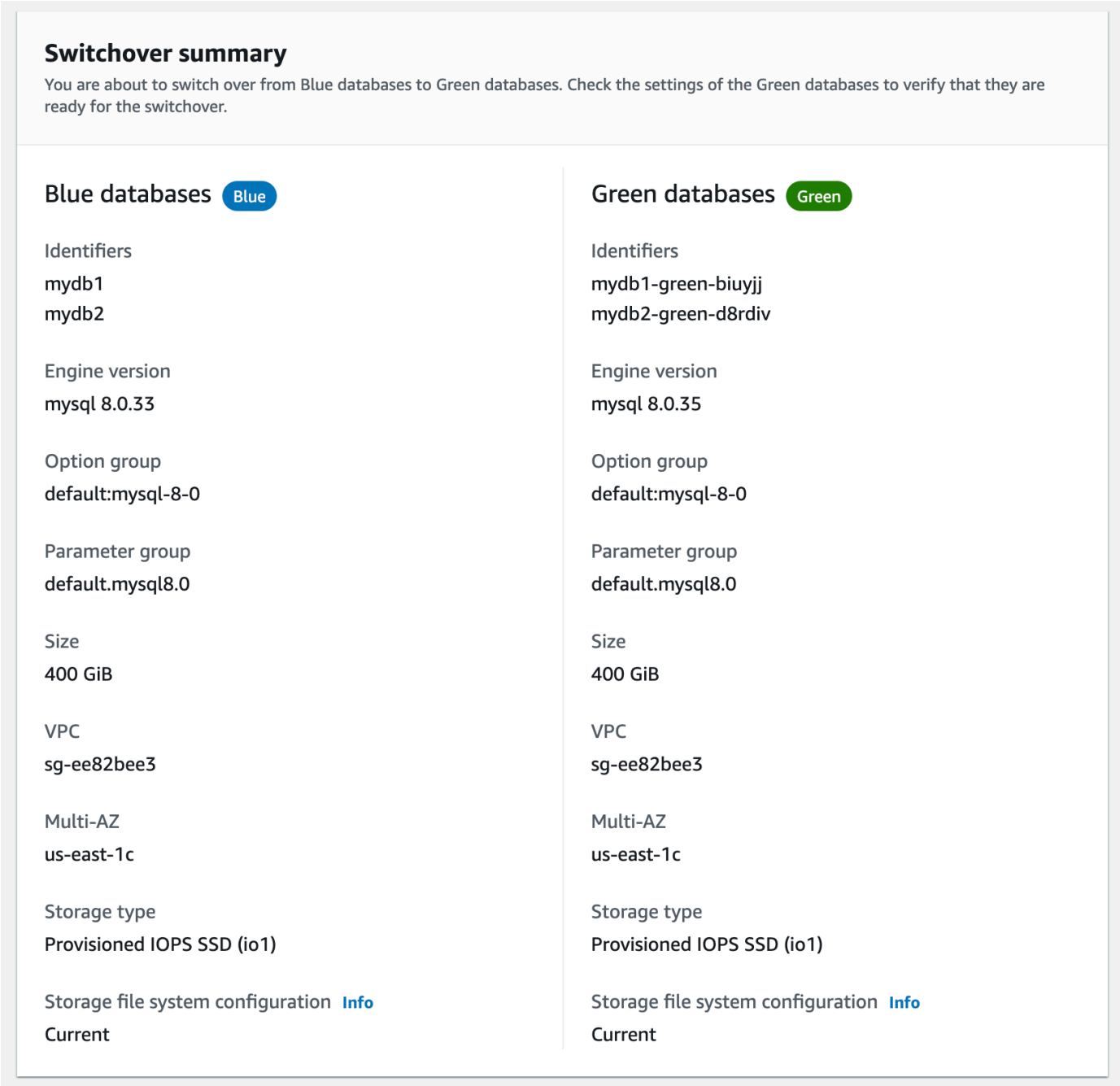
You can switch over a blue/green deployment using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To switch over a blue/green deployment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the blue/green deployment that you want to switch over.
3. For **Actions**, choose **Switch over**.

   The **Switch over** page appears.



4. On the **Switch over** page, review the switchover summary. Make sure the resources in both environments match what you expect. If they don't, choose **Cancel**.
5. For **Timeout settings**, enter the time limit for switchover.
6. If your instance is running RDS for PostgreSQL, review and acknowledge the pre-switchover recommendations. For more information, see the section called “Logical replication-specific limitations for blue/green deployments”.
7. Choose **Switch over**.

AWS CLI

To switch over a blue/green deployment by using the AWS CLI, use the switchover-blue-green-deployment command with the following options:

- `--blue-green-deployment-identifier` – Specify the resource ID of the blue/green deployment.
- `--switchover-timeout` – Specify the time limit for the switchover, in seconds. The default is 300.

Example Switch over a blue/green deployment

For Linux, macOS, or Unix:

```
aws rds switchover-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --switchover-timeout 600
```

For Windows:

```
aws rds switchover-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --switchover-timeout 600
```

RDS API

To switch over a blue/green deployment by using the Amazon RDS API, use the `SwitchoverBlueGreenDeployment` operation with the following parameters:

- `BlueGreenDeploymentIdentifier` – Specify the resource ID of the blue/green deployment.
- `SwitchoverTimeout` – Specify the time limit for the switchover, in seconds. The default is 300.

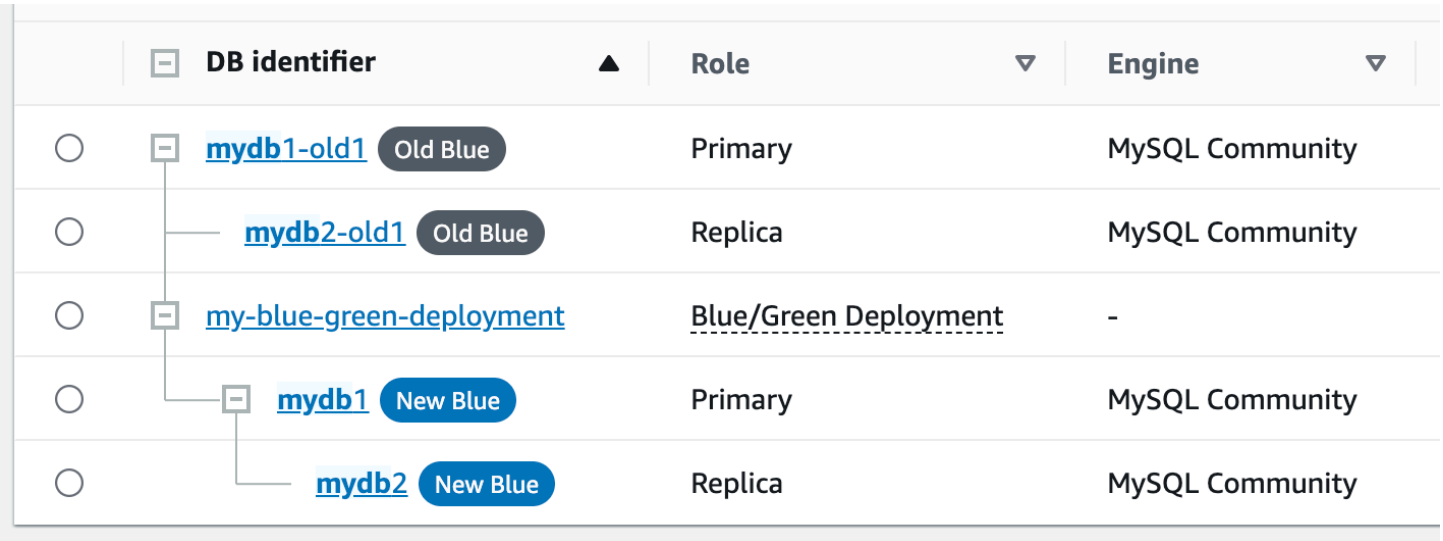
After switchover

After a switchover, the DB instances in the previous blue environment are retained. Standard costs apply to these resources. Replication between the blue and green environments stops.

RDS renames the DB instances in the blue environment by appending `-old`*n* to the current resource name, where *n* is a number. The DB instances in the old blue environment

are read-only until you set the `read_only` parameter (for RDS for MySQL) or the `default_transaction_read_only` parameter (for RDS for PostgreSQL) to `0`. RDS names the DB instances in the green environment `-new`*n*.

If you delete the blue/green deployment resource, RDS retains the `-old`*n* and `-new`*n* resources.



Updating the parent node for consumers

RDS offers fully managed read replicas. However, it also provides the option to set up self-managed replicas, also known as *external replicas.* External replicas allow you to use third-party resources as replication targets.

After you switch over an RDS for MariaDB or RDS for MySQL blue/green deployment, if the blue DB instance had any external replicas or binary log consumers prior to switchover, you must update their parent node after switchover in order to maintain replication continuity.

To update the parent node

1. After switchover, the DB instance that was previously in the green environment emits an event that contains the master log file name and master log position. To locate the event, navigate to the RDS console and choose **Events** from the left navigation pane.
2. Filter by events where the source is the name of the old green DB instance, before switchover.
3. Locate the event that contains the binary log coordinates. The event message is similar to: `Binary log coordinates in green environment after switchover: file mysql-bin-changelog.`*`000003`*` and position `*`40134574`*.

4. Make sure that the consumer or replica has applied all binary logs from the old blue environment. Then, use the provided binary log coordinates to resume replication on the consumers. For example, if you're running a MySQL replica on EC2, you can use the following commands:

   MySQL 8.0.22 and lower major and minor versions

   ```
   CHANGE MASTER TO MASTER_HOST='{new-writer-endpoint}', MASTER_LOG_FILE='mysql-bin-
   changelog.000003', MASTER_LOG_POS=40134574;
   ```

   MySQL 8.0.23 and higher major and minor versions

   ```
   CHANGE REPLICATION SOURCE TO SOURCE_HOST='{new-writer-endpoint}',
    SOURCE_LOG_FILE='mysql-bin-changelog.000003', SOURCE_LOG_POS=40134574;
   ```

If the consumer is another RDS for MySQL or RDS for MariaDB DB instance, run the following stored procedures in order:

1. the section called “mysql.rds_stop_replication”
2. mysql.rds_reset_external_master (for version 8.0 and lower) or mysql_rds_reset_external_source (for version 8.4 and higher)
3. mysql.rds_set_external_master (for version 8.0 and lower) or mysql_rds_set_external_source (for version 8.4 and higher)
4. the section called “mysql.rds_start_replication”

Deleting a blue/green deployment in Amazon RDS

You can delete a blue/green deployment before or after you switch it over.

When you delete a blue/green deployment before switching it over, Amazon RDS optionally deletes the DB instances in the green environment:

- If you choose to delete the DB instances in the green environment (`--delete-target`), they must have deletion protection turned off.
- If you don't delete the DB instances in the green environment (`--no-delete-target`), the instances are retained, but they're no longer part of a blue/green deployment. For RDS for

MySQL, replication continues between the environments. For RDS for PostgreSQL, the green environment is promoted to a standalone environment, so replication stops.

The option to delete the green databases isn't available in the console after switchover. When you delete blue/green deployments using the AWS CLI, you can't specify the `--delete-target` option if the deployment status is `SWITCHOVER_COMPLETED`.

> **Important**
>
> Deleting a blue/green deployment doesn't affect the blue environment.

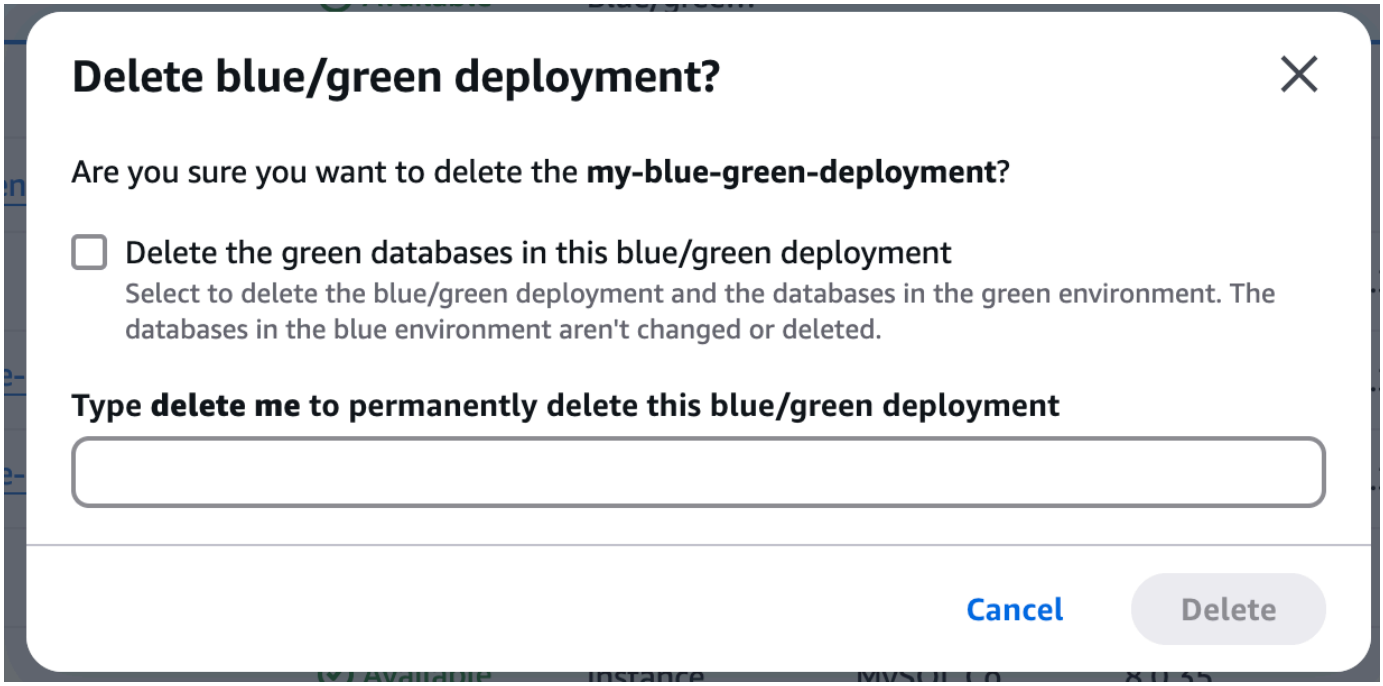
You can delete a blue/green deployment using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To delete a blue/green deployment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the blue/green deployment that you want to delete.
3. For **Actions**, choose **Delete**.

   The **Delete blue/green deployment?** window appears.



To delete the green databases, select **Delete the green databases in this blue/green deployment**.

4. Enter **`delete me`** in the box.
5. Choose **Delete**.

AWS CLI

To delete a blue/green deployment by using the AWS CLI, use the delete-blue-green-deployment command with the following options:

- `--blue-green-deployment-identifier` – The resource ID of the blue/green deployment to be deleted.
- `--delete-target` – Specifies that the DB instances in the green environment are deleted. You can't specify this option if the blue/green deployment has a status of `SWITCHOVER_COMPLETED`.
- `--no-delete-target` – Specifies that the DB instances in the green environment are retained.

Example Delete a blue/green deployment and the DB instances in the green environment

For Linux, macOS, or Unix:

```
aws rds delete-blue-green-deployment \
```

```
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --delete-target
```

For Windows:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --delete-target
```

Example Delete a blue/green deployment but retain the DB instances in the green environment

For Linux, macOS, or Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --no-delete-target
```

For Windows:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --no-delete-target
```

RDS API

To delete a blue/green deployment by using the Amazon RDS API, use the `DeleteBlueGreenDeployment` operation with the following parameters:

- `BlueGreenDeploymentIdentifier` – The resource ID of the blue/green deployment to be deleted.
- `DeleteTarget` – Specify `TRUE` to delete the DB instances in the green environment or `FALSE` to retain them. Cannot be `TRUE` if the blue/green deployment has a status of `SWITCHOVER_COMPLETED`.

Backing up, restoring, and exporting data

This section shows how to back up, restore, and export data from an Amazon RDS DB instance or Multi-AZ DB cluster.

For information about backing up your Amazon RDS DB instance or Multi-AZ DB cluster, see the following topics.

- Introduction to backups
- Managing automated backups
- Managing manual backups

For information about restoring your Amazon RDS DB instance or Multi-AZ DB cluster, see Restoring to a DB instance.

For information about copying, sharing, or exporting DB snapshots, see the following topics.

- Copying a DB snapshot for Amazon RDS
- Sharing a DB snapshot for Amazon RDS
- Exporting DB snapshot data to Amazon S3 for Amazon RDS

For information about viewing automated backups managed by AWS Backup, see Using AWS Backup to manage automated backups for Amazon RDS.

Introduction to backups

Amazon RDS creates and saves automated backups of your DB instance or Multi-AZ DB cluster during the backup window of your DB instance. RDS creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. RDS saves the automated backups of your DB instance according to the backup retention period that you specify. If necessary, you can recover your DB instance to any point in time during the backup retention period.

Automated backups follow these rules:

- Your DB instance must be in the `available` state for automated backups to occur. Automated backups don't occur while your DB instance is in a state other than `available`, for example, `storage_full`.
- Automated backups don't occur while a DB snapshot copy is running in the same AWS Region for the same database.

You can also back up your DB instance manually by creating a DB snapshot. For more information about manually creating a DB snapshot, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Snapshot and backup functionality supports multi-volume configurations. All backup operations include both the primary volume and any additional storage volumes. Snapshots capture the entire database storage configuration. Point-in-time recovery (PITR) works across all storage volumes.

The first snapshot of a DB instance contains the data for the full database. Subsequent snapshots of the same database are incremental, which means that only the data that has changed after your most recent snapshot is saved.

You can copy both automatic and manual DB snapshots, and share manual DB snapshots. For more information about copying a DB snapshot, see Copying a DB snapshot for Amazon RDS. For more information about sharing a DB snapshot, see Sharing a DB snapshot for Amazon RDS.

Backup storage

Your Amazon RDS backup storage for each AWS Region is composed of the automated backups and manual DB snapshots for that Region. Total backup storage space equals the sum of the storage for all backups in that Region. Moving a DB snapshot to another Region increases the backup storage in the destination Region. Backups are stored in Amazon S3.

For more information about backup storage costs, see Amazon RDS pricing.

If you choose to retain automated backups when you delete a DB instance, the automated backups are saved for the full retention period. If you don't choose **Retain automated backups** when you delete a DB instance, all automated backups are deleted with the DB instance. After they are deleted, the automated backups can't be recovered. If you choose to have Amazon RDS create a final DB snapshot before it deletes your DB instance, you can use that to recover your DB instance. Optionally, you can use a previously created manual snapshot. Manual snapshots are not deleted. You can have up to 100 manual snapshots per Region.

Managing automated backups

This section shows how to manage automated backups for DB instances and Multi-AZ DB clusters.

Topics

- Backup window
- Backup retention period
- Enabling automated backups
- Retaining automated backups
- Deleting retained automated backups
- Automated backups with unsupported MySQL storage engines
- Automated backups with unsupported MariaDB storage engines
- Replicating automated backups to another AWS Region

Backup window

Automated backups occur daily during the preferred backup window. If the backup requires more time than allotted to the backup window, the backup continues after the window ends until it finishes. The backup window can't overlap with the weekly maintenance window for the DB instance or Multi-AZ DB cluster.

During the automatic backup window, storage I/O might be suspended briefly while the backup process initializes (typically under a few seconds). You might experience elevated latencies for a few minutes during backups for Multi-AZ deployments. For MariaDB, MySQL, Oracle, and PostgreSQL, I/O activity isn't suspended on your primary during backup for Multi-AZ deployments because the backup is taken from the standby. For SQL Server, I/O activity is suspended briefly during backup for both Single-AZ and Multi-AZ deployments because the backup is taken from the primary. For Db2, I/O activity is also suspended briefly during backup even though the backup is taken from the standby.

Automated backups might occasionally be skipped if the DB instance or cluster has a heavy workload at the time a backup is supposed to start. If a backup is skipped, you can still do a point-in-time-recovery (PITR), and a backup is still attempted during the next backup window. For more information on PITR, see Restoring a DB instance to a specified time for Amazon RDS.

If you don't specify a preferred backup window when you create the DB instance or Multi-AZ DB cluster, Amazon RDS assigns a default 30-minute backup window. This window is selected at random from an 8-hour block of time for each AWS Region. The following table lists the time blocks for each AWS Region from which the default backup windows are assigned.

Region Name	Region	Time Block
US East (N. Virginia)	us-east-1	03:00–11:00 UTC
US East (Ohio)	us-east-2	03:00–11:00 UTC
US West (N. California)	us-west-1	06:00–14:00 UTC
US West (Oregon)	us-west-2	06:00–14:00 UTC
Africa (Cape Town)	af-south-1	03:00–11:00 UTC
Asia Pacific (Hong Kong)	ap-east-1	06:00–14:00 UTC
Asia Pacific (Hyderabad)	ap-south-2	06:30–14:30 UTC
Asia Pacific (Jakarta)	ap-southeast-3	08:00–16:00 UTC
Asia Pacific (Malaysia)	ap-southeast-5	09:00–17:00 UTC
Asia Pacific (Melbourne)	ap-southeast-4	11:00–19:00 UTC
Asia Pacific (Mumbai)	ap-south-1	16:30–00:30 UTC
Asia Pacific (New Zealand)	ap-southeast-6	13:00–21:00 UTC
Asia Pacific (Osaka)	ap-northeast-3	00:00–08:00 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00–21:00 UTC

Region Name	Region	Time Block
Asia Pacific (Singapore)	ap-southeast-1	14:00–22:00 UTC
Asia Pacific (Sydney)	ap-southeast-2	12:00–20:00 UTC
Asia Pacific (Taipei)	ap-east-2	9:00–17:00 UTC
Asia Pacific (Thailand)	ap-southeast-7	8:00–16:00 UTC
Asia Pacific (Tokyo)	ap-northeast-1	13:00–21:00 UTC
Canada (Central)	ca-central-1	03:00–11:00 UTC
Canada West (Calgary)	ca-west-1	18:00–02:00 UTC
China (Beijing)	cn-north-1	06:00–14:00 UTC
China (Ningxia)	cn-northwest-1	06:00–14:00 UTC
Europe (Frankfurt)	eu-central-1	20:00–04:00 UTC
Europe (Ireland)	eu-west-1	22:00–06:00 UTC
Europe (London)	eu-west-2	22:00–06:00 UTC
Europe (Milan)	eu-south-1	02:00–10:00 UTC
Europe (Paris)	eu-west-3	07:29–14:29 UTC
Europe (Spain)	eu-south-2	02:00–10:00 UTC
Europe (Stockholm)	eu-north-1	23:00–07:00 UTC
Europe (Zurich)	eu-central-2	02:00–10:00 UTC
Israel (Tel Aviv)	il-central-1	03:00–11:00 UTC
Mexico (Central)	mx-central-1	19:00–03:00 UTC

Region Name	Region	Time Block
Middle East (Bahrain)	me-south-1	06:00–14:00 UTC
Middle East (UAE)	me-central-1	05:00–13:00 UTC
South America (São Paulo)	sa-east-1	23:00–07:00 UTC
AWS GovCloud (US-East)	us-gov-east-1	17:00–01:00 UTC
AWS GovCloud (US-West)	us-gov-west-1	06:00–14:00 UTC

Backup retention period

You can set the backup retention period when you create or restore a DB instance or Multi-AZ DB cluster. If you create a DB instance using the Amazon RDS API or the AWS CLI and if you don't set the backup retention period, the default backup retention period is one day. If you create a DB instance using the console, the default backup retention period is seven days.

After you create a DB instance or cluster, you can modify the backup retention period. You can set the backup retention period of a DB instance to between 0 and 35 days. Setting the backup retention period to 0 disables automated backups. For a Multi-AZ DB cluster, you can set the backup retention period to between 1 and 35 days. Manual snapshot limits (100 per Region) don't apply to automated backups.

During restore operations, you have the option to specify a backup retention period for your DB instance or Multi-AZ DB cluster. When you don't explicitly set this value, the restored database inherits the backup retention period from the source snapshot or instance. Note that this inheritance behavior is unique to restore operations—when creating a new database, the system applies default retention periods instead.

> **Important**
>
> An outage occurs if you change the backup retention period of a DB instance from 0 to a nonzero value or from a nonzero value to 0.

RDS doesn't include time spent in the `stopped` state when the backup retention period is calculated. Automated backups aren't created while a DB instance or cluster is stopped. Backups can be retained longer than the backup retention period if a DB instance has been stopped.

Enabling automated backups

If your DB instance doesn't have automated backups enabled, you can enable them at any time. You enable automated backups by setting the backup retention period to a positive nonzero value. When automated backups are turned on, your DB instance is taken offline and a backup is immediately created.

> **Note**
>
> If you manage your backups in AWS Backup, you can't enable automated backups. For more information, see Using AWS Backup to manage automated backups for Amazon RDS.

Console

To enable automated backups immediately

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance or Multi-AZ DB cluster that you want to modify.
3. Choose **Modify**.
4. For **Backup retention period**, choose a positive nonzero value, for example three days.
5. Choose **Continue**.
6. Choose **Apply immediately**.
7. Choose **Modify DB instance** or **Modify cluster** to save your changes and enable automated backups.

AWS CLI

To enable automated backups, use the AWS CLI modify-db-instance or modify-db-cluster command.

Include the following parameters:

- `--db-instance-identifier` (or `--db-cluster-identifier` for a Multi-AZ DB cluster)
- `--backup-retention-period`
- `--apply-immediately` or `--no-apply-immediately`

In the following example, we enable automated backups by setting the backup retention period to three days. The changes are applied immediately.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my_db_instance  \
    --backup-retention-period 3 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my_db_instance  ^
    --backup-retention-period 3 ^
    --apply-immediately
```

RDS API

To enable automated backups, use the RDS API ModifyDBInstance or ModifyDBCluster operation with the following required parameters:

- `DBInstanceIdentifier` or `DBClusterIdentifier`
- `BackupRetentionPeriod`

Viewing automated backups

To view your automated backups, choose **Automated backups** in the navigation pane. To view individual snapshots associated with an automated backup, choose **Snapshots** in the navigation pane. Alternatively, you can describe individual snapshots associated with an automated backup. From there, you can restore a DB instance directly from one of those snapshots.

Automated snapshot names follow the pattern `rds:<database-name>-yyyy-mm-dd-hh-mm`, with `yyyy-mm-dd-hh-mm` representing the date and time the snapshot was created.

To describe the automated backups for your existing DB instances using the AWS CLI, use one of the following commands:

```
aws rds describe-db-instance-automated-backups --db-instance-
identifier DBInstanceIdentifier
```

or

```
aws rds describe-db-instance-automated-backups --dbi-resource-id DbiResourceId
```

To describe the retained automated backups for your existing DB instances using the RDS API, call the `DescribeDBInstanceAutomatedBackups` action with one of the following parameters:

- `DBInstanceIdentifier`
- `DbiResourceId`

Retaining automated backups

> **Note**
>
> You can only retain automated backups of DB instances, not Multi-AZ DB clusters.

When you delete a DB instance, you can choose to retain automated backups. Automated backups can be retained for a number of days equal to the backup retention period configured for the DB instance at the time when you delete it.

Retained automated backups contain system snapshots and transaction logs from a DB instance. They also include your DB instance properties like allocated storage and DB instance class, which are required to restore it to an active instance.

Retained automated backups and manual snapshots incur billing charges until they're deleted. For more information, see Retention costs.

You can retain automated backups for RDS instances running the Db2, MariaDB, MySQL, PostgreSQL, Oracle, and Microsoft SQL Server engines.

You can restore or remove retained automated backups using the AWS Management Console, RDS API, and AWS CLI.

Topics

- Retention period
- Viewing retained backups
- Restoration
- Retention costs
- Limitations

Retention period

The system snapshots and transaction logs in a retained automated backup expire the same way that they expire for the source DB instance. Because there are no new snapshots or logs created for this instance, the retained automated backups eventually expire completely. Effectively, they live as long their last system snapshot would have done, based on the settings for retention period the source instance had when you deleted it. Retained automated backups are removed by the system after their last system snapshot expires.

You can remove a retained automated backup in the same way that you can delete a DB instance. You can remove retained automated backups using the console or the RDS API operation `DeleteDBInstanceAutomatedBackup`.

Final snapshots are independent of retained automated backups. We strongly suggest that you take a final snapshot even if you retain automated backups because the retained automated backups eventually expire. The final snapshot doesn't expire.

Viewing retained backups

To view your retained automated backups, choose **Automated backups** in the navigation pane, then choose **Retained**. To view individual snapshots associated with a retained automated backup, choose **Snapshots** in the navigation pane. Alternatively, you can describe individual snapshots associated with a retained automated backup. From there, you can restore a DB instance directly from one of those snapshots.

To describe your retained automated backups using the AWS CLI, use the following command:

```
aws rds describe-db-instance-automated-backups --dbi-resource-id DbiResourceId
```

To describe your retained automated backups using the RDS API, call the `DescribeDBInstanceAutomatedBackups` action with the `DbiResourceId` parameter.

Restoration

For information on restoring DB instances from automated backups, see Restoring a DB instance to a specified time for Amazon RDS.

Retention costs

The cost of a retained automated backup is the cost of total storage of the system snapshots that are associated with it. There is no additional charge for transaction logs or instance metadata. All other pricing rules for backups apply to restorable instances.

For example, suppose that your total allocated storage of running instances is 100 GB. Suppose also that you have 50 GB of manual snapshots plus 75 GB of system snapshots associated with a retained automated backup. In this case, you are charged only for the additional 25 GB of backup storage, like this: (50 GB + 75 GB) – 100 GB = 25 GB.

Limitations

The following limitations apply to retained automated backups:

- The maximum number of retained automated backups in one AWS Region is 40. It's not included in the DB instances quota. You can have 40 running DB instances and an additional 40 retained automated backups at the same time.
- Retained automated backups don't contain information about parameters or option groups.
- You can restore a deleted instance to a point in time that is within the retention period at the time of deletion.
- You can't modify a retained automated backup. That's because it consists of system backups, transaction logs, and the DB instance properties that existed at the time that you deleted the source instance.

Deleting retained automated backups

You can delete retained automated backups when they are no longer needed.

Console

To delete a retained automated backup

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Automated backups**.
3. On the **Retained** tab, choose the retained automated backup that you want to delete.
4. For **Actions**, choose **Delete**.
5. On the confirmation page, enter **`delete me`** and choose **Delete**.

AWS CLI

You can delete a retained automated backup by using the AWS CLI command delete-db-instance-automated-backup with the following option:

- `--dbi-resource-id` – The resource identifier for the source DB instance.

  You can find the resource identifier for the source DB instance of a retained automated backup by running the AWS CLI command describe-db-instance-automated-backups.

Example

The following example deletes the retained automated backup with source DB instance resource identifier `db-123ABCEXAMPLE`.

For Linux, macOS, or Unix:

```
aws rds delete-db-instance-automated-backup \
    --dbi-resource-id db-123ABCEXAMPLE
```

For Windows:

```
aws rds delete-db-instance-automated-backup ^
    --dbi-resource-id db-123ABCEXAMPLE
```

RDS API

You can delete a retained automated backup by using the Amazon RDS API operation DeleteDBInstanceAutomatedBackup with the following parameter:

- `DbiResourceId` – The resource identifier for the source DB instance.

  You can find the resource identifier for the source DB instance of a retained automated backup using the Amazon RDS API operation DescribeDBInstanceAutomatedBackups.

Disabling automated backups

You might want to temporarily disable automated backups in certain situations, for example while loading large amounts of data.

> **Important**
>
> We highly discourage disabling automated backups because it disables point-in-time recovery. Disabling automatic backups for a DB instance or Multi-AZ DB cluster deletes all existing automated backups for the database. If you disable and then re-enable automated backups, you can restore starting only from the time you re-enabled automated backups.

Console

To disable automated backups immediately

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance or Multi-AZ DB cluster that you want to modify.
3. Choose **Modify**.
4. For **Backup retention period**, choose **0 days**.
5. Choose **Continue**.
6. Choose **Apply immediately**.
7. Choose **Modify DB instance** or **Modify cluster** to save your changes and disable automated backups.

AWS CLI

To disable automated backups immediately, use the modify-db-instance or modify-db-cluster command and set the backup retention period to 0 with `--apply-immediately`.

Example

The following example immediately disables automatic backups on a Multi-AZ DB cluster.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --backup-retention-period 0 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --backup-retention-period 0 ^
    --apply-immediately
```

To know when the modification is in effect, call `describe-db-instances` for the DB instance (or `describe-db-clusters` for a Multi-AZ DB cluster) until the value for backup retention period is 0 and `mydbcluster` status is available.

```
aws rds describe-db-clusters --db-cluster-identifier mydcluster
```

RDS API

To disable automated backups immediately, call the ModifyDBInstance or ModifyDBCluster operation with the following parameters:

- `DBInstanceIdentifier = mydbinstance` (or `DBClusterIdentifier = mydbcluster`)
- `BackupRetentionPeriod = 0`

Example

```
https://rds.amazonaws.com/
```

```
    ?Action=ModifyDBInstance
    &DBInstanceIdentifier=mydbinstance
    &BackupRetentionPeriod=0
    &SignatureVersion=2
    &SignatureMethod=HmacSHA256
    &Timestamp=2009-10-14T17%3A48%3A21.746Z
    &AWSAccessKeyId=<&AWS; Access Key ID>
    &Signature=<Signature>
```

Automated backups with unsupported MySQL storage engines

For the MySQL DB engine, automated backups are only supported for the InnoDB storage engine. Using these features with other MySQL storage engines, including MyISAM, can lead to unreliable behavior when you're restoring from backups. Specifically, since storage engines like MyISAM don't support reliable crash recovery, your tables can be corrupted in the event of a crash. For this reason, we encourage you to use the InnoDB storage engine.

- To convert existing MyISAM tables to InnoDB tables, you can use the `ALTER TABLE` command, for example: `ALTER TABLE` *`table_name`* `ENGINE=innodb, ALGORITHM=COPY;`
- If you choose to use MyISAM, you can attempt to manually repair tables that become damaged after a crash by using the `REPAIR` command. For more information, see REPAIR TABLE statement in the MySQL documentation. However, as noted in the MySQL documentation, there is a good chance that you might not be able to recover all your data.
- If you want to take a snapshot of your MyISAM tables before restoring, follow these steps:
  1. Stop all activity to your MyISAM tables (that is, close all sessions).

     You can close all sessions by calling the mysql.rds_kill command for each process that is returned from the `SHOW FULL PROCESSLIST` command.
  2. Lock and flush each of your MyISAM tables. For example, the following commands lock and flush two tables named `myisam_table1` and `myisam_table2`:

     ```
     mysql> FLUSH TABLES myisam_table, myisam_table2 WITH READ LOCK;
     ```
  3. Create a snapshot of your DB instance or Multi-AZ DB cluster. When the snapshot has completed, release the locks and resume activity on the MyISAM tables. You can release the locks on your tables using the following command:

     ```
     mysql> UNLOCK TABLES;
     ```

These steps force MyISAM to flush data stored in memory to disk, which ensures a clean start when you restore from a DB snapshot. For more information on creating a DB snapshot, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Automated backups with unsupported MariaDB storage engines

For the MariaDB DB engine, automated backups are only supported with the InnoDB storage engine. Using these features with other MariaDB storage engines, including Aria, can lead to unreliable behavior when you're restoring from backups. Even though Aria is a crash-resistant alternative to MyISAM, your tables can still be corrupted in the event of a crash. For this reason, we encourage you to use the InnoDB storage engine.

- To convert existing Aria tables to InnoDB tables, you can use the `ALTER TABLE` command. For example: `ALTER TABLE table_name ENGINE=innodb, ALGORITHM=COPY;`
- If you choose to use Aria, you can attempt to manually repair tables that become damaged after a crash by using the `REPAIR TABLE` command. For more information, see http://mariadb.com/kb/en/mariadb/repair-table/.
- If you want to take a snapshot of your Aria tables before restoring, follow these steps:
  1. Stop all activity to your Aria tables (that is, close all sessions).
  2. Lock and flush each of your Aria tables.
  3. Create a snapshot of your DB instance or Multi-AZ DB cluster. When the snapshot has completed, release the locks and resume activity on the Aria tables. These steps force Aria to flush data stored in memory to disk, thereby ensuring a clean start when you restore from a DB snapshot.

Replicating automated backups to another AWS Region

For added disaster recovery capability, you can configure your Amazon RDS database instance to replicate snapshots and transaction logs to a destination AWS Region of your choice. When backup replication is configured for a DB instance, RDS initiates a cross-Region copy of all snapshots and transaction logs as soon as they are ready on the DB instance.

DB snapshot copy charges apply to the data transfer. After the DB snapshot is copied, standard charges apply to storage in the destination Region. For more details, see RDS Pricing.

For an example of using backup replication, see the AWS online tech talk Managed Disaster Recovery with Amazon RDS for Oracle Cross-Region Automated Backups.

> **Note**
>
> Automated backup replication isn't supported for Multi-AZ DB clusters. However, it is supported for Multi-AZ DB instance deployments. For more information about limitations for automated backups, see Quotas and constraints for Amazon RDS.

For information about configuring and managing automated backups for Amazon RDS, see the following topics.

Topics

- Enabling cross-Region automated backups for Amazon RDS
- Finding information about replicated backups for Amazon RDS
- Restoring to a specified time from a replicated backup for Amazon RDS
- Stopping automated backup replication for Amazon RDS
- Deleting replicated backups for Amazon RDS
- Troubleshooting stopped cross-Region automated backups

Multi-AZ deployment support

Cross-Region automated backup replication is supported for Multi-AZ DB instance deployments for the following engines:

- RDS for Db2

- RDS for MariaDB
- RDS for MySQL
- RDS for Oracle
- RDS for PostgreSQL
- RDS for SQL Server

Cross-Region automated backup replication isn't supported for Multi-AZ DB clusters.

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability with cross-Region automated backups, see Supported Regions and DB engines for cross-Region automated backups in Amazon RDS.

Source and destination AWS Region support

Backup replication is supported between the following AWS Regions.

Source Region	Destination Regions available
Africa (Cape Town)	Europe (Frankfurt), Europe (Ireland), Europe (London)
Asia Pacific (Hong Kong)	Asia Pacific (Singapore), Asia Pacific (Tokyo)
Asia Pacific (Hyderabad)	Asia Pacific (Mumbai)
Asia Pacific (Jakarta)	Asia Pacific (Malaysia), Asia Pacific (Singapore)
Asia Pacific (Melbourne)	Asia Pacific (Sydney)
Asia Pacific (Malaysia)	Asia Pacific (Singapore)
Asia Pacific (Mumbai)	Asia Pacific (Hyderabad), Asia Pacific (Singapore) US East (N. Virginia), US East (Ohio), US West (Oregon)
Asia Pacific (Osaka)	Asia Pacific (Tokyo)

Source Region	Destination Regions available
Asia Pacific (Seoul)	Asia Pacific (Singapore), Asia Pacific (Tokyo) US East (N. Virginia), US East (Ohio), US West (Oregon)
Asia Pacific (Singapore)	Asia Pacific (Hong Kong), Asia Pacific (Malaysia), Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Sydney), Asia Pacific (Tokyo) US East (N. Virginia), US East (Ohio), US West (Oregon)
Asia Pacific (Sydney)	Asia Pacific (Melbourne), Asia Pacific (Singapore) US East (N. Virginia), US West (N. California), US West (Oregon)
Asia Pacific (Tokyo)	Asia Pacific (Hong Kong), Asia Pacific (Osaka), Asia Pacific (Seoul), Asia Pacific (Singapore) US East (N. Virginia), US East (Ohio), US West (Oregon)
Canada (Central)	Canada West (Calgary) Europe (Ireland) US East (N. Virginia), US East (Ohio), US West (N. California), US West (Oregon)
Canada West (Calgary)	Canada (Central)
China (Beijing)	China (Ningxia)
China (Ningxia)	China (Beijing)
Europe (Frankfurt)	Africa (Cape Town) Europe (Ireland), Europe (London), Europe (Paris), Europe (Stockholm), Europe (Zurich) US East (N. Virginia), US East (Ohio), US West (Oregon)

Source Region	Destination Regions available
Europe (Ireland)	Africa (Cape Town) Canada (Central) Europe (Frankfurt), Europe (London), Europe (Paris), Europe (Stockholm), Europe (Zurich) US East (N. Virginia), US East (Ohio), US West (N. California), US West (Oregon)
Europe (London)	Africa (Cape Town) Europe (Frankfurt), Europe (Ireland), Europe (Paris), Europe (Stockholm) US East (N. Virginia)
Europe (Milan)	Europe (Frankfurt)
Europe (Paris)	Europe (Frankfurt), Europe (Ireland), Europe (London), Europe (Stockholm) US East (N. Virginia)
Europe (Spain)	Europe (Ireland), Europe (Paris)
Europe (Stockholm)	Europe (Frankfurt), Europe (Ireland), Europe (London), Europe (Paris) US East (N. Virginia)
Europe (Zurich)	Europe (Frankfurt), Europe (Ireland)
Israel (Tel Aviv)	Europe (Ireland)
Middle East (UAE)	Europe (Frankfurt)
South America (São Paulo)	US East (N. Virginia), US East (Ohio)

Source Region	Destination Regions available
AWS GovCloud (US-East)	AWS GovCloud (US-West)
AWS GovCloud (US-West)	AWS GovCloud (US-East)
US East (N. Virginia)	Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo) Canada (Central) Europe (Frankfurt), Europe (Ireland), Europe (London), Europe (Paris), Europe (Stockholm) South America (São Paulo) US East (Ohio), US West (N. California), US West (Oregon)
US East (Ohio)	Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Tokyo) Canada (Central) Europe (Frankfurt), Europe (Ireland) South America (São Paulo) US East (N. Virginia), US West (N. California), US West (Oregon)
US West (N. California)	Asia Pacific (Sydney) Canada (Central) Europe (Ireland) US East (N. Virginia), US East (Ohio), US West (Oregon)

Source Region	Destination Regions available
US West (Oregon)	Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo) Canada (Central) Europe (Frankfurt), Europe (Ireland) US East (N. Virginia), US East (Ohio), US West (N. California)

You can also use the `describe-source-regions` AWS CLI command to find out which AWS Regions can replicate to each other. For more information, see Finding information about replicated backups for Amazon RDS.

Limitations

Following are limitations for cross–Region automated backups for Amazon RDS.

- Automated backup replication isn't supported for Multi-AZ DB clusters.
- By default, you can have up to 20 cross–Region automated backups per AWS account.

Enabling cross-Region automated backups for Amazon RDS

You can enable backup replication on new or existing DB instances using the Amazon RDS console. You can also use the `start-db-instance-automated-backups-replication` AWS CLI command or the `StartDBInstanceAutomatedBackupsReplication` RDS API operation. You can replicate up to 20 backups to each destination AWS Region for each AWS account.

> **Note**
>
> To be able to replicate automated backups, make sure to enable them. For more information, see Enabling automated backups.

Console

You can enable backup replication for a new or existing DB instance:

- For a new DB instance, enable it when you launch the instance. For more information, see Settings for DB instances.
- For an existing DB instance, use the following procedure.

To enable backup replication for an existing DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Automated backups**.
3. On the **Current Region** tab, choose the DB instance for which you want to enable backup replication.
4. For **Actions**, choose **Manage cross-Region replication**.
5. Under **Backup replication**, choose **Enable replication to another AWS Region**.
6. Choose the **Destination Region**.
7. Choose the **Replicated backup retention period**.
8. If you've enabled encryption on the source DB instance, choose the **AWS KMS key** for encrypting the backups or enter a key ARN.
9. Choose **Save**.

In the source Region, replicated backups are listed on the **Current Region** tab of the **Automated backups** page. In the destination Region, replicated backups are listed on the **Replicated backups** tab of the **Automated backups** page.

AWS CLI

Enable backup replication by using the `start-db-instance-automated-backups-replication` AWS CLI command.

The following CLI example replicates automated backups from a DB instance in the US West (Oregon) Region to the US East (N. Virginia) Region. It also encrypts the replicated backups, using an AWS KMS key in the destination Region.

To enable backup replication

- Run one of the following commands.

For Linux, macOS, or Unix:

```
aws rds start-db-instance-automated-backups-replication \
--region us-east-1 \
--source-db-instance-arn "arn:aws:rds:us-west-2:123456789012:db:mydatabase" \
--kms-key-id "arn:aws:kms:us-east-1:123456789012:key/AKIAIOSFODNN7EXAMPLE" \
--backup-retention-period 7
```

For Windows:

```
aws rds start-db-instance-automated-backups-replication ^
--region us-east-1 ^
--source-db-instance-arn "arn:aws:rds:us-west-2:123456789012:db:mydatabase" ^
--kms-key-id "arn:aws:kms:us-east-1:123456789012:key/AKIAIOSFODNN7EXAMPLE" ^
--backup-retention-period 7
```

The `--source-region` option is required when you encrypt backups between the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions. For `--source-region`, specify the AWS Region of the source DB instance.

If `--source-region` isn't specified, make sure to specify a `--pre-signed-url` value. A *presigned URL* is a URL that contains a Signature Version 4 signed request for the `start-db-instance-automated-backups-replication` command that is called in the source AWS Region. To learn more about the `pre-signed-url` option, see start-db-instance-automated-backups-replication in the *AWS CLI Command Reference*.

RDS API

Enable backup replication by using the `StartDBInstanceAutomatedBackupsReplication` RDS API operation with the following parameters:

- `Region` (if you aren't calling the API operation from the destination Region)
- `SourceDBInstanceArn`
- `BackupRetentionPeriod`
- `KmsKeyId` (optional)
- `PreSignedUrl` (required if you use `KmsKeyId`)

Note

If you encrypt the backups, you must also include a presigned URL. For more information on presigned URLs, see Authenticating Requests: Using Query Parameters (AWS Signature Version 4) in the *Amazon Simple Storage Service API Reference* and Signature Version 4 signing process in the *AWS General Reference*.

Finding information about replicated backups for Amazon RDS

You can use the following CLI commands to find information about replicated backups:

- `describe-source-regions`
- `describe-db-instances`
- `describe-db-instance-automated-backups`

The following `describe-source-regions` example lists the source AWS Regions from which automated backups can be replicated to the US West (Oregon) destination Region.

To show information about source Regions

- Run the following command.

```
aws rds describe-source-regions --region us-west-2
```

The output shows that backups can be replicated from US East (N. Virginia), but not from US East (Ohio) or US West (N. California), into US West (Oregon).

```
{
    "SourceRegions": [
        ...
        {
            "RegionName": "us-east-1",
            "Endpoint": "https://rds.us-east-1.amazonaws.com",
            "Status": "available",
            "SupportsDBInstanceAutomatedBackupsReplication": true
        },
        {
```

```
            "RegionName": "us-east-2",
            "Endpoint": "https://rds.us-east-2.amazonaws.com",
            "Status": "available",
            "SupportsDBInstanceAutomatedBackupsReplication": false
        },
            "RegionName": "us-west-1",
            "Endpoint": "https://rds.us-west-1.amazonaws.com",
            "Status": "available",
            "SupportsDBInstanceAutomatedBackupsReplication": false
        }
    ]
}
```

The following `describe-db-instances` example shows the automated backups for a DB instance.

To show the replicated backups for a DB instance

- Run one of the following commands.

  For Linux, macOS, or Unix:

  ```
  aws rds describe-db-instances \
  --db-instance-identifier mydatabase
  ```

  For Windows:

  ```
  aws rds describe-db-instances ^
  --db-instance-identifier mydatabase
  ```

The output includes the replicated backups.

```
{
    "DBInstances": [
        {
            "StorageEncrypted": false,
            "Endpoint": {
                "HostedZoneId": "Z1PVIF0B656C1W",
                "Port": 1521,
            ...
```

```
            "BackupRetentionPeriod": 7,
            "DBInstanceAutomatedBackupsReplications":
 [{"DBInstanceAutomatedBackupsArn": "arn:aws:rds:us-east-1:123456789012:auto-backup:ab-
L2IJCEXJP7XQ7HOJ4SIEXAMPLE"}]
        }
    ]
}
```

The following `describe-db-instance-automated-backups` example shows the automated backups for a DB instance.

To show automated backups for a DB instance

- Run one of the following commands.

  For Linux, macOS, or Unix:

  ```
  aws rds describe-db-instance-automated-backups \
  --db-instance-identifier mydatabase
  ```

  For Windows:

  ```
  aws rds describe-db-instance-automated-backups ^
  --db-instance-identifier mydatabase
  ```

The output shows the source DB instance and automated backups in US West (Oregon), with backups replicated to US East (N. Virginia).

```
{
    "DBInstanceAutomatedBackups": [
        {
            "DBInstanceArn": "arn:aws:rds:us-west-2:868710585169:db:mydatabase",
            "DbiResourceId": "db-L2IJCEXJP7XQ7HOJ4SIEXAMPLE",
            "DBInstanceAutomatedBackupsArn": "arn:aws:rds:us-west-2:123456789012:auto-
backup:ab-L2IJCEXJP7XQ7HOJ4SIEXAMPLE",
            "BackupRetentionPeriod": 7,
            "DBInstanceAutomatedBackupsReplications":
 [{"DBInstanceAutomatedBackupsArn": "arn:aws:rds:us-east-1:123456789012:auto-backup:ab-
L2IJCEXJP7XQ7HOJ4SIEXAMPLE"}]
            "Region": "us-west-2",
            "DBInstanceIdentifier": "mydatabase",
```

```
            "RestoreWindow": {
                "EarliestTime": "2020-10-26T01:09:07Z",
                "LatestTime": "2020-10-31T19:09:53Z",
            }
            ...
        }
    ]
}
```

The following `describe-db-instance-automated-backups` example uses the `--db-instance-automated-backups-arn` option to show the replicated backups in the destination Region.

To show replicated backups

- Run one of the following commands.

  For Linux, macOS, or Unix:

  ```
  aws rds describe-db-instance-automated-backups \
  --db-instance-automated-backups-arn "arn:aws:rds:us-east-1:123456789012:auto-
  backup:ab-L2IJCEXJP7XQ7HOJ4SIEXAMPLE"
  ```

  For Windows:

  ```
  aws rds describe-db-instance-automated-backups ^
  --db-instance-automated-backups-arn "arn:aws:rds:us-east-1:123456789012:auto-
  backup:ab-L2IJCEXJP7XQ7HOJ4SIEXAMPLE"
  ```

The output shows the source DB instance in US West (Oregon), with replicated backups in US East (N. Virginia).

```
{
    "DBInstanceAutomatedBackups": [
        {
            "DBInstanceArn": "arn:aws:rds:us-west-2:868710585169:db:mydatabase",
            "DbiResourceId": "db-L2IJCEXJP7XQ7HOJ4SIEXAMPLE",
            "DBInstanceAutomatedBackupsArn": "arn:aws:rds:us-east-1:123456789012:auto-
backup:ab-L2IJCEXJP7XQ7HOJ4SIEXAMPLE",
            "Region": "us-west-2",
```

```
            "DBInstanceIdentifier": "mydatabase",
            "RestoreWindow": {
                "EarliestTime": "2020-10-26T01:09:07Z",
                "LatestTime": "2020-10-31T19:01:23Z"
            },
            "AllocatedStorage": 50,
            "BackupRetentionPeriod": 7,
            "Status": "replicating",
            "Port": 1521,
            ...
        }
    ]
}
```

Restoring to a specified time from a replicated backup for Amazon RDS

You can restore a DB instance to a specific point in time from a replicated backup using the Amazon RDS console. You can also use the `restore-db-instance-to-point-in-time` AWS CLI command or the `RestoreDBInstanceToPointInTime` RDS API operation.

For general information on point-in-time recovery (PITR), see Restoring a DB instance to a specified time for Amazon RDS.

Note

Note the following DB engine restrictions when automated backups are replicated across AWS Regions:

- On RDS for SQL Server, option groups aren't copied.
- On RDS for Oracle, the following options aren't copied: `NATIVE_NETWORK_ENCRYPTION`, `OEM`, `OEM_AGENT`, and `SSL`.

If you've associated a custom option group with your DB instance, you can re-create that option group in the destination Region. Then restore the DB instance in the destination Region and associate the custom option group with it. For more information, see Working with option groups.

Console

To restore a DB instance to a specified time from a replicated backup

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose the destination Region (where backups are replicated to) from the Region selector.
3. In the navigation pane, choose **Automated backups**.
4. On the **Replicated backups** tab, choose the DB instance that you want to restore.
5. For **Actions**, choose **Restore to point in time**.
6. Choose **Latest restorable time** to restore to the latest possible time, or choose **Custom** to choose a time.

   If you chose **Custom**, enter the date and time that you want to restore the instance to.

   > **Note**
   >
   > Times are shown in your local time zone, which is indicated by an offset from Coordinated Universal Time (UTC). For example, UTC-5 is Eastern Standard Time/Central Daylight Time.

7. For **DB instance identifier**, enter the name of the target restored DB instance.
8. (Optional) Choose other options as needed, such as enabling autoscaling.
9. Choose **Restore to point in time**.

AWS CLI

Use the `restore-db-instance-to-point-in-time` AWS CLI command to create a new DB instance.

To restore a DB instance to a specified time from a replicated backup

- Run one of the following commands.

  For Linux, macOS, or Unix:

```
aws rds restore-db-instance-to-point-in-time \
```

```
    --source-db-instance-automated-backups-arn "arn:aws:rds:us-
east-1:123456789012:auto-backup:ab-L2IJCEXJP7XQ7HOJ4SIEXAMPLE" \
    --target-db-instance-identifier mytargetdbinstance \
    --restore-time 2020-10-14T23:45:00.000Z
```

For Windows:

```
aws rds restore-db-instance-to-point-in-time ^
    --source-db-instance-automated-backups-arn "arn:aws:rds:us-
east-1:123456789012:auto-backup:ab-L2IJCEXJP7XQ7HOJ4SIEXAMPLE" ^
    --target-db-instance-identifier mytargetdbinstance ^
    --restore-time 2020-10-14T23:45:00.000Z
```

RDS API

To restore a DB instance to a specified time, call the `RestoreDBInstanceToPointInTime` Amazon RDS API operation with the following parameters:

- `SourceDBInstanceAutomatedBackupsArn`
- `TargetDBInstanceIdentifier`
- `RestoreTime`

Stopping automated backup replication for Amazon RDS

You can stop backup replication for DB instances using the Amazon RDS console. You can also use the `stop-db-instance-automated-backups-replication` AWS CLI command or the `StopDBInstanceAutomatedBackupsReplication` RDS API operation.

Replicated backups are retained, subject to the backup retention period set when they were created.

Console

Stop backup replication from the **Automated backups** page in the source Region.

To stop backup replication to an AWS Region

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. Choose the source Region from the **Region selector**.
3. In the navigation pane, choose **Automated backups**.
4. On the **Current Region** tab, choose the DB instance for which you want to stop backup replication.
5. For **Actions**, choose **Manage cross-Region replication**.
6. Under **Backup replication**, clear the **Enable replication to another AWS Region** check box.
7. Choose **Save**.

Replicated backups are listed on the **Retained** tab of the **Automated backups** page in the destination Region.

AWS CLI

Stop backup replication by using the `stop-db-instance-automated-backups-replication` AWS CLI command.

The following CLI example stops automated backups of a DB instance from replicating in the US West (Oregon) Region.

To stop backup replication

- Run one of the following commands.

  For Linux, macOS, or Unix:

```
aws rds stop-db-instance-automated-backups-replication \
--region us-east-1 \
--source-db-instance-arn "arn:aws:rds:us-west-2:123456789012:db:mydatabase"
```

  For Windows:

```
aws rds stop-db-instance-automated-backups-replication ^
--region us-east-1 ^
--source-db-instance-arn "arn:aws:rds:us-west-2:123456789012:db:mydatabase"
```

RDS API

Stop backup replication by using the `StopDBInstanceAutomatedBackupsReplication` RDS API operation with the following parameters:

- `Region`
- `SourceDBInstanceArn`

Deleting replicated backups for Amazon RDS

You can delete replicated backups for DB instances using the Amazon RDS console. You can also use the `delete-db-instance-automated-backups` AWS CLI command or the `DeleteDBInstanceAutomatedBackup` RDS API operation.

Console

Delete replicated backups in the destination Region from the **Automated backups** page.

To delete replicated backups

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose the destination Region from the **Region selector**.
3. In the navigation pane, choose **Automated backups**.
4. On the **Replicated backups** tab, choose the DB instance for which you want to delete the replicated backups.
5. For **Actions**, choose **Delete**.
6. On the confirmation page, enter **`delete me`** and choose **Delete**.

AWS CLI

Delete replicated backups by using the `delete-db-instance-automated-backup` AWS CLI command.

You can use the `describe-db-instances` CLI command to find the Amazon Resource Names (ARNs) of the replicated backups. For more information, see Finding information about replicated backups for Amazon RDS.

To delete replicated backups

- Run one of the following commands.

  For Linux, macOS, or Unix:

```
aws rds delete-db-instance-automated-backup \
--db-instance-automated-backups-arn "arn:aws:rds:us-east-1:123456789012:auto-
backup:ab-L2IJCEXJP7XQ7HOJ4SIEXAMPLE"
```

  For Windows:

```
aws rds delete-db-instance-automated-backup ^
--db-instance-automated-backups-arn "arn:aws:rds:us-east-1:123456789012:auto-
backup:ab-L2IJCEXJP7XQ7HOJ4SIEXAMPLE"
```

RDS API

Delete replicated backups by using the `DeleteDBInstanceAutomatedBackup` RDS API operation with the `DBInstanceAutomatedBackupsArn` parameter.

Troubleshooting stopped cross-Region automated backups

Amazon RDS automatically stops cross-Region automated backup replication under specific circumstances to protect your data and maintain compliance with AWS operational requirements.

Account suspension

If your AWS account is suspended, Amazon RDS automatically stops cross-Region automated backup replication for all DB instances in that account. The replicated backups that already exist in the destination AWS Region are kept up to your specified retention period.

After your account suspension is resolved, you must manually re-enable cross-Region automated backup replication to resume replicating backups to the destination AWS Region.

Opt-in Region changes

Cross-Region automated backup replication stops automatically when you opt out of either the source AWS Region (where the primary DB instance is located) or the destination AWS Region (where backups are being replicated). For more information about opt-in Regions, see Managing AWS Regions.

The replicated backups that already exist in the destination AWS Region are kept up to your specified retention period. To resume replication, opt back into the required AWS Region and manually re-enable cross-Region automated backup replication.

AWS KMS key issues

For encrypted DB instances, Amazon RDS requires access to AWS KMS keys in both the source and destination AWS Region to replicate backups. If you disable or delete the AWS KMS key in either AWS Region, cross-Region automated backup replication stops automatically. The replicated backups that already exist in the destination AWS Region are kept up to your specified retention period.

To resume cross-Region automated backup replication:

1. Re-enable the disabled AWS KMS key, or create a new AWS KMS key if the original was deleted
2. If using a new AWS KMS key, delete existing replicated backup in the destination AWS Region that was encrypted with the previous key.
3. Re-enable cross-Region automated backup replication for the instance

For more information about managing AWS KMS keys, see AWS Key Management Service documentation.

> **Note**
>
> You cannot restore from previously replicated backups in the destination AWS Region unless the previous AWS KMS key is re-enabled. The backups remain encrypted and inaccessible without a valid AWS KMS key.

Monitoring backup replication status

You can monitor the status of your cross-Region automated backups using the Amazon RDS console, AWS CLI, or RDS API. For more information, see Finding information about replicated backups for Amazon RDS.

Managing manual backups

This section shows how to manage manual backups for DB instances and DB clusters.

For information about creating a DB snapshot for a Single-AZ DB instance, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

For information about creating a DB snapshot for a Multi-AZ DB cluster, see Creating a Multi-AZ DB cluster snapshot for Amazon RDS.

For information about deleting a DB snapshot, see Deleting a DB snapshot for Amazon RDS.

Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS

Amazon RDS creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. Creating this DB snapshot on a Single-AZ DB instance results in a brief I/O suspension that can last from a few seconds to a few minutes, depending on the size and class of your DB instance. For MariaDB, MySQL, Oracle, and PostgreSQL, I/O activity is not suspended on your primary during backup for Multi-AZ deployments, because the backup is taken from the standby. For SQL Server, I/O activity is suspended briefly during backup for Multi-AZ deployments.

When you create a DB snapshot, you need to identify which DB instance you are going to back up, and then give your DB snapshot a name so you can restore from it later. The amount of time it takes to create a snapshot varies with the size of your databases. Since the snapshot includes the entire storage volume, the size of files, such as temporary files, also affects the amount of time it takes to create the snapshot.

> **Note**
>
> Your DB instance must be in the `available` state to take a DB snapshot.
> For PostgreSQL DB instances, data in unlogged tables might not be restored from snapshots. For more information, see Best practices for working with PostgreSQL.

Unlike automated backups, manual snapshots aren't subject to the backup retention period. Snapshots don't expire.

For very long-term backups of MariaDB, MySQL, and PostgreSQL data, we recommend exporting snapshot data to Amazon S3. If the major version of your DB engine is no longer supported, you can't restore to that version from a snapshot. For more information, see Exporting DB snapshot data to Amazon S3 for Amazon RDS.

You can create a DB snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To create a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Snapshots**.

   The **Manual snapshots** list appears.
3. Choose **Take snapshot**.

   The **Take DB snapshot** window appears.
   Take DB snapshot.
4. Choose the **DB instance** for which you want to take a snapshot.
5. Enter the **Snapshot name**.
6. Choose **Take snapshot**.

The **Manual snapshots** list appears, with the new DB snapshot's status shown as `Creating`. After its status is `Available`, you can see its creation time.

AWS CLI

When you create a DB snapshot using the AWS CLI, you need to identify which DB instance you are going to back up, and then give your DB snapshot a name so you can restore from it later. You can do this by using the AWS CLI `create-db-snapshot` command with the following parameters:

- `--db-instance-identifier`
- `--db-snapshot-identifier`

In this example, you create a DB snapshot called *`mydbsnapshot`* for a DB instance called *`mydbinstance`*.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-snapshot \
    --db-instance-identifier mydbinstance \
    --db-snapshot-identifier mydbsnapshot
```

For Windows:

```
aws rds create-db-snapshot ^
    --db-instance-identifier mydbinstance ^
```

```
    --db-snapshot-identifier mydbsnapshot
```

RDS API

When you create a DB snapshot using the Amazon RDS API, you need to identify which DB instance you are going to back up, and then give your DB snapshot a name so you can restore from it later. You can do this by using the Amazon RDS API `CreateDBSnapshot` command with the following parameters:

- `DBInstanceIdentifier`
- `DBSnapshotIdentifier`

Creating a Multi-AZ DB cluster snapshot for Amazon RDS

When you create a Multi-AZ DB cluster snapshot, make sure to identify which Multi-AZ DB cluster you are going to back up, and then give your DB cluster snapshot a name so you can restore from it later. You can also share a Multi-AZ DB cluster snapshot. For instructions, see the section called "Sharing a DB snapshot".

You can create a Multi-AZ DB cluster snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

For very long-term backups, we recommend exporting snapshot data to Amazon S3. If the major version of your DB engine is no longer supported, you can't restore to that version from a snapshot. For more information, see Exporting DB snapshot data to Amazon S3 for Amazon RDS.

Console

To create a DB cluster snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. In the list, choose the Multi-AZ DB cluster for which you want to take a snapshot.
4. For **Actions**, choose **Take snapshot**.

   The **Take DB snapshot** window appears.
5. For **Snapshot name**, enter the name of the snapshot.
6. Choose **Take snapshot**.

The **Snapshots** page appears, with the new Multi-AZ DB cluster snapshot's status shown as `Creating`. After its status is `Available`, you can see its creation time.

AWS CLI

You can create a Multi-AZ DB cluster snapshot by using the AWS CLI create-db-cluster-snapshot command with the following options:

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

In this example, you create a Multi-AZ DB cluster snapshot called *mymultiazdbclustersnapshot* for a DB cluster called *mymultiazdbcluster*.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier mymultiazdbcluster \
    --db-cluster-snapshot-identifier mymultiazdbclustersnapshot
```

For Windows:

```
aws rds create-db-cluster-snapshot ^
    --db-cluster-identifier mymultiazdbcluster ^
    --db-cluster snapshot-identifier mymultiazdbclustersnapshot
```

RDS API

You can create a Multi-AZ DB cluster snapshot by using the Amazon RDS API CreateDBClusterSnapshot operation with the following parameters:

- `DBClusterIdentifier`
- `DBClusterSnapshotIdentifier`

Deleting a Multi-AZ DB cluster snapshot

You can delete Multi-AZ DB snapshots managed by Amazon RDS when you no longer need them. For instructions, see the section called “Deleting a DB snapshot”.

Deleting a DB snapshot for Amazon RDS

You can delete DB snapshots managed by Amazon RDS when you no longer need them.

> **Note**
>
> To delete backups managed by AWS Backup, use the AWS Backup console. For information about AWS Backup, see the *AWS Backup Developer Guide*.

Deleting a DB snapshot

You can delete a manual, shared, or public DB snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

To delete a shared or public snapshot, you must sign in to the AWS account that owns the snapshot.

If you have automated DB snapshots that you want to delete without deleting the DB instance, change the backup retention period for the DB instance to 0. The automated snapshots are deleted when the change is applied. You can apply the change immediately if you don't want to wait until the next maintenance period. After the change is complete, you can then re-enable automatic backups by setting the backup retention period to a number greater than 0. For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

Retained automated backups and manual snapshots incur billing charges until they're deleted. For more information, see Retention costs.

If you deleted a DB instance, you can delete its automated DB snapshots by removing the automated backups for the DB instance. For information about automated backups, see Introduction to backups.

Console

To delete a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.

The **Manual snapshots** list appears.

3. Choose the DB snapshot that you want to delete.
4. For **Actions**, choose **Delete snapshot**.
   Delete snapshot
5. Choose **Delete** on the confirmation page.
   Confirm snapshot deletion

AWS CLI

You can delete a DB snapshot by using the AWS CLI command delete-db-snapshot.

The following options are used to delete a DB snapshot.

- `--db-snapshot-identifier` – The identifier for the DB snapshot.

Example

The following code deletes the `mydbsnapshot` DB snapshot.

For Linux, macOS, or Unix:

```
aws rds delete-db-snapshot \
    --db-snapshot-identifier mydbsnapshot
```

For Windows:

```
aws rds delete-db-snapshot ^
    --db-snapshot-identifier mydbsnapshot
```

RDS API

You can delete a DB snapshot by using the Amazon RDS API operation DeleteDBSnapshot.

The following parameters are used to delete a DB snapshot.

- `DBSnapshotIdentifier` – The identifier for the DB snapshot.

Restoring to a DB instance

This section shows how to restore to a DB instance. This page shows how to restore to an Amazon RDS DB instance from a DB snapshot.

Amazon RDS creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. You can create a new DB instance by restoring from a DB snapshot. You provide the name of the DB snapshot to restore from, and then provide a name for the new DB instance that is created from the restore. You can't restore from a DB snapshot to an existing DB instance; you create a new DB instance when you restore the snapshot.

You can use the restored DB instance as soon as its status is `available`. The DB instance continues to load data in the background. This is known as *lazy loading*. If you access data that hasn't been loaded yet, the DB instance immediately downloads the requested data from Amazon S3, and then continues loading the rest of the data in the background. For more information, see Amazon EBS snapshots.

To help mitigate the effects of lazy loading on tables to which you require quick access, you can perform operations that involve full-table scans, such as `SELECT *`. This allows Amazon RDS to download all of the backed-up table data from S3.

You can restore a DB instance and use a different storage type than the source DB snapshot. In this case, the restore process is slower because of the additional work required to migrate the data to the new storage type. If you restore to or from magnetic storage, the migration process is the slowest. That's because magnetic storage doesn't have the IOPS capability of Provisioned IOPS or General Purpose (SSD) storage.

You can use CloudFormation to restore a DB instance from a DB instance snapshot. For more information, see AWS::RDS::DBInstance in the *AWS CloudFormation User Guide*.

> **Note**
>
> You can't restore a DB instance from a DB snapshot that is both shared and encrypted. Instead, you can make a copy of the DB snapshot and restore the DB instance from the copy. For more information, see Copying a DB snapshot for Amazon RDS.

For information about restoring a DB instance with an RDS Extended Support version, see Restoring a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support.

Restoring from a snapshot

You can restore a DB instance from a DB snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

> **Note**
>
> You can't reduce the amount of storage when you restore a DB instance. When you increase the allocated storage, it must be by at least 10 percent. If you try to increase the value by less than 10 percent, you get an error. You can't increase the allocated storage when restoring RDS for SQL Server DB instances.

Console

To restore a DB instance from a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.
5. On the **Restore snapshot** page, for **DB instance identifier**, enter the name for your restored DB instance.
6. Specify other settings, such as allocated storage size.

   For information about each setting, see Settings for DB instances.
7. Choose **Restore DB instance**.

AWS CLI

To restore a DB instance from a DB snapshot, use the AWS CLI command restore-db-instance-from-db-snapshot.

In this example, you restore from a previously created DB snapshot named `mydbsnapshot`. You restore to a new DB instance named `mynewdbinstance`. This example also sets the allocated storage size.

You can specify other settings. For information about each setting, see Settings for DB instances.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier mynewdbinstance \
    --db-snapshot-identifier mydbsnapshot \
    --allocated-storage 100
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^
    --db-instance-identifier mynewdbinstance ^
    --db-snapshot-identifier mydbsnapshot ^
    --allocated-storage 100
```

This command returns output similar to the following:

```
DBINSTANCE  mynewdbinstance  db.t3.small  MySQL     50       sa             creating
 3  n  8.0.28  general-public-license
```

Example

The following example shows restoring a snapshot while adding an additional storage volume to the newly created instance. The snapshot included additional volume `rdsdbdata2`. The restore operation adds `rdsdbdata3`, making a total of three volumes in the newly created instance. You can't delete a volume when you restore the snapshot.

```
aws rds restore-db-instance-from-db-snapshot \
     --db-instance-identifier my-restored-instance \
     --db-snapshot-identifier my-asv-snapshot \
     --additional-storage-volumes '[{ \
            "VolumeName": "rdsdbdata3", \
            "StorageType":"gp3", \
            "AllocatedStorage": 5000, \
            "IOPS": 12000 \
        }]'
```

RDS API

To restore a DB instance from a DB snapshot, call the Amazon RDS API function RestoreDBInstanceFromDBSnapshot with the following parameters:

- `DBInstanceIdentifier`
- `DBSnapshotIdentifier`

Considerations

For considerations when restoring to a DB instance from a DB snapshot, see the following topics.

Topics

- Parameter group considerations
- Security group considerations
- Option group considerations
- Resource tagging considerations
- Db2 considerations
- Microsoft SQL Server considerations
- MySQL considerations
- Oracle Database considerations

Parameter group considerations

We recommend that you retain the DB parameter group for any DB snapshots you create, so that you can associate your restored DB instance with the correct parameter group.

The default DB parameter group is associated with the restored instance, unless you choose a different one. No custom parameter settings are available in the default parameter group.

You can specify the parameter group when you restore the DB instance.

For more information about DB parameter groups, see Parameter groups for Amazon RDS.

Security group considerations

When you restore a DB instance, the default virtual private cloud (VPC), DB subnet group, and VPC security group are associated with the restored instance, unless you choose different ones.

- If you're using the Amazon RDS console, you can specify a custom VPC security group to associate with the instance or create a new VPC security group.
- If you're using the AWS CLI, you can specify a custom VPC security group to associate with the instance by including the `--vpc-security-group-ids` option in the `restore-db-instance-from-db-snapshot` command.
- If you're using the Amazon RDS API, you can include the `VpcSecurityGroupIds.VpcSecurityGroupId.N` parameter in the `RestoreDBInstanceFromDBSnapshot` action.

As soon as the restore is complete and your new DB instance is available, you can also change the VPC settings by modifying the DB instance. For more information, see Modifying an Amazon RDS DB instance.

Option group considerations

When you restore a DB instance, the default DB option group is associated with the restored DB instance in most cases.

The exception is when the source DB instance is associated with an option group that contains a persistent or permanent option. For example, if the source DB instance uses Oracle Transparent Data Encryption (TDE), the restored DB instance must use an option group that has the TDE option.

If you restore a DB instance into a different VPC, you must do one of the following to assign a DB option group:

- Assign the default option group for that VPC group to the instance.
- Assign another option group that is linked to that VPC.
- Create a new option group and assign it to the DB instance. With persistent or permanent options, such as Oracle TDE, you must create a new option group that includes the persistent or permanent option.

For more information about DB option groups, see Working with option groups.

Resource tagging considerations

When you restore a DB instance from a DB snapshot, RDS checks whether you specify new tags. If yes, the new tags are added to the restored DB instance. If there are no new tags, RDS adds the tags from the source DB instance at the time of snapshot creation to the restored DB instance.

For more information, see Copying tags to DB snapshots.

Db2 considerations

With the BYOL model, your Amazon RDS for Db2 DB instances must be associated with a custom parameter group that contains your IBM Site ID and your IBM Customer ID. Otherwise, attempts to restore a DB instance from a snapshot will fail. Your Amazon RDS for Db2 DB instances must also be associated with an AWS License Manager self-managed license. For more information, see Bring your own license (BYOL) for Db2.

With the Db2 license through AWS Marketplace model, you need an active AWS Marketplace subscription for the particular IBM Db2 edition that you want to use. If you don't already have one, subscribe to Db2 in AWS Marketplace for that IBM Db2 edition. For more information, see Db2 license through AWS Marketplace.

Microsoft SQL Server considerations

When you restore an RDS for Microsoft SQL Server DB snapshot to a new instance, you can always restore to the same edition as your snapshot. In some cases, you can also change the edition of the DB instance. The following limitations apply when you change editions:

- The DB snapshot must have enough storage allocated for the new edition.
- Only the following edition changes are supported:
  - From Standard Edition to Enterprise Edition
  - From Web Edition to Standard Edition or Enterprise Edition
  - From Express Edition to Web Edition, Standard Edition, or Enterprise Edition

If you want to change from one edition to a new edition that isn't supported by restoring a snapshot, you can try using the native backup and restore feature. SQL Server verifies whether your database is compatible with the new edition based on what SQL Server features you have enabled on the database. For more information, see Importing and exporting SQL Server databases using native backup and restore.

MySQL considerations

To restore from a RDS for MySQL DB snapshot with an unsupported engine version, you might have to upgrade your DB snapshot more than once. For more information about upgrade options, see Upgrade options for DB snapshots with unsupported engine versions for RDS for MySQL.

For more information about upgrading the engine version of a RDS for MySQL DB snapshot, Upgrading a MySQL DB snapshot engine version.

Oracle Database considerations

When you restore an Oracle database from a DB snapshot, consider the following:

- Before you restore a DB snapshot, you can upgrade it to a later Oracle database release. For more information, see Upgrading an Oracle DB snapshot.
- If you restore a snapshot of a CDB instance that uses the single-tenant configuration, you can change the PDB name. You can't change the PDB names when your CDB instance uses the multi-tenant configuration. For more information, see Backing up and restoring a CDB.
- You can't change the CDB name, which is always RDSCDB. This CDB name is the same for all CDB instances.
- You can't directly interact with the tenant databases in a DB snapshot. If you restore a snapshot of a CDB instance that uses the multi-tenant configuration, you restore all its tenant databases. You can use describe-db-snapshot-tenant-databases to inspect the tenant databases within a DB snapshot before restoring it.
- If you use Oracle GoldenGate, always retain the parameter group with the `compatible` parameter. When you restore a DB instance from a DB snapshot, specify a parameter group that has a matching or greater `compatible` value.
- You might choose to rename your database when you restore a DB snapshot. If the total size of online redo log is greater than 20GB, RDS might reset your online redo log size to its default settings of 512MB (4 x 128MB). The smaller size allows the restore operation to complete in a reasonable time. You can re-create the online redo logs later and change the size.
- You can manage your master user password in AWS Secrets Manager. For more information, see Overview of managing master user passwords with AWS Secrets Manager.

Restoring a DB instance to a specified time for Amazon RDS

You can restore a DB instance to a specific point in time, creating a new DB instance without modifying the source DB instance.

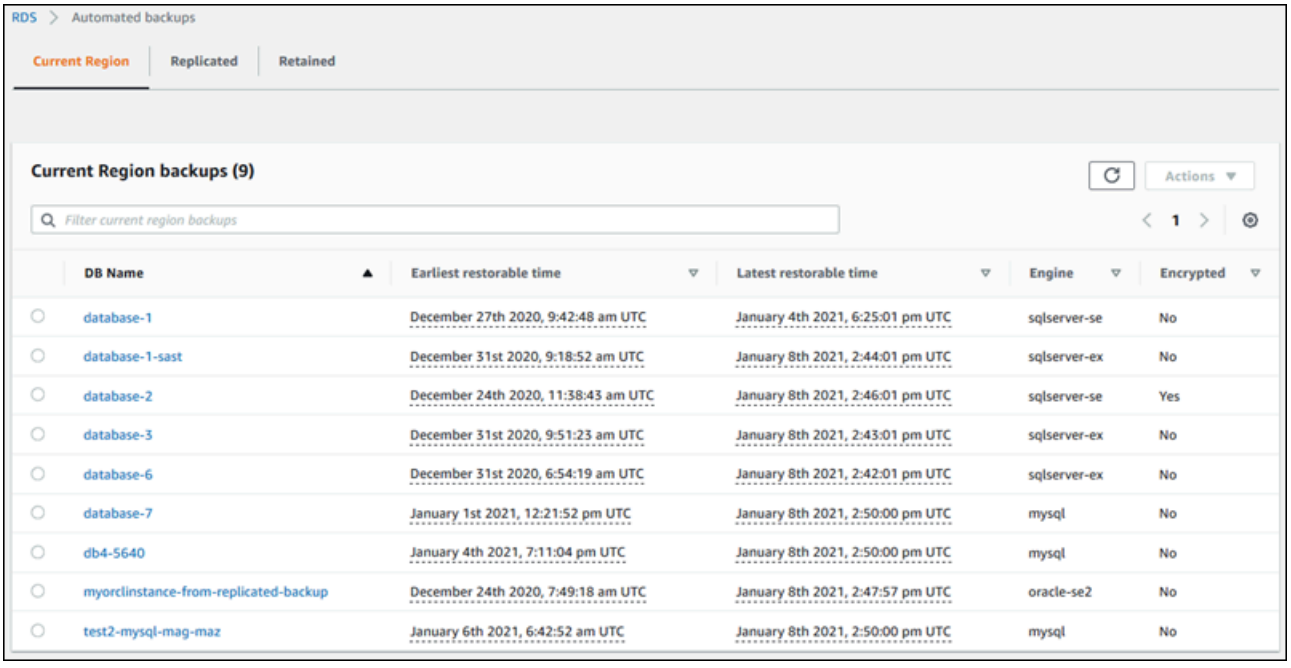
When you restore a DB instance to a point in time, you can choose the default virtual private cloud (VPC) security group. Or you can apply a custom VPC security group to your DB instance.

Restored DB instances are automatically associated with the default DB parameter and option groups. However, you can apply a custom parameter group and option group by specifying them during a restore.

If tags are provided in the request then the provided tags are applied to the restored DB instance. If tags are not provided in the request and if the source DB instance is in-region active and has tags, RDS adds the latest tags from the source DB instance to the restored DB instance.

RDS uploads transaction logs for DB instances to Amazon S3 every five minutes. To see the latest restorable time for a DB instance, use the AWS CLI describe-db-instances command and look at the value returned in the `LatestRestorableTime` field for the DB instance. To see the latest restorable time for each DB instance in the Amazon RDS console, choose **Automated backups**.

You can restore to any point in time within your backup retention period. To see the earliest restorable time for each DB instance, choose **Automated backups** in the Amazon RDS console.



> **Note**
>
> We recommend that you restore to the same or similar DB instance size—and IOPS if using Provisioned IOPS storage—as the source DB instance. You might get an error if, for example, you choose a DB instance size with an incompatible IOPS value.

For information about restoring a DB instance with an RDS Extended Support version, see Restoring a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support.

Some Amazon RDS database engines have special considerations when restoring from a point in time:

- If you use password authentication with an Amazon RDS for Db2 DB instance, user management actions, including `rdsadmin.add_user`, won't be captured in the logs. These actions require a full snapshot backup.

  With the BYOL model, your RDS for Db2 DB instances must be associated with a custom parameter group that contains your IBM Site ID and your IBM Customer ID. Otherwise, attempts to restore a DB instance to a specific point in time will fail. Your Amazon RDS for Db2 DB instances must also be associated with an AWS License Manager self-managed license. For more information, see Bring your own license (BYOL) for Db2.

  With the Db2 license through AWS Marketplace model, you need an active AWS Marketplace subscription for the particular IBM Db2 edition that you want to use. If you don't already have one, subscribe to Db2 in AWS Marketplace for that IBM Db2 edition. For more information, see Db2 license through AWS Marketplace.
- When you restore an RDS for Oracle DB instance to a point in time, you can specify a different DB engine, license model, and DBName (SID) for the restored DB instance. You can also specify that RDS should store manage master user passwords in AWS Secrets Manager. For more information, see Overview of managing master user passwords with AWS Secrets Manager.
- When you restore a Microsoft SQL Server DB instance to a point in time, each database within that instance is restored to a point in time within 1 second of each other database within the instance. Transactions that span multiple databases within the instance might be restored inconsistently.

- For a SQL Server DB instance, the `OFFLINE`, `EMERGENCY`, and `SINGLE_USER` modes aren't supported. Setting any database into one of these modes causes the latest restorable time to stop moving ahead for the whole instance.
- Some actions, such as changing the recovery model of a SQL Server database, can break the sequence of logs that are used for point-in-time recovery. In some cases, Amazon RDS can detect this issue and the latest restorable time is prevented from moving forward. In other cases, such as when a SQL Server database uses the `BULK_LOGGED` recovery model, the break in log sequence isn't detected. It might not be possible to restore a SQL Server DB instance to a point in time if there is a break in the log sequence. For these reasons, Amazon RDS doesn't support changing the recovery model of SQL Server databases.

You can also use AWS Backup to manage backups of Amazon RDS DB instances. If your DB instance is associated with a backup plan in AWS Backup, that backup plan is used for point-in-time recovery. Backups that were created with AWS Backup have names ending in `awsbackup:`*`AWS-Backup-job-number`*. For information about AWS Backup, see the *AWS Backup Developer Guide*.

> **Note**
>
> Information in this topic applies to Amazon RDS. For information on restoring an Amazon Aurora DB cluster, see Restoring a DB cluster to a specified time.

You can restore a DB instance to a point in time using the AWS Management Console, the AWS CLI, or the RDS API.

> **Note**
>
> You can't reduce the amount of storage when you restore a DB instance. When you increase the allocated storage, it must be by at least 10 percent. If you try to increase the value by less than 10 percent, you get an error. You can't increase the allocated storage when restoring RDS for SQL Server DB instances.

Console

To restore a DB instance to a specified time

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Automated backups**.

   The automated backups are displayed on the **Current Region** tab.
3. Choose the DB instance that you want to restore.
4. For **Actions**, choose **Restore to point in time**.

   The **Restore to point in time** window appears.
5. Choose **Latest restorable time** to restore to the latest possible time, or choose **Custom** to choose a time.

   If you chose **Custom**, enter the date and time to which you want to restore the instance.

   > **Note**
   >
   > Times are shown in your local time zone, which is indicated by an offset from Coordinated Universal Time (UTC). For example, UTC-5 is Eastern Standard Time/Central Daylight Time.
6. For **DB instance identifier**, enter the name of the target restored DB instance. The name must be unique.
7. Choose other options as needed, such as DB instance class, storage, and whether you want to use storage autoscaling.

   For information about each setting, see Settings for DB instances.
8. Choose **Restore to point in time**.

AWS CLI

To restore a DB instance to a specified time, use the AWS CLI command restore-db-instance-to-point-in-time to create a new DB instance. This example also sets the allocated storage size and enables storage autoscaling.

Resource tagging is supported for this operation. When you use the `--tags` option, the source DB instance tags are ignored and the provided ones are used. Otherwise, the latest tags from the source instance are used.

You can specify other settings. For information about each setting, see Settings for DB instances.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-to-point-in-time \
    --source-db-instance-identifier mysourcedbinstance \
    --target-db-instance-identifier mytargetdbinstance \
    --restore-time 2017-10-14T23:45:00.000Z \
    --allocated-storage 100 \
    --max-allocated-storage 1000
```

For Windows:

```
aws rds restore-db-instance-to-point-in-time ^
    --source-db-instance-identifier mysourcedbinstance ^
    --target-db-instance-identifier mytargetdbinstance ^
    --restore-time 2017-10-14T23:45:00.000Z ^
    --allocated-storage 100 ^
    --max-allocated-storage 1000
```

Example

The following example shows adding a volume when restoring the instance to a point in time.

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-to-point-in-time \
     --source-db-instance-identifier my-asv-instance \
     --target-db-instance-identifier my-pitr-instance \
     --use-latest-restorable-time \
     --additional-storage-volumes '[{ \
             "VolumeName": "rdsdbdata2", \
             "StorageType":"gp3", \
             "AllocatedStorage": 5000, \
             "IOPS": 12000 \
         }]'
```

For Windows:

```
aws rds restore-db-instance-to-point-in-time ^
     --source-db-instance-identifier my-asv-instance ^
     --target-db-instance-identifier my-pitr-instance ^
     --use-latest-restorable-time ^
     --additional-storage-volumes '[{ ^
             "VolumeName": "rdsdbdata2", ^
             "StorageType":"gp3", ^
             "AllocatedStorage": 5000, ^
             "IOPS": 12000 ^
         }]'
```

RDS API

To restore a DB instance to a specified time, call the Amazon RDS API `RestoreDBInstanceToPointInTime` operation with the following parameters:

- `SourceDBInstanceIdentifier`
- `TargetDBInstanceIdentifier`
- `RestoreTime`

Restoring a Multi-AZ DB cluster to a specified time

You can restore a Multi-AZ DB cluster to a specific point in time, creating a new Multi-AZ DB cluster.

RDS uploads transaction logs for Multi-AZ DB clusters to Amazon S3 continuously. You can restore to any point in time within your backup retention period. To see the earliest restorable time for a Multi-AZ DB cluster, use the AWS CLI describe-db-clusters command. Look at the value returned in the `EarliestRestorableTime` field for the DB cluster. To see the latest restorable time for a Multi-AZ DB cluster, look at the value returned in the `LatestRestorableTime` field for the DB cluster.

When you restore a Multi-AZ DB cluster to a point in time, you can choose the default VPC security group for your Multi-AZ DB cluster, or you can apply a custom VPC security group to your Multi-AZ DB cluster.

Restored Multi-AZ DB clusters are automatically associated with the default DB cluster parameter group. However, you can apply a custom DB cluster parameter group by specifying it during a restore.

If the source DB cluster has resource tags, RDS adds the latest tags to the restored DB cluster.

> **Note**
>
> We recommend that you restore to the same or similar Multi-AZ DB cluster size as the source DB cluster. We also recommend that you restore with the same or similar IOPS value if you're using Provisioned IOPS storage. You might get an error if, for example, you choose a DB cluster size with an incompatible IOPS value.
> If the source Multi-AZ DB cluster uses General Purpose SSD (gp3) storage and has less than 400 GiB of allocated storage, you can't modify the provisioned IOPS for the restored DB cluster.

For information about restoring a Multi-AZ DB cluster with an RDS Extended Support version, see Restoring a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support.

You can restore a Multi-AZ DB cluster to a point in time using the AWS Management Console, the AWS CLI, or the RDS API.

Console

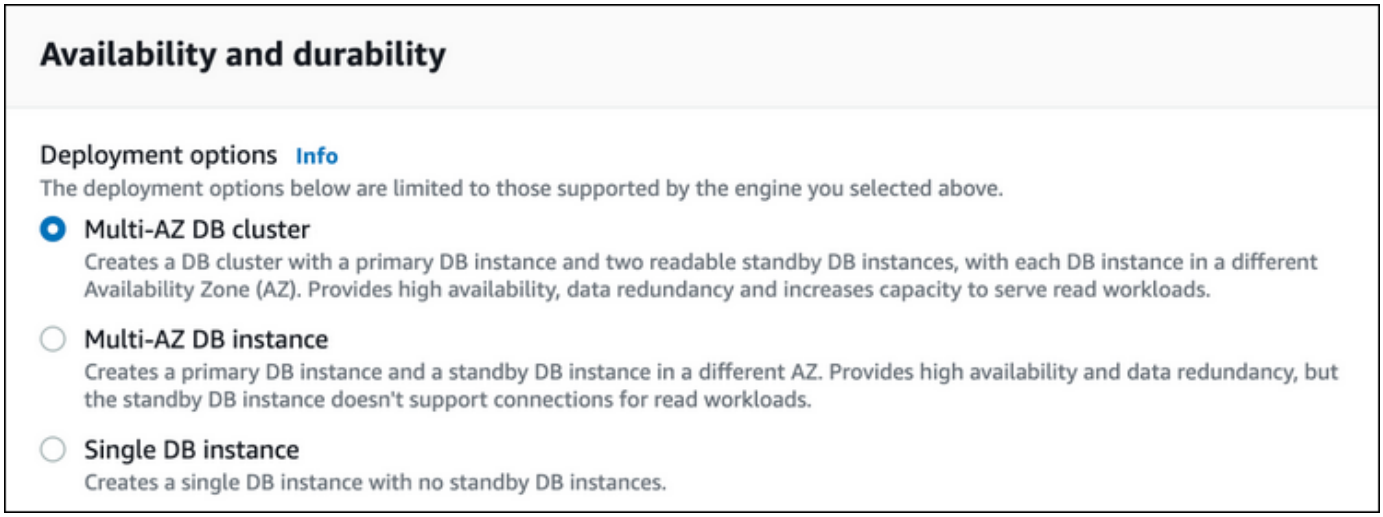
To restore a Multi-AZ DB cluster to a specified time

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the Multi-AZ DB cluster that you want to restore.
4. For **Actions**, choose **Restore to point in time**.

   The **Restore to point in time** window appears.
5. Choose **Latest restorable time** to restore to the latest possible time, or choose **Custom** to choose a time.

   If you chose **Custom**, enter the date and time to which you want to restore the Multi-AZ DB cluster.

   > **Note**
   >
   > Times are shown in your local time zone, which is indicated by an offset from Coordinated Universal Time (UTC). For example, UTC-5 is Eastern Standard Time/ Central Daylight Time.
6. For **DB cluster identifier**, enter the name for your restored Multi-AZ DB cluster.
7. In **Availability and durability**, choose **Multi-AZ DB cluster**.

8. In **DB instance class**, choose a DB instance class.

Currently, Multi-AZ DB clusters only support db.m6gd and db.r6gd DB instance classes. For more information about DB instance classes, see DB instance classes.

9. For the remaining sections, specify your DB cluster settings. For information about each setting, see Settings for creating Multi-AZ DB clusters.
10. Choose **Restore to point in time**.

AWS CLI

To restore a Multi-AZ DB cluster to a specified time, use the AWS CLI command restore-db-cluster-to-point-in-time to create a new Multi-AZ DB cluster.

Currently, Multi-AZ DB clusters only support db.m6gd and db.r6gd DB instance classes. For more information about DB instance classes, see DB instance classes.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier mysourcemultiazdbcluster \
    --db-cluster-identifier mytargetmultiazdbcluster \
    --restore-to-time 2021-08-14T23:45:00.000Z \
    --db-cluster-instance-class db.r6gd.xlarge
```

For Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier mysourcemultiazdbcluster ^
    --db-cluster-identifier mytargetmultiazdbcluster ^
    --restore-to-time 2021-08-14T23:45:00.000Z ^
    --db-cluster-instance-class db.r6gd.xlarge
```

RDS API

To restore a DB cluster to a specified time, call the Amazon RDS API RestoreDBClusterToPointInTime operation with the following parameters:

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`

- `RestoreToTime`

Restoring from a snapshot to a Multi-AZ DB cluster

You can restore a snapshot to a Multi-AZ DB cluster using the AWS Management Console, the AWS CLI, or the RDS API. You can restore each of these types of snapshots to a Multi-AZ DB cluster:

- A snapshot of a Single-AZ deployment
- A snapshot of a Multi-AZ DB cluster deployment with a single DB instance
- A snapshot of a Multi-AZ DB cluster

For information about Multi-AZ deployments, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

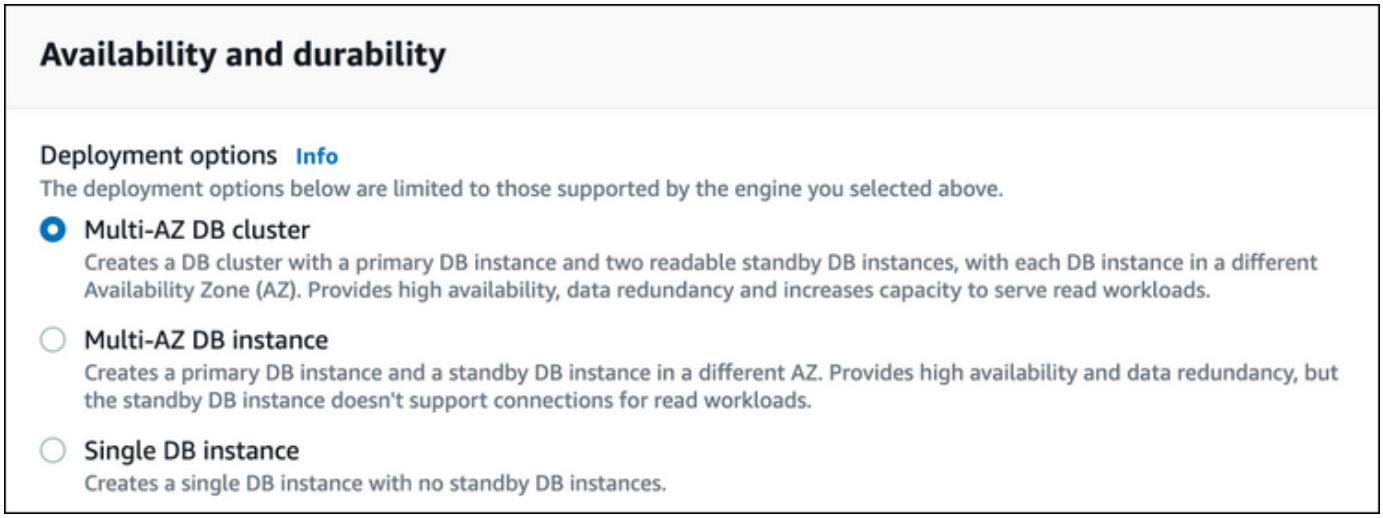
> **Tip**
>
> You can migrate a Single-AZ deployment or a Multi-AZ DB cluster deployment to a Multi-AZ DB cluster deployment by restoring a snapshot.

For information about restoring Multi-AZ DB cluster with an RDS Extended Support version, see Restoring a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support.

Console

To restore a snapshot to a Multi-AZ DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.
5. On the **Restore snapshot** page, in **Availability and durability**, choose **Multi-AZ DB cluster**.



6. For **DB cluster identifier**, enter the name for your restored Multi-AZ DB cluster.
7. For the remaining sections, specify your DB cluster settings. For information about each setting, see Settings for creating Multi-AZ DB clusters.
8. Choose **Restore DB instance**.

AWS CLI

To restore a snapshot to a Multi-AZ DB cluster, use the AWS CLI command restore-db-cluster-from-snapshot.

In the following example, you restore from a previously created snapshot named `mysnapshot`. You restore to a new Multi-AZ DB cluster named `mynewmultiazdbcluster`. You also specify the DB instance class used by the DB instances in the Multi-AZ DB cluster. Specify either `mysql` or `postgres` for the DB engine.

For the `--snapshot-identifier` option, you can use either the name or the Amazon Resource Name (ARN) to specify a DB cluster snapshot. However, you can use only the ARN to specify a DB snapshot.

For the `--db-cluster-instance-class` option, specify the DB instance class for the new Multi-AZ DB cluster. Multi-AZ DB clusters only support specific DB instance classes, such as the db.m6gd and db.r6gd DB instance classes. For more information about DB instance classes, see DB instance classes.

You can also specify other options.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewmultiazdbcluster \
    --snapshot-identifier mysnapshot \
    --engine mysql|postgres \
    --db-cluster-instance-class db.r6gd.xlarge
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mynewmultiazdbcluster ^
    --snapshot-identifier mysnapshot ^
    --engine mysql|postgres ^
    --db-cluster-instance-class db.r6gd.xlarge
```

After you restore the DB cluster, you can add the Multi-AZ DB cluster to the security group associated with the DB cluster or DB instance that you used to create the snapshot, if applicable. Completing this action provides the same functions of the previous DB cluster or DB instance.

RDS API

To restore a snapshot to a Multi-AZ DB cluster, call the RDS API operation RestoreDBClusterFromSnapshot with the following parameters:

- `DBClusterIdentifier`
- `SnapshotIdentifier`
- `Engine`

You can also specify other optional parameters.

After you restore the DB cluster, you can add the Multi-AZ DB cluster to the security group associated with the DB cluster or DB instance that you used to create the snapshot, if applicable. Completing this action provides the same functions of the previous DB cluster or DB instance.

Restoring from a Multi-AZ DB cluster snapshot to a DB instance

A *Multi-AZ DB cluster snapshot* is a storage volume snapshot of your DB cluster, backing up the entire DB cluster and not just individual databases. You can restore a Multi-AZ DB cluster snapshot to a Single-AZ deployment or Multi-AZ DB instance deployment. For information about Multi-AZ deployments, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

> **Note**
>
> You can also restore a Multi-AZ DB cluster snapshot to a new Multi-AZ DB cluster. For instructions, see Restoring from a snapshot to a Multi-AZ DB cluster.

For information about restoring a Multi-AZ DB cluster with an RDS Extended Support version, see Restoring a DB instance or a Multi-AZ DB cluster with Amazon RDS Extended Support.

Use the AWS Management Console, the AWS CLI, or the RDS API to restore a Multi-AZ DB cluster snapshot to a Single-AZ deployment or Multi-AZ DB instance deployment.

Console

To restore a Multi-AZ DB cluster snapshot to a Single-AZ deployment or Multi-AZ DB instance deployment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the Multi-AZ DB cluster snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.
5. On the **Restore snapshot** page, in **Availability and durability**, choose one of the following:
   - **Single DB instance** – Restores the snapshot to one DB instance with no standby DB instance.
   - **Multi-AZ DB instance** – Restores the snapshot to a Multi-AZ DB instance deployment with one primary DB instance and one standby DB instance.
6. For **DB instance identifier**, enter the name for your restored DB instance.
7. For the remaining sections, specify your DB instance settings. For information about each setting, see Settings for DB instances.

8. Choose **Restore DB instance**.

AWS CLI

To restore a Multi-AZ DB cluster snapshot to a DB instance deployment, use the AWS CLI command restore-db-instance-from-db-snapshot.

In the following example, you restore from a previously created Multi-AZ DB cluster snapshot named `myclustersnapshot`. You restore to a new Multi-AZ DB instance deployment with a primary DB instance named `mynewdbinstance`. For the `--db-cluster-snapshot-identifier` option, specify the name of the Multi-AZ DB cluster snapshot.

For the `--db-instance-class` option, specify the DB instance class for the new DB instance deployment. For more information about DB instance classes, see DB instance classes.

You can also specify other options.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier mynewdbinstance \
    --db-cluster-snapshot-identifier myclustersnapshot \
    --engine mysql \
    --multi-az \
    --db-instance-class db.r6g.xlarge
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^
    --db-instance-identifier mynewdbinstance ^
    --db-cluster-snapshot-identifier myclustersnapshot ^
    --engine mysql ^
    --multi-az ^
    --db-instance-class db.r6g.xlarge
```

After you restore the DB instance, you can add it to the security group associated with the Multi-AZ DB cluster that you used to create the snapshot, if applicable. Completing this action provides the same functions of the previous Multi-AZ DB cluster.

RDS API

To restore a Multi-AZ DB cluster snapshot to a DB instance deployment, call the RDS API operation RestoreDBInstanceFromDBSnapshot with the following parameters:

- `DBInstanceIdentifier`
- `DBClusterSnapshotIdentifier`
- `Engine`

You can also specify other optional parameters.

After you restore the DB instance, you can add it to the security group associated with the Multi-AZ DB cluster that you used to create the snapshot, if applicable. Completing this action provides the same functions of the previous Multi-AZ DB cluster.

Tutorial: Restore an Amazon RDS DB instance from a DB snapshot

Often, when working with Amazon RDS you might have a DB instance that you work with occasionally but don't need full time. For example, suppose that you have a quarterly customer survey that uses an Amazon EC2 instance to host a customer survey website. You also have a DB instance that is used to store the survey results. One way to save money on such a scenario is to take a DB snapshot of the DB instance after the survey is completed. You then delete the DB instance and restore it when you need to conduct the survey again.

When you restore the DB instance, you provide the name of the DB snapshot to restore from. You then provide a name for the new DB instance that's created from the restore operation.

For more detailed information on restoring DB instances from snapshots, see Restoring to a DB instance.

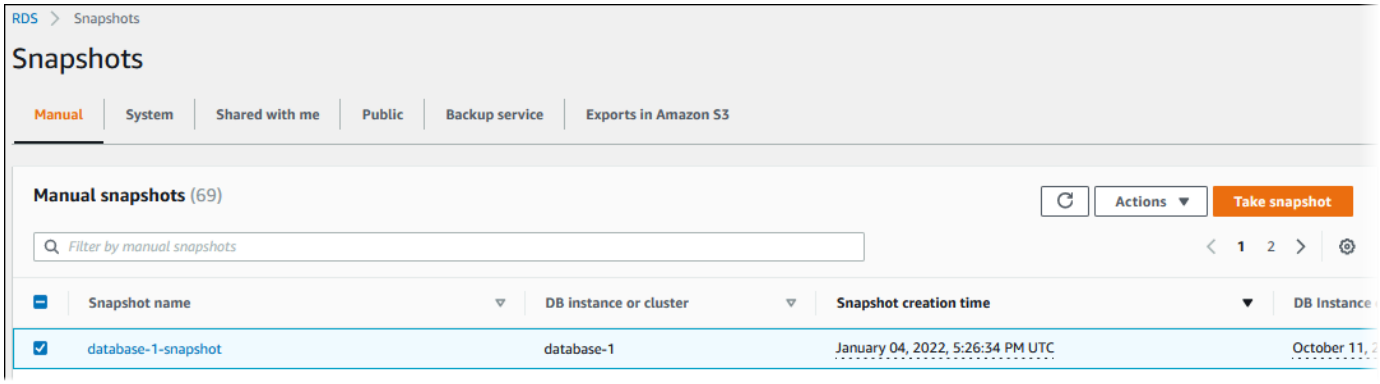
For information about AWS KMS key management for Amazon RDS, see AWS KMS key management.

Restoring a DB instance from a DB snapshot

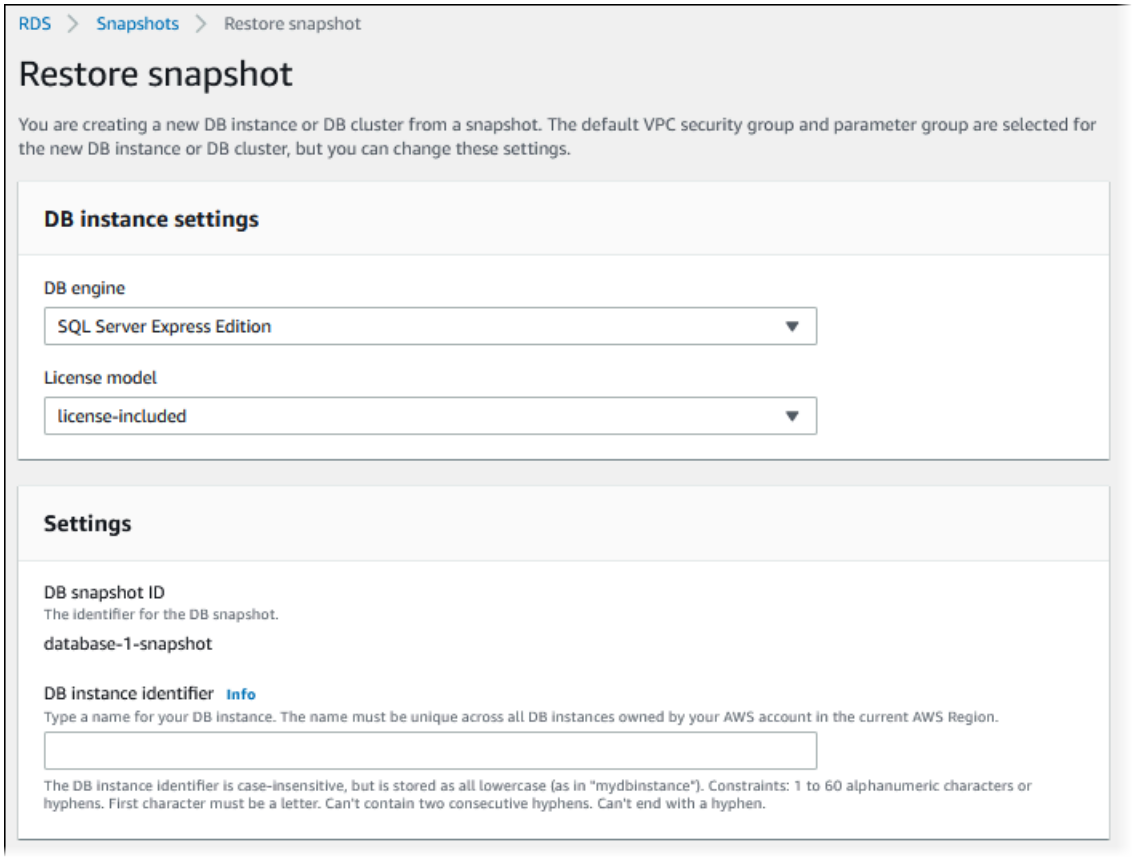
Use the following procedure to restore from a snapshot in the AWS Management Console.

To restore a DB instance from a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.



The **Restore snapshot** page appears.



5. Under **DB instance settings**, use the default settings for **DB engine** and **License model** (for Oracle or Microsoft SQL Server).
6. Under **Settings**, for **DB instance identifier** enter the unique name that you want to use for the restored DB instance, for example **`mynewdbinstance`**.

   If you're restoring from a DB instance that you deleted after you made the DB snapshot, you can use the name of that DB instance.
7. Under **Availability & durability**, choose whether to create a standby instance in another Availability Zone.

   For this tutorial, don't create a standby instance.
8. Under **Connectivity**, use the default settings for the following:

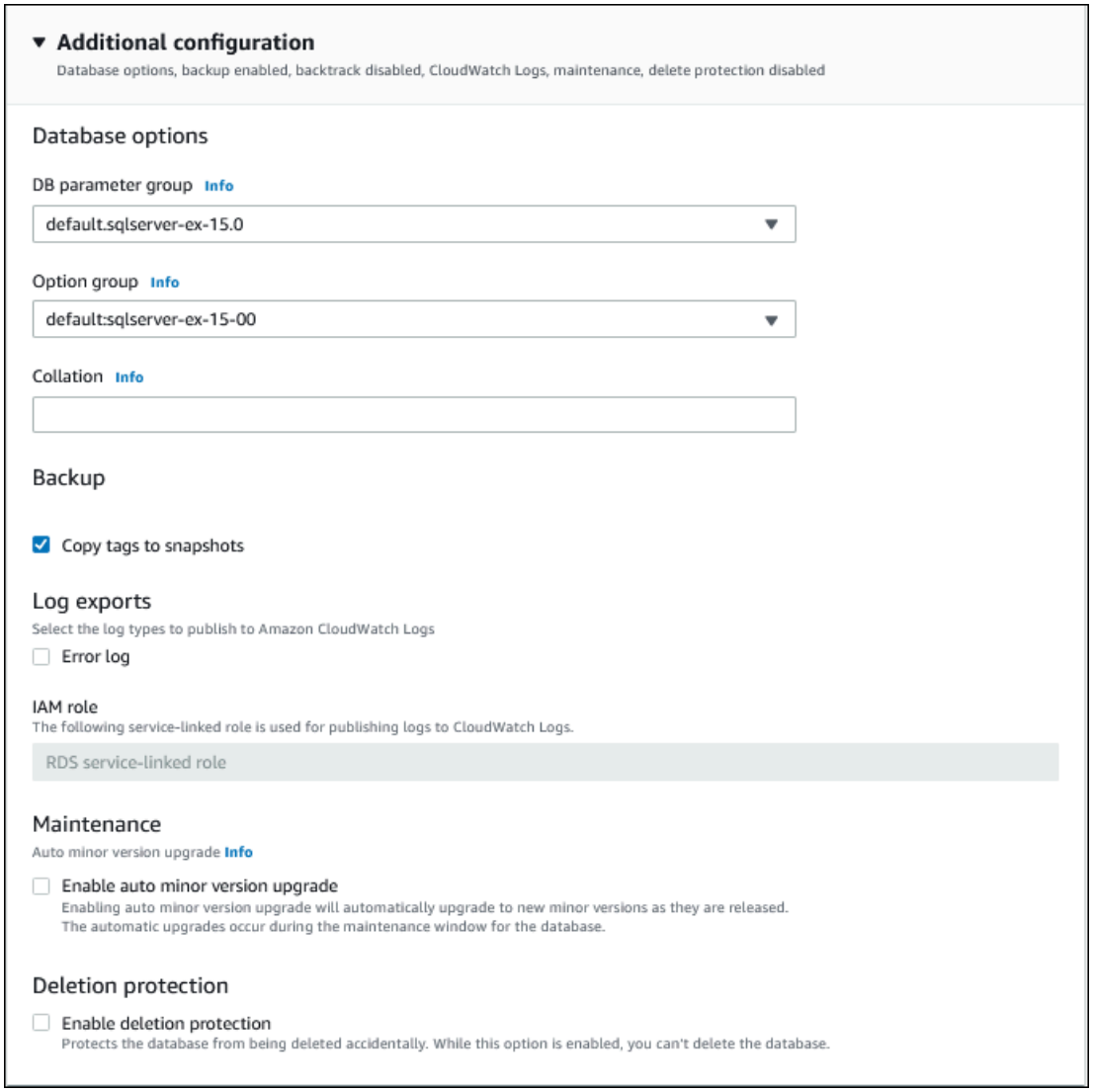
   - **Virtual private cloud (VPC)**
   - **DB subnet group**
   - **Public access**
   - **VPC security group (firewall)**
9. Choose the **DB instance class**.

For this tutorial, choose **Burstable classes (includes t classes)**, and then choose **db.t3.small**.

10. For **Encryption**, use the default settings.

    If the source DB instance for the snapshot was encrypted, the restored DB instance is also encrypted. You can't make it unencrypted.

11. Expand **Additional configuration** at the bottom of the page.



12. Do the following under **Database options**:

    a. Choose the **DB parameter group**.

       For this tutorial, use the default parameter group.

    b. Choose the **Option group**.

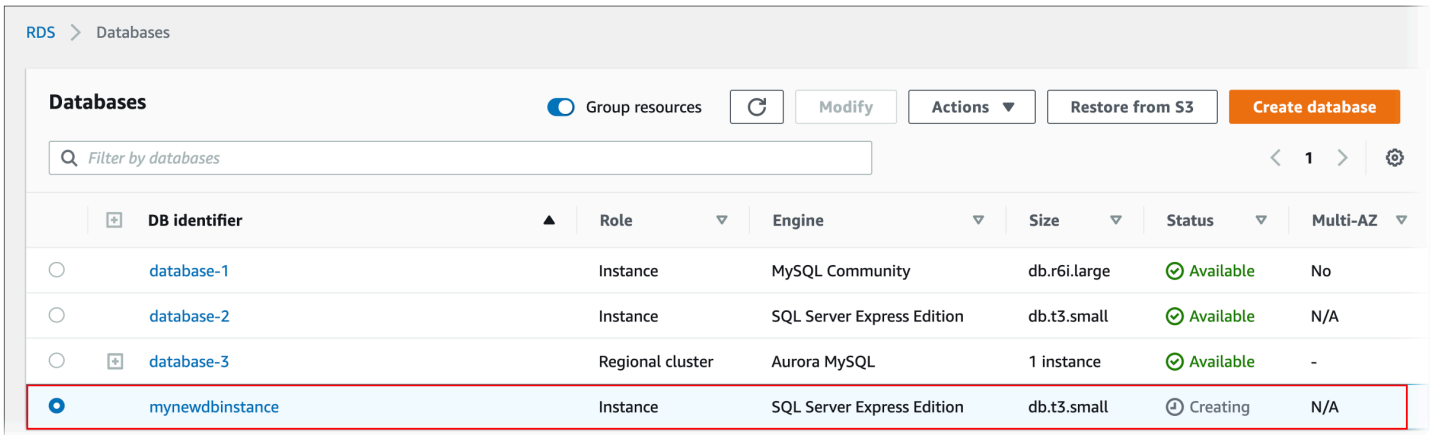
For this tutorial, use the default option group.

> **Important**
>
> In some cases, you might restore from a DB snapshot of a DB instance that uses a persistent or permanent option. If so, make sure to choose an option group that uses the same option.

c. For **Deletion protection**, choose the **Enable deletion protection** check box.

13. Choose **Restore DB instance**.

The **Databases** page displays the restored DB instance, with a status of `Creating`.



Copying a DB snapshot for Amazon RDS

With Amazon RDS, you can copy automated backups or manual DB snapshots. After you copy a snapshot, the copy is a manual snapshot. You can make multiple copies of an automated backup or manual snapshot, but each copy must have a unique identifier.

You can copy a snapshot within the same AWS Region, you can copy a snapshot across AWS Regions, and you can copy shared snapshots. You can copy snapshots to another AWS Region or account in a single step.

Copying a DB snapshot

For each AWS account, you can copy up to 20 DB snapshots at a time from one AWS Region to another. If you copy a DB snapshot to another AWS Region, you create a manual DB snapshot that is retained in that AWS Region. Copying a DB snapshot out of the source AWS Region incurs Amazon RDS data transfer charges.

For more information about data transfer pricing, see Amazon RDS pricing.

After the DB snapshot copy has been created in the new AWS Region, the DB snapshot copy behaves the same as all other DB snapshots in that AWS Region.

You can copy a DB snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

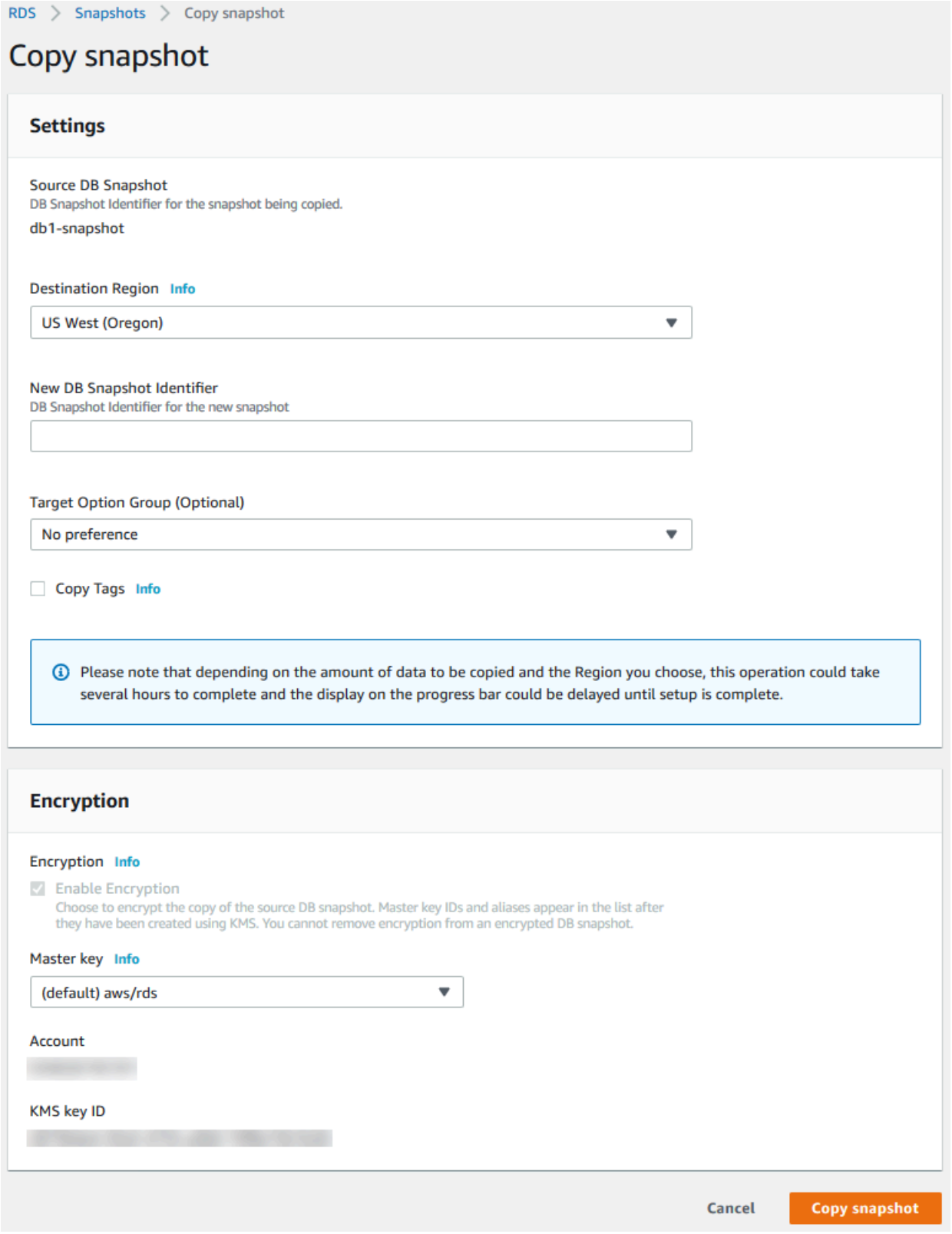
Console

The following procedure copies an encrypted or unencrypted DB snapshot, in the same AWS Region or across Regions, by using the AWS Management Console.

To copy a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Select the DB snapshot that you want to copy.
4. For **Actions**, choose **Copy snapshot**.

   The **Copy snapshot** page appears.



5. For **Target option group (optional)**, choose a new option group if you want.

Specify this option if you are copying a snapshot from one AWS Region to another, and your DB instance uses a nondefault option group.

If your source DB instance uses Transparent Data Encryption for Oracle or Microsoft SQL Server, you must specify this option when copying across Regions. For more information, see Considerations for option groups.

6. (Optional) To copy the DB snapshot to a different AWS Region, for **Destination Region**, choose the new AWS Region.

   > **Note**
   >
   > The destination AWS Region must have the same database engine version available as the source AWS Region.

7. For **New DB snapshot identifier**, type the name of the DB snapshot copy.

   You can make multiple copies of an automated backup or manual snapshot, but each copy must have a unique identifier.

8. (Optional) Select **Copy Tags** to copy tags and values from the snapshot to the copy of the snapshot.

9. (Optional) For **Encryption**, do the following:

   a. Choose **Enable Encryption** if the DB snapshot isn't encrypted but you want to encrypt the copy.

      > **Note**
      >
      > If the DB snapshot is encrypted, you must encrypt the copy, so the check box is already selected.

   b. For **AWS KMS key**, specify the KMS key identifier to use to encrypt the DB snapshot copy.

10. Choose **Copy snapshot**.

AWS CLI

You can copy a DB snapshot by using the AWS CLI command copy-db-snapshot. If you are copying the snapshot to a new AWS Region, run the command in the new AWS Region.

The following options are used to copy a DB snapshot. Not all options are required for all scenarios. Use the descriptions and the examples that follow to determine which options to use.

- `--source-db-snapshot-identifier` – The identifier for the source DB snapshot.
  - If the source snapshot is in the same AWS Region as the copy, specify a valid DB snapshot identifier. For example, `rds:mysql-instance1-snapshot-20130805`.
  - If the source snapshot is in the same AWS Region as the copy, and has been shared with your AWS account, specify a valid DB snapshot ARN. For example, `arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20130805`.
  - If the source snapshot is in a different AWS Region than the copy, specify a valid DB snapshot ARN. For example, `arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20130805`.
  - If you are copying from a shared manual DB snapshot, this parameter must be the Amazon Resource Name (ARN) of the shared DB snapshot.
  - If you are copying an encrypted snapshot this parameter must be in the ARN format for the source AWS Region, and must match the `SourceDBSnapshotIdentifier` in the `PreSignedUrl` parameter.
- `--target-db-snapshot-identifier` – The identifier for the new copy of the encrypted DB snapshot.
- `--copy-option-group` – Copy the option group from a snapshot that has been shared with your AWS account.
- `--copy-tags` – Include the copy tags option to copy tags and values from the snapshot to the copy of the snapshot.
- `--option-group-name` – The option group to associate with the copy of the snapshot.

  Specify this option if you are copying a snapshot from one AWS Region to another, and your DB instance uses a non-default option group.

  If your source DB instance uses Transparent Data Encryption for Oracle or Microsoft SQL Server, you must specify this option when copying across Regions. For more information, see Considerations for option groups.
- `--kms-key-id` – The KMS key identifier for an encrypted DB snapshot. The KMS key identifier is the Amazon Resource Name (ARN), key identifier, or key alias for the KMS key.
  - If you copy an encrypted DB snapshot from your AWS account, you can specify a value for this parameter to encrypt the copy with a new KMS key. If you don't specify a value for this

parameter, then the copy of the DB snapshot is encrypted with the same KMS key as the source DB snapshot.

- If you copy an encrypted DB snapshot that is shared from another AWS account, then you must specify a value for this parameter.
- If you specify this parameter when you copy an unencrypted snapshot, the copy is encrypted.
- If you copy an encrypted snapshot to a different AWS Region, then you must specify a KMS key for the destination AWS Region. KMS keys are specific to the AWS Region that they are created in, and you cannot use encryption keys from one AWS Region in another AWS Region.

Example from unencrypted, to the same Region

The following code creates a copy of a snapshot, with the new name `mydbsnapshotcopy`, in the same AWS Region as the source snapshot. When the copy is made, the DB option group and tags on the original snapshot are copied to the snapshot copy.

For Linux, macOS, or Unix:

```
aws rds copy-db-snapshot \
    --source-db-snapshot-identifier arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20130805 \
    --target-db-snapshot-identifier mydbsnapshotcopy \
    --copy-option-group \
    --copy-tags
```

For Windows:

```
aws rds copy-db-snapshot ^
    --source-db-snapshot-identifier arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20130805 ^
    --target-db-snapshot-identifier mydbsnapshotcopy ^
    --copy-option-group ^
    --copy-tags
```

Example from unencrypted, across Regions

The following code creates a copy of a snapshot, with the new name `mydbsnapshotcopy`, in the AWS Region in which the command is run.

For Linux, macOS, or Unix:

```
aws rds copy-db-snapshot \
    --source-db-snapshot-identifier arn:aws:rds:us-east-1:123456789012:snapshot:mysql-instance1-snapshot-20130805 \
    --target-db-snapshot-identifier mydbsnapshotcopy
```

For Windows:

```
aws rds copy-db-snapshot ^
    --source-db-snapshot-identifier arn:aws:rds:us-east-1:123456789012:snapshot:mysql-instance1-snapshot-20130805 ^
    --target-db-snapshot-identifier mydbsnapshotcopy
```

Example from encrypted, across Regions

The following code example copies an encrypted DB snapshot from the US West (Oregon) Region in the US East (N. Virginia) Region. Run the command in the destination (us-east-1) Region.

For Linux, macOS, or Unix:

```
aws rds copy-db-snapshot \
    --source-db-snapshot-identifier arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20161115 \
    --target-db-snapshot-identifier mydbsnapshotcopy \
    --kms-key-id my-us-east-1-key \
    --option-group-name custom-option-group-name
```

For Windows:

```
aws rds copy-db-snapshot ^
    --source-db-snapshot-identifier arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20161115 ^
    --target-db-snapshot-identifier mydbsnapshotcopy ^
    --kms-key-id my-us-east-1-key ^
    --option-group-name custom-option-group-name
```

The `--source-region` parameter is required when you're copying an encrypted snapshot between the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions. For `--source-region`, specify the AWS Region of the source DB instance.

If `--source-region` isn't specified, specify a `--pre-signed-url` value. A *presigned URL* is a URL that contains a Signature Version 4 signed request for the `copy-db-snapshot` command that's

called in the source AWS Region. To learn more about the `pre-signed-url` option, see copy-db-snapshot in the *AWS CLI Command Reference*.

RDS API

You can copy a DB snapshot by using the Amazon RDS API operation CopyDBSnapshot. If you are copying the snapshot to a new AWS Region, perform the action in the new AWS Region.

The following parameters are used to copy a DB snapshot. Not all parameters are required for all scenarios. Use the descriptions and the examples that follow to determine which parameters to use.

- `SourceDBSnapshotIdentifier` – The identifier for the source DB snapshot.
  - If the source snapshot is in the same AWS Region as the copy, specify a valid DB snapshot identifier. For example, `rds:mysql-instance1-snapshot-20130805`.
  - If the source snapshot is in the same AWS Region as the copy, and has been shared with your AWS account, specify a valid DB snapshot ARN. For example, `arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20130805`.
  - If the source snapshot is in a different AWS Region than the copy, specify a valid DB snapshot ARN. For example, `arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20130805`.
  - If you are copying from a shared manual DB snapshot, this parameter must be the Amazon Resource Name (ARN) of the shared DB snapshot.
  - If you are copying an encrypted snapshot this parameter must be in the ARN format for the source AWS Region, and must match the `SourceDBSnapshotIdentifier` in the `PreSignedUrl` parameter.
- `TargetDBSnapshotIdentifier` – The identifier for the new copy of the encrypted DB snapshot.
- `CopyOptionGroup` – Set this parameter to `true` to copy the option group from a shared snapshot to the copy of the snapshot. The default is `false`.
- `CopyTags` – Set this parameter to `true` to copy tags and values from the snapshot to the copy of the snapshot. The default is `false`.
- `OptionGroupName` – The option group to associate with the copy of the snapshot.

  Specify this parameter if you are copying a snapshot from one AWS Region to another, and your DB instance uses a non-default option group.

If your source DB instance uses Transparent Data Encryption for Oracle or Microsoft SQL Server, you must specify this parameter when copying across Regions. For more information, see Considerations for option groups.

- `KmsKeyId` – The KMS key identifier for an encrypted DB snapshot. The KMS key identifier is the Amazon Resource Name (ARN), key identifier, or key alias for the KMS key.
  - If you copy an encrypted DB snapshot from your AWS account, you can specify a value for this parameter to encrypt the copy with a new KMS key. If you don't specify a value for this parameter, then the copy of the DB snapshot is encrypted with the same KMS key as the source DB snapshot.
  - If you copy an encrypted DB snapshot that is shared from another AWS account, then you must specify a value for this parameter.
  - If you specify this parameter when you copy an unencrypted snapshot, the copy is encrypted.
  - If you copy an encrypted snapshot to a different AWS Region, then you must specify a KMS key for the destination AWS Region. KMS keys are specific to the AWS Region that they are created in, and you can't use encryption keys from one AWS Region in another AWS Region.
- `PreSignedUrl` – The URL that contains a Signature Version 4 signed request for the `CopyDBSnapshot` API operation in the source AWS Region that contains the source DB snapshot to copy.

  Specify this parameter when you copy an encrypted DB snapshot from another AWS Region by using the Amazon RDS API. You can specify the source Region option instead of this parameter when you copy an encrypted DB snapshot from another AWS Region by using the AWS CLI.

  The presigned URL must be a valid request for the `CopyDBSnapshot` API operation that can be run in the source AWS Region containing the encrypted DB snapshot to be copied. The presigned URL request must contain the following parameter values:

  - `DestinationRegion` – The AWS Region that the encrypted DB snapshot will be copied to. This AWS Region is the same one where the `CopyDBSnapshot` operation is called that contains this presigned URL.

    For example, suppose that you copy an encrypted DB snapshot from the us-west-2 Region to the us-east-1 Region. You then call the `CopyDBSnapshot` operation in the us-east-1 Region and provide a presigned URL that contains a call to the `CopyDBSnapshot` operation in the us-west-2 Region. For this example, the `DestinationRegion` in the presigned URL must be set to the us-east-1 Region.

- `KmsKeyId` – The KMS key identifier for the key to use to encrypt the copy of the DB snapshot in the destination AWS Region. This is the same identifier for both the `CopyDBSnapshot` operation that is called in the destination AWS Region, and the operation contained in the presigned URL.
- `SourceDBSnapshotIdentifier` – The DB snapshot identifier for the encrypted snapshot to be copied. This identifier must be in the Amazon Resource Name (ARN) format for the source AWS Region. For example, if you're copying an encrypted DB snapshot from the us-west-2 Region, then your `SourceDBSnapshotIdentifier` looks like the following example: `arn:aws:rds:us-west-2:123456789012:snapshot:mysql-instance1-snapshot-20161115`.

For more information on Signature Version 4 signed requests, see the following:

- Authenticating requests: Using query parameters (AWS signature version 4) in the Amazon Simple Storage Service API Reference
- Signature version 4 signing process in the AWS General Reference

Example from unencrypted, to the same Region

The following code creates a copy of a snapshot, with the new name `mydbsnapshotcopy`, in the same AWS Region as the source snapshot. When the copy is made, all tags on the original snapshot are copied to the snapshot copy.

```
https://rds.us-west-1.amazonaws.com/
 ?Action=CopyDBSnapshot
 &CopyTags=true
 &SignatureMethod=HmacSHA256
 &SignatureVersion=4
 &SourceDBSnapshotIdentifier=mysql-instance1-snapshot-20130805
 &TargetDBSnapshotIdentifier=mydbsnapshotcopy
 &Version=2013-09-09
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
 &X-Amz-Date=20140429T175351Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Example from unencrypted, across Regions

The following code creates a copy of a snapshot, with the new name mydbsnapshotcopy, in the US West (N. California) Region.

```
https://rds.us-west-1.amazonaws.com/
 ?Action=CopyDBSnapshot
 &SignatureMethod=HmacSHA256
 &SignatureVersion=4
 &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Asnapshot
%3Amysql-instance1-snapshot-20130805
 &TargetDBSnapshotIdentifier=mydbsnapshotcopy
 &Version=2013-09-09
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
 &X-Amz-Date=20140429T175351Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Example from encrypted, across Regions

The following code creates a copy of a snapshot, with the new name mydbsnapshotcopy, in the US East (N. Virginia) Region.

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CopyDBSnapshot
    &KmsKeyId=my-us-east-1-key
    &OptionGroupName=custom-option-group-name
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCopyDBSnapshot
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526SourceDBSnapshotIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Asnapshot%25253Amysql-instance1-snapshot-20161115
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
```

```
        %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-west-2%3A123456789012%3Asnapshot
%3Amysql-instance1-snapshot-20161115
    &TargetDBSnapshotIdentifier=mydbsnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

Limitations

The following are some limitations when you copy snapshots:

- You can't copy a snapshot to or from the China (Beijing) Region or the China (Ningxia) Region.
- You can copy a snapshot between AWS GovCloud (US-East) and AWS GovCloud (US-West). However, you can't copy a snapshot between these GovCloud (US) Regions and Regions that aren't GovCloud (US) Regions.
- If you delete a source snapshot before the target snapshot becomes available, the snapshot copy might fail. Verify that the target snapshot has a status of `AVAILABLE` before you delete a source snapshot.
- You can have up to 20 snapshot copy requests in progress to a single destination Region per account.
- When you request multiple snapshot copies for the same source DB instance, they're queued internally. The copies requested later won't start until the previous snapshot copies are completed. For more information, see Why is my EC2 AMI or EBS snapshot creation slow? in the AWS Knowledge Center.
- Depending on the AWS Regions involved and the amount of data to be copied, a cross-Region snapshot copy can take hours to complete. In some cases, there might be a large number of cross-Region snapshot copy requests from a given source Region. In such cases, Amazon RDS might put new cross-Region copy requests from that source Region into a queue until some in-

progress copies complete. No progress information is displayed about copy requests while they are in the queue. Progress information is displayed when the copy starts.

- If a copy is still pending when you start another copy, the second copy doesn't start until the first copy finishes.
- You can't copy a snapshot of a Multi-AZ DB cluster.
- You can only copy snapshots of DB instances that use io2 volumes to AWS Regions where io2 Block Express volumes are available. For more information, see Amazon RDS DB instance storage.

Considerations

For considerations when copying a DB snapshot, see the following topics.

Topics

- Snapshot retention
- Considerations for shared snapshot copying
- Considerations for encrypted snapshot copying
- Considerations for incremental snapshot copying
- Considerations for cross-Region snapshot copying
- Considerations for option groups
- Considerations for parameter group

Snapshot retention

Amazon RDS deletes automated backups in several situations:

- At the end of their retention period.
- When you disable automated backups for a DB instance.
- When you delete a DB instance.

If you want to keep an automated backup for a longer period, copy it to create a manual snapshot, which is retained until you delete it. Amazon RDS storage costs might apply to manual snapshots if they exceed your default storage space.

For more information about backup storage costs, see Amazon RDS pricing.

Considerations for shared snapshot copying

You can copy snapshots shared to you by other AWS accounts. In some cases, you might copy an encrypted snapshot that has been shared from another AWS account. In these cases, you must have access to the AWS KMS key that was used to encrypt the snapshot.

> **Note**
>
> Amazon RDS storage costs apply to shared snapshots you copy. Amazon RDS might attach the ARN of the source DB instance to the snapshot you copied.

You can copy a shared DB snapshot across AWS Regions if the snapshot is unencrypted.

> **Note**
>
> Copying shared incremental snapshots in the same AWS Region is supported when they're unencrypted, or encrypted using the same KMS key as the initial full snapshot. If you use a different KMS key to encrypt subsequent snapshots when copying them, those shared snapshots are full snapshots. For more information, see Considerations for incremental snapshot copying.

Cross-Region and cross-account copy in a single step

To copy a snapshot cross-Region and cross-account in a single action, you must first share the snapshot with the target AWS account. If the snapshot is encrypted, you must also share the AWS KMS key with the target AWS account. If the snapshot is encrypted with the default AWS KMS key, you must first copy the snapshot to re-encrypt it with a customer managed key before sharing it with the target account. Once shared, you can initiate a copy to that account (in-Region or cross-Region) from the target account.

Considerations for encrypted snapshot copying

You can copy a snapshot that has been encrypted using a KMS key. If you copy an encrypted snapshot, the copy of the snapshot must also be encrypted. If you copy an encrypted snapshot within the same AWS Region, you can encrypt the copy with the same KMS key as the original snapshot. Or you can specify a different KMS key.

If you copy an encrypted snapshot across Regions, you must specify a KMS key valid in the destination AWS Region. It can be a Region-specific KMS key, or a multi-Region key. For more information on multi-Region KMS keys, see Using multi-Region keys in AWS KMS.

The source snapshot remains encrypted throughout the copy process. For more information, see Limitations of Amazon RDS encrypted DB instances.

You can also encrypt a copy of an unencrypted snapshot. This way, you can quickly add encryption to a previously unencrypted DB instance. To do this, you create a snapshot of your DB instance when you are ready to encrypt it. You then create a copy of that snapshot and specify a KMS key to encrypt that snapshot copy. You can then restore an encrypted DB instance from the encrypted snapshot.

Important

When you use explicit deny statements for all resources (*) in AWS KMS key policies with managed services like Amazon RDS, you must specify a condition to allow the resource owning account. The copy operation might fail without this condition, even if the deny rule includes exceptions for your IAM user.

For more information about AWS KMS key management for Amazon RDS, see AWS KMS key management.

Required permissions for copying encrypted snapshots

To copy an encrypted DB snapshot, your user must have the following permissions to use Amazon RDS encryption.

- `kms:DescribeKey`
- `kms:CreateGrant`
- `kms:Decrypt`
- `kms:Encrypt`
- `kms:GenerateDataKey`
- `kms:GenerateDataKeyWithoutPlaintext`
- `kms:ReEncrypt`

If you use IAM policy conditions to restrict AWS KMS grant operations, ensure your policy includes all operations required by Amazon RDS. If you receive a `KMSKeyNotAccessibleFault` error when copying an encrypted snapshot across Regions, verify that your IAM policy includes all required AWS KMS grant operations mentioned above.

Considerations for incremental snapshot copying

An *incremental* snapshot contains only the data that has changed after the most recent snapshot of the same DB instance. Incremental snapshot copying is faster and results in lower storage costs than full snapshot copying.

Whether a snapshot copy is incremental is determined by the most recently completed snapshot copy and the source snapshot. If the most recent snapshot copy was deleted, the next copy is a full copy, not an incremental copy. A snapshot copy will be the same type as the source snapshot. If the source snapshot is an incremental snapshot, then the snapshot copy will be an incremental snapshot. Incrementality is also determined by the number of changes that have taken place in the source DB instance since the most recent snapshot.

When you copy a snapshot across AWS accounts, the copy is an incremental copy only if all of the following conditions are met:

- The most recent snapshot copy is of the same source DB instance and still exists in the destination account.
- All copies of the snapshot in the destination account are either unencrypted, or were encrypted using the same KMS key. If they are encrypted, then they must have the same lineage i.e matching inheritance depth and consistent encryption keys at corresponding levels.
- If the source DB instance is a Multi-AZ instance, it hasn't failed over to another AZ since the last snapshot was taken from it.

The following examples illustrate the difference between full and incremental snapshots. They apply to both shared and unshared snapshots.

Snapshot	Encryption key	Full or incremental
S1	K1	Full
S2	K1	Incremental of S1

Snapshot	Encryption key	Full or incremental
S3	K1	Incremental of S2
S4	K1	Incremental of S3
Copy of S1 (S1C)	K2	Full
Copy of S2 (S2C)	K3	Full
Copy of S3 (S3C)	K3	Incremental of S2C
Copy of S4 (S4C)	K3	Incremental of S3C
Copy 2 of S4 (S4C2)	K4	Full

Note

In these examples, snapshots S2, S3, and S4 are incremental only if the previous snapshot still exists.
The same applies to copies. Snapshot copies S3C and S4C are incremental only if the previous copy still exists.

For information on copying incremental snapshots across AWS Regions, see Full and incremental copies.

Considerations for cross-Region snapshot copying

You can copy DB snapshots across AWS Regions. Data transfer charges apply for cross-Region snapshot copy. Cross-Region snapshot copying is subject to certain constraints and considerations.

Requesting a cross-Region DB snapshot copy

To communicate with the source Region to request a cross-Region DB snapshot copy, the requester (IAM role or IAM user) must have access to the source DB snapshot and the source Region.

Certain conditions in the requester's IAM policy can cause the request to fail. The following examples assume that you're copying the DB snapshot from US East (Ohio) to US East (N. Virginia). These examples show conditions in the requester's IAM policy that cause the request to fail:

- The requester's policy has a condition for `aws:RequestedRegion`.

```
...
"Effect": "Allow",
"Action": "rds:CopyDBSnapshot",
"Resource": "*",
"Condition": {
    "StringEquals": {
        "aws:RequestedRegion": "us-east-1"
    }
}
```

  The request fails because the policy doesn't allow access to the source Region. For a successful request, specify both the source and destination Regions.

```
...
"Effect": "Allow",
"Action": "rds:CopyDBSnapshot",
"Resource": "*",
"Condition": {
    "StringEquals": {
        "aws:RequestedRegion": [
            "us-east-1",
            "us-east-2"
        ]
    }
}
```

- The requester's policy doesn't allow access to the source DB snapshot.

```
...
"Effect": "Allow",
"Action": "rds:CopyDBSnapshot",
"Resource": "arn:aws:rds:us-east-1:123456789012:snapshot:target-snapshot"
...
```

  For a successful request, specify both the source and target snapshots.

```
...
"Effect": "Allow",
"Action": "rds:CopyDBSnapshot",
```

```
"Resource": [
    "arn:aws:rds:us-east-1:123456789012:snapshot:target-snapshot",
    "arn:aws:rds:us-east-2:123456789012:snapshot:source-snapshot"
]
...
```

- The requester's policy denies `aws:ViaAWSService`.

  ```
  ...
  "Effect": "Allow",
  "Action": "rds:CopyDBSnapshot",
  "Resource": "*",
  "Condition": {
      "Bool": {"aws:ViaAWSService": "false"}
  }
  ```

  Communication with the source Region is made by RDS on the requester's behalf. For a successful request, don't deny calls made by AWS services.

- The requester's policy has a condition for `aws:SourceVpc` or `aws:SourceVpce`.

  These requests might fail because when RDS makes the call to the remote Region, it isn't from the specified VPC or VPC endpoint.

If you need to use one of the previous conditions that would cause a request to fail, you can include a second statement with `aws:CalledVia` in your policy to make the request succeed. For example, you can use `aws:CalledVia` with `aws:SourceVpce` as shown here:

```
...
"Effect": "Allow",
"Action": "rds:CopyDBSnapshot",
"Resource": "*",
"Condition": {
    "Condition" : {
        "ForAnyValue:StringEquals" : {
          "aws:SourceVpce": "vpce-1a2b3c4d"
        }
     }
},
{
    "Effect": "Allow",
    "Action": [
```

```
            "rds:CopyDBSnapshot"
        ],
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "aws:CalledVia": [
                    "rds.amazonaws.com"
                ]
            }
        }
}
```

For more information, see Policies and permissions in IAM in the *IAM User Guide*.

Authorizing the snapshot copy

After a cross-Region DB snapshot copy request returns `success`, RDS starts the copy in the background. An authorization for RDS to access the source snapshot is created. This authorization links the source DB snapshot to the target DB snapshot, and allows RDS to copy only to the specified target snapshot.

The authorization is verified by RDS using the `rds:CrossRegionCommunication` permission in the service-linked IAM role. If the copy is authorized, RDS communicates with the source Region and completes the copy.

RDS doesn't have access to DB snapshots that weren't authorized previously by a `CopyDBSnapshot` request. The authorization is revoked when copying completes.

RDS uses the service-linked role to verify the authorization in the source Region. If you delete the service-linked role during the copy process, the copy fails.

For more information, see Using service-linked roles in the *IAM User Guide*.

Using AWS Security Token Service credentials

Session tokens from the global AWS Security Token Service (AWS STS) endpoint are valid only in AWS Regions that are enabled by default (commercial Regions). If you use credentials from the `assumeRole` API operation in AWS STS, use the regional endpoint if the source Region is an opt-in Region. Otherwise, the request fails. This happens because your credentials must be valid in both Regions, which is true for opt-in Regions only when the regional AWS STS endpoint is used.

To use the global endpoint, make sure that it's enabled for both Regions in the operations. Set the global endpoint to `Valid in all AWS Regions` in the AWS STS account settings.

The same rule applies to credentials in the presigned URL parameter.

For more information, see Managing AWS STS in an AWS Region in the *IAM User Guide*.

Latency and multiple copy requests

Depending on the AWS Regions involved and the amount of data to be copied, a cross-Region snapshot copy can take hours to complete.

In some cases, there might be a large number of cross-Region snapshot copy requests from a given source AWS Region. In such cases, Amazon RDS might put new cross-Region copy requests from that source AWS Region into a queue until some in-progress copies complete. No progress information is displayed about copy requests while they are in the queue. Progress information is displayed when the copying starts.

Full and incremental copies

When you copy a snapshot to a different AWS Region from the source snapshot, the first copy is a full snapshot copy, even if you copy an incremental snapshot. A full snapshot copy contains all of the data and metadata required to restore the DB instance. After the first snapshot copy, you can copy incremental snapshots of the same DB instance to the same destination Region within the same AWS account. For more information on incremental snapshots, see Considerations for incremental snapshot copying.

Incremental snapshot copying across AWS Regions is supported for both unencrypted and encrypted snapshots.

When you copy a snapshot across AWS Regions, the copy is an incremental copy if the following conditions are met:

- The snapshot was previously copied to the destination Region.
- The most recent snapshot copy still exists in the destination Region.
- All copies of the snapshot in the destination Region are either unencrypted, or were encrypted using the same KMS key.

Considerations for option groups

DB option groups are specific to the AWS Region that they are created in, and you can't use an option group from one AWS Region in another AWS Region.

For Oracle databases, you can use the AWS CLI or RDS API to copy the custom DB option group from a snapshot that has been shared with your AWS account. You can only copy option groups within the same AWS Region. The option group isn't copied if it has already been copied to the destination account and no changes have been made to it since being copied. If the source option group has been copied before, but has changed since being copied, RDS copies the new version to the destination account. Default option groups aren't copied.

When you copy a snapshot across Regions, you can specify a new option group for the snapshot. We recommend that you prepare the new option group before you copy the snapshot. In the destination AWS Region, create an option group with the same settings as the original DB instance. If one already exists in the new AWS Region, you can use that one.

In some cases, you might copy a snapshot and not specify a new option group for the snapshot. In these cases, when you restore the snapshot the DB instance gets the default option group. To give the new DB instance the same options as the original, do the following:

1. In the destination AWS Region, create an option group with the same settings as the original DB instance. If one already exists in the new AWS Region, you can use that one.
2. After you restore the snapshot in the destination AWS Region, modify the new DB instance and add the new or existing option group from the previous step.

Considerations for parameter group

When you copy a snapshot across Regions, the copy doesn't include the parameter group used by the original DB instance. When you restore a snapshot to create a new DB instance, that DB instance gets the default parameter group for the AWS Region it is created in. To give the new DB instance the same parameters as the original, do the following:

1. In the destination AWS Region, create a DB parameter group with the same settings as the original DB instance. If one already exists in the new AWS Region, you can use that one.
2. After you restore the snapshot in the destination AWS Region, modify the new DB instance and add the new or existing parameter group from the previous step.

Sharing a DB snapshot for Amazon RDS

Using Amazon RDS, you can share a manual DB snapshot in the following ways:

- Sharing a manual DB snapshot, whether encrypted or unencrypted, enables authorized AWS accounts to copy the snapshot.
- Sharing an unencrypted manual DB snapshot enables authorized AWS accounts to directly restore a DB instance from the snapshot instead of taking a copy of it and restoring from that. However, you can't restore a DB instance from a DB snapshot that is both shared and encrypted. Instead, you can make a copy of the DB snapshot and restore the DB instance from the copy.

> **Note**
>
> To share an automated DB snapshot, create a manual DB snapshot by copying the automated snapshot, and then share that copy. This process also applies to AWS Backup–generated resources.

For more information on copying a snapshot, see Copying a DB snapshot for Amazon RDS. For more information on restoring a DB instance from a DB snapshot, see Restoring to a DB instance.

You can share a manual snapshot with up to 20 other AWS accounts.

The following limitations apply when you share manual snapshots with other AWS accounts:

- When you restore a DB instance from a shared snapshot using the AWS Command Line Interface (AWS CLI) or Amazon RDS API, you must specify the Amazon Resource Name (ARN) of the shared snapshot as the snapshot identifier.
- You can't share a DB snapshot that uses an option group with permanent or persistent options, except for Oracle DB instances that have the `Timezone` or `OLS` option (or both).

  A *permanent option* can't be removed from an option group. Option groups with persistent options can't be removed from a DB instance once the option group has been assigned to the DB instance.

  The following table lists permanent and persistent options and their related DB engines.

Option name	Persistent	Permanent	DB engine
TDE	Yes	No	Microsoft SQL Server Enterprise Edition
TDE	Yes	Yes	Oracle Enterprise Edition
Timezone	Yes	Yes	Oracle Enterprise Edition Oracle Standard Edition Oracle Standard Edition One Oracle Standard Edition 2

For Oracle DB instances, you can copy shared DB snapshots that have the `Timezone` or `OLS` option (or both). To do so, specify a target option group that includes these options when you copy the DB snapshot. The OLS option is permanent and persistent only for Oracle DB instances running Oracle version 12.2 or higher. For more information about these options, see Oracle time zone and Oracle Label Security.

- You can't share a snapshot of a Multi-AZ DB cluster.
- You can't share a snapshot when it contains an additional storage volume.

See the following topics for information about sharing public snapshots, sharing encrypted snapshots, and stopping snapshot sharing.

Topics

- Sharing public snapshots for Amazon RDS
- Sharing encrypted snapshots for Amazon RDS
- Stopping snapshot sharing for Amazon RDS

Sharing a snapshot

You can share a DB snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

Console

Using the Amazon RDS console, you can share a manual DB snapshot with up to 20 AWS accounts. You can also use the console to stop sharing a manual snapshot with one or more accounts.

To share a manual DB snapshot by using the Amazon RDS console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Select the manual snapshot that you want to share.
4. For **Actions**, choose **Share snapshot**.
5. Choose one of the following options for **DB snapshot visibility**.

   - If the source is unencrypted, choose **Public** to permit all AWS accounts to restore a DB instance from your manual DB snapshot, or choose **Private** to permit only AWS accounts that you specify to restore a DB instance from your manual DB snapshot.

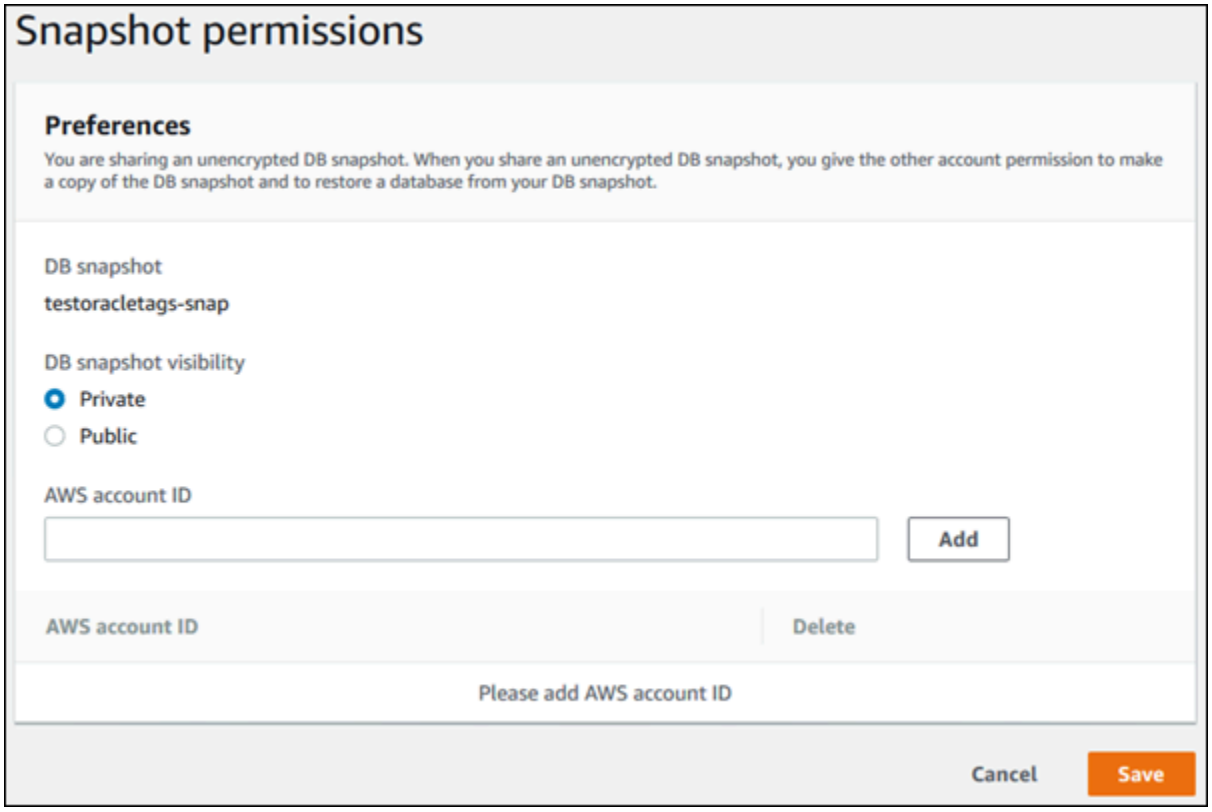
     > **Warning**
     >
     > If you set **DB snapshot visibility** to **Public**, all AWS accounts can restore a DB instance from your manual DB snapshot and have access to your data. Do not share any manual DB snapshots that contain private information as **Public**.
     > For more information, see Sharing public snapshots for Amazon RDS.

   - If the source is encrypted, **DB snapshot visibility** is set as **Private** because encrypted snapshots can't be shared as public.

     > **Note**
     >
     > Snapshots that have been encrypted with the default AWS KMS key can't be shared. For information on how to work around this issue, see Sharing encrypted snapshots for Amazon RDS.

6. For **AWS Account ID**, enter the AWS account identifier for an account that you want to permit to restore a DB instance from your manual snapshot, and then choose **Add**. Repeat to include additional AWS account identifiers, up to 20 AWS accounts.

If you make an error when adding an AWS account identifier to the list of permitted accounts, you can delete it from the list by choosing **Delete** at the right of the incorrect AWS account identifier.



7. After you have added identifiers for all of the AWS accounts that you want to permit to restore the manual snapshot, choose **Save** to save your changes.

AWS CLI

To share a DB snapshot, use the `aws rds modify-db-snapshot-attribute` command. Use the `--values-to-add` parameter to add a list of the IDs for the AWS accounts that are authorized to restore the manual snapshot.

Example of sharing a snapshot with a single account

The following example enables AWS account identifier `123456789012` to restore the DB snapshot named `db7-snapshot`.

For Linux, macOS, or Unix:

```
aws rds modify-db-snapshot-attribute \
```

```
--db-snapshot-identifier db7-snapshot \
--attribute-name restore \
--values-to-add 123456789012
```

For Windows:

```
aws rds modify-db-snapshot-attribute ^
--db-snapshot-identifier db7-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example of sharing a snapshot with multiple accounts

The following example enables two AWS account identifiers, 111122223333 and 444455556666, to restore the DB snapshot named `manual-snapshot1`.

For Linux, macOS, or Unix:

```
aws rds modify-db-snapshot-attribute \
--db-snapshot-identifier manual-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

For Windows:

```
aws rds modify-db-snapshot-attribute ^
--db-snapshot-identifier manual-snapshot1 ^
--attribute-name restore ^
--values-to-add "[\"111122223333\",\"444455556666\"]"
```

> **Note**
>
> When using the Windows command prompt, you must escape double quotes (") in JSON code by prefixing them with a backslash (\).

To list the AWS accounts enabled to restore a snapshot, use the `describe-db-snapshot-attributes` AWS CLI command.

RDS API

You can also share a manual DB snapshot with other AWS accounts by using the Amazon RDS API. To do so, call the `ModifyDBSnapshotAttribute` operation. Specify `restore` for `AttributeName`, and use the `ValuesToAdd` parameter to add a list of the IDs for the AWS accounts that are authorized to restore the manual snapshot.

To make a manual snapshot public and restorable by all AWS accounts, use the value `all`. However, take care not to add the `all` value for any manual snapshots that contain private information that you don't want to be available to all AWS accounts. Also, don't specify `all` for encrypted snapshots, because making such snapshots public isn't supported.

To list all of the AWS accounts permitted to restore a snapshot, use the `DescribeDBSnapshotAttributes` API operation.

Sharing public snapshots for Amazon RDS

You can share an unencrypted manual snapshot as public, which makes the snapshot available to all AWS accounts. Make sure when sharing a snapshot as public that none of your private information is included in the public snapshot.

When a snapshot is shared publicly, it gives all AWS accounts permission both to copy the snapshot and to create DB instances from it.

You aren't billed for the backup storage of public snapshots owned by other accounts. You're billed only for snapshots that you own.

If you copy a public snapshot, you own the copy. You're billed for the backup storage of your snapshot copy. If you create a DB instance from a public snapshot, you're billed for that DB instance. For Amazon RDS pricing information, see the Amazon RDS product page.

You can delete only the public snapshots that you own. To delete a shared or public snapshot, make sure to log into the AWS account that owns the snapshot.

Viewing public snapshots owned by other AWS accounts

You can view public snapshots owned by other accounts in a particular AWS Region on the **Public** tab of the **Snapshots** page in the Amazon RDS console. Your snapshots (those owned by your account) don't appear on this tab.

To view public snapshots

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the **Public** tab.

   The public snapshots appear. You can see which account owns a public snapshot in the **Owner** column.

   > **Note**
   >
   > You might have to modify the page preferences, by selecting the gear icon at the upper right of the **Public snapshots** list, to see this column.

Viewing your own public snapshots

You can use the following AWS CLI command (Unix only) to view the public snapshots owned by your AWS account in a particular AWS Region.

```
aws rds describe-db-snapshots --snapshot-type public --include-public |
 grep account_number
```

The output returned is similar to the following example if you have public snapshots.

```
"DBSnapshotArn": "arn:aws:rds:us-east-1:123456789012:snapshot:mysnapshot1",
"DBSnapshotArn": "arn:aws:rds:us-east-1:123456789012:snapshot:mysnapshot2",
```

> **Note**
>
> You might see duplicate entries for `DBSnapshotIdentifier` or `SourceDBSnapshotIdentifier`.

Sharing public snapshots from deprecated DB engine versions

Restoring or copying public snapshots from deprecated DB engine versions isn't supported.

The RDS for Oracle and RDS for PostgreSQL DB engines support upgrading DB snapshot engine versions directly. You can upgrade your snapshots, then re-share them publicly. For more information, see the following:

- Upgrading an Oracle DB snapshot
- Upgrading a PostgreSQL DB snapshot engine version

For other DB engines, perform the following steps to make your existing unsupported public snapshot available to restore or copy:

1. Mark the snapshot as private.
2. Restore the snapshot.
3. Upgrade the restored DB instance to a supported engine version.
4. Create a new snapshot.
5. Re-share the snapshot publicly.

Sharing encrypted snapshots for Amazon RDS

You can share DB snapshots that have been encrypted "at rest" using the AES-256 encryption algorithm, as described in Encrypting Amazon RDS resources.

The following restrictions apply to sharing encrypted snapshots:

- You can't share encrypted snapshots as public.
- You can't share Oracle or Microsoft SQL Server snapshots that are encrypted using Transparent Data Encryption (TDE).
- You can't share a snapshot that has been encrypted using the default KMS key of the AWS account that shared the snapshot.

  For more information about AWS KMS key management for Amazon RDS, see AWS KMS key management.

To work around the default KMS key issue, perform the following tasks:

1. Create a customer managed key and give access to it.
2. Copy and share the snapshot from the source account.

3. Copy the shared snapshot in the target account.

Create a customer managed key and give access to it

First you create a custom KMS key in the same AWS Region as the encrypted DB snapshot. While creating the customer managed key, you give access to it for another AWS account.

> **Note**
>
> You can also use a KMS key from another AWS account when the key policy grants access to the source and target accounts.

To create a customer managed key and give access to it

1. Sign in to the AWS Management Console from the source AWS account.
2. Open the AWS KMS console at https://console.aws.amazon.com/kms.
3. To change the AWS Region, use the Region selector in the upper-right corner of the page.
4. In the navigation pane, choose **Customer managed keys**.
5. Choose **Create key**.
6. On the **Configure key** page:
    a. For **Key type**, select **Symmetric**.
    b. For **Key usage**, select **Encrypt and decrypt**.
    c. Expand **Advanced options**.
    d. For **Key material origin**, select **KMS**.
    e. For **Regionality**, select **Single-Region key**.
    f. Choose **Next**.
7. On the **Add labels** page:
    a. For **Alias**. enter a display name for your KMS key, for example **`share-snapshot`**.
    b. (Optional) Enter a description for your KMS key.
    c. (Optional) Add tags to your KMS key.
    d. Choose **Next**.
8. On the **Define key administrative permissions** page, choose **Next.**

9. On the **Define key usage permissions** page:

    a. For **Other AWS accounts**, choose **Add another AWS account**.

    b. Enter the ID of the AWS account to which you want to give access.

    You can give access to multiple AWS accounts.

    c. Choose **Next**.

10. Review your KMS key, then choose **Finish**.

Copy and share the snapshot from the source account

Next you copy the source DB snapshot to a new snapshot using the customer managed key. Then you share it with the target AWS account.

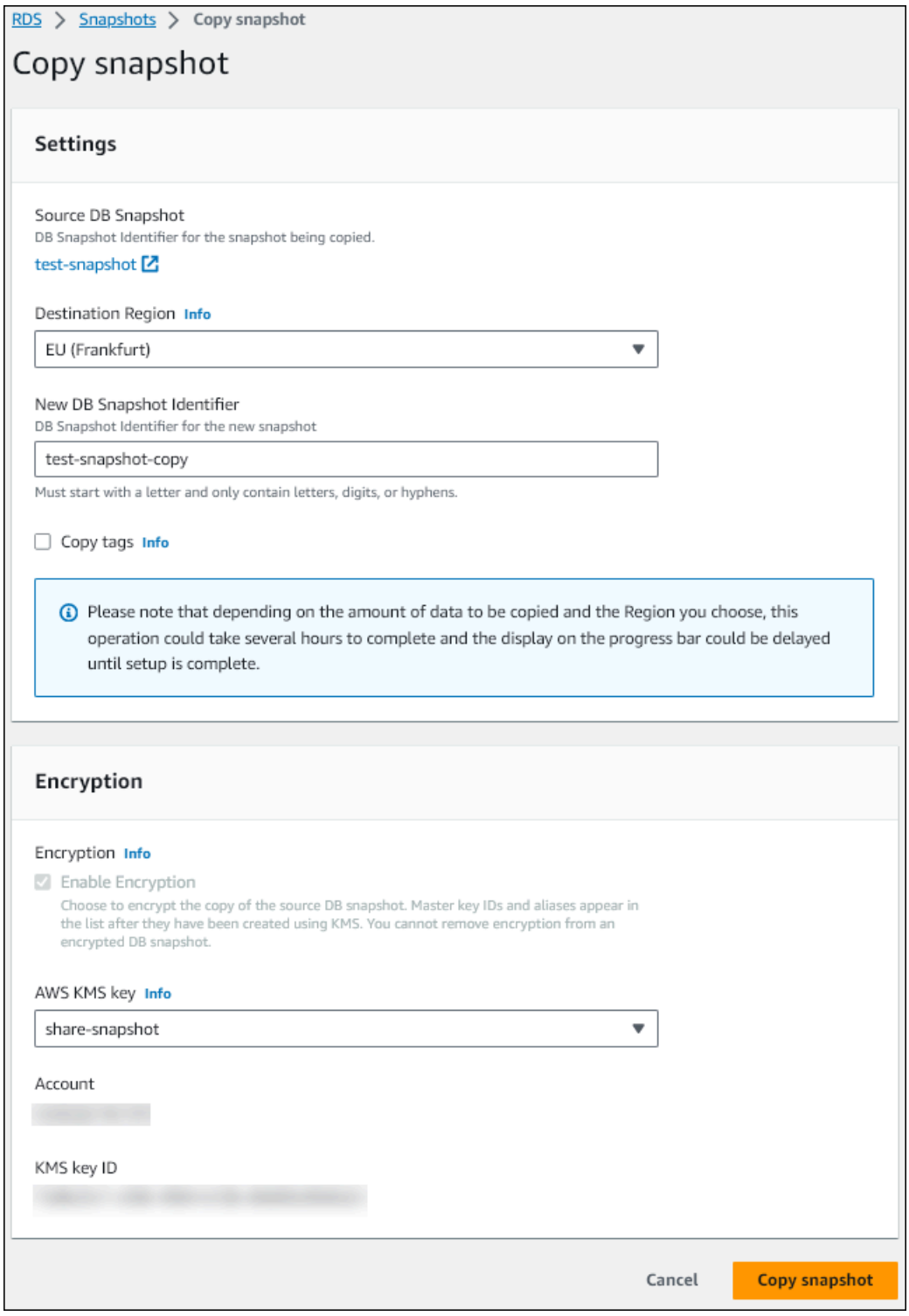
To copy and share the snapshot

1. Sign in to the AWS Management Console from the source AWS account.
2. Open the Amazon RDS console at https://console.aws.amazon.com/rds/
3. In the navigation pane, choose **Snapshots**.
4. Select the DB snapshot you want to copy.
5. For **Actions**, choose **Copy snapshot**.
6. On the **Copy snapshot** page:

    a. For **Destination Region**, choose the AWS Region where you created the customer managed key in the previous procedure.

    b. Enter the name of the DB snapshot copy in **New DB Snapshot Identifier**.

    c. For **AWS KMS key**, choose the customer managed key that you created.



d. Choose **Copy snapshot**.

7. When the snapshot copy is available, select it.
8. For **Actions**, choose **Share snapshot**.
9. On the **Snapshot permissions** page:

a. Enter the **AWS account ID** with which you're sharing the snapshot copy, then choose **Add**.

b. Choose **Save**.

The snapshot is shared.

Copy the shared snapshot in the target account

Now you can copy the shared snapshot in the target AWS account.

To copy the shared snapshot

1. Sign in to the AWS Management Console from the target AWS account.
2. Open the Amazon RDS console at https://console.aws.amazon.com/rds/
3. In the navigation pane, choose **Snapshots**.
4. Choose the **Shared with me** tab.
5. Select the shared snapshot.
6. For **Actions**, choose **Copy snapshot**.
7. Choose your settings for copying the snapshot as in the previous procedure, but use an AWS KMS key that belongs to the target account.

   Choose **Copy snapshot**.

Stopping snapshot sharing for Amazon RDS

To stop sharing a DB snapshot, you remove permission from the target AWS account.

Console

To stop sharing a manual DB snapshot with an AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Select the manual snapshot that you want to stop sharing.
4. Choose **Actions**, and then choose **Share snapshot**.

5. To remove permission for an AWS account, choose **Delete** for the AWS account identifier for that account from the list of authorized accounts.
6. Choose **Save** to save your changes.

CLI

To remove an AWS account identifier from the list, use the `--values-to-remove` parameter.

Example of stopping snapshot sharing

The following example prevents AWS account ID 444455556666 from restoring the snapshot.

For Linux, macOS, or Unix:

```
aws rds modify-db-snapshot-attribute \
--db-snapshot-identifier manual-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

For Windows:

```
aws rds modify-db-snapshot-attribute ^
--db-snapshot-identifier manual-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

RDS API

To remove sharing permission for an AWS account, use the `ModifyDBSnapshotAttribute` operation with `AttributeName` set to `restore` and the `ValuesToRemove` parameter. To mark a manual snapshot as private, remove the value `all` from the values list for the `restore` attribute.

Exporting DB snapshot data to Amazon S3 for Amazon RDS

You can export DB snapshot data to an Amazon S3 bucket. The export process runs in the background and doesn't affect the performance of your active database.

When you export a DB snapshot, Amazon RDS extracts data from the snapshot and stores it in an Amazon S3 bucket. The data is stored in an Apache Parquet format that is compressed and consistent.

You can export all types of DB snapshots—including manual snapshots, automated system snapshots, and snapshots created by the AWS Backup service. By default, all data in the snapshot is exported. However, you can choose to export specific sets of databases, schemas, or tables.

After the data is exported, you can analyze the exported data directly through tools like Amazon Athena or Amazon Redshift Spectrum. For more information on using Athena to read Parquet data, see Parquet SerDe in the *Amazon Athena User Guide*. For more information on using Redshift Spectrum to read Parquet data, see COPY from columnar data formats in the *Amazon Redshift Database Developer Guide*.

> **Warning**
>
> You can't restore exported snapshot data from S3 to a new DB instance or import snapshot data from S3 into an existing DB instance. However, you can process the data using Amazon Athena or Redshift Spectrum for analysis. Additionally, you can use AWS Glue to transform the data and then import it into Amazon RDS using tools like AWS DMS or custom scripts.

For more information about exporting DB snapshots to Amazon S3, see the following topics.

Topics

- Monitoring snapshot exports for Amazon RDS
- Canceling a snapshot export task for Amazon RDS
- Failure messages for Amazon S3 export tasks for Amazon RDS
- Troubleshooting RDS for PostgreSQL permissions errors
- File naming conventions for exports to Amazon S3 for Amazon RDS
- Data conversion when exporting to an Amazon S3 bucket for Amazon RDS

Overview of exporting snapshot data

You use the following process to export DB snapshot data to an Amazon S3 bucket. For more details, see the following sections.

1. Identify the snapshot to export.

   Use an existing automated or manual snapshot, or create a manual snapshot of a DB instance or Multi-AZ DB cluster.

2. Set up access to the Amazon S3 bucket.

   A *bucket* is a container for Amazon S3 objects or files. To provide the information to access a bucket, take the following steps:

   a. Identify the S3 bucket where the snapshot is to be exported to. The S3 bucket must be in the same AWS Region as the snapshot. For more information, see Identifying the Amazon S3 bucket for export.

   b. Create an AWS Identity and Access Management (IAM) role that grants the snapshot export task access to the S3 bucket. For more information, see Providing access to an Amazon S3 bucket using an IAM role.

3. Create a symmetric encryption AWS KMS key for the server-side encryption. The KMS key is used by the snapshot export task to set up AWS KMS server-side encryption when writing the export data to S3.

   The KMS key policy must include both the `kms:CreateGrant` and `kms:DescribeKey` permissions. For more information on using KMS keys in Amazon RDS, see AWS KMS key management.

   If you have a deny statement in your KMS key policy, make sure to explicitly exclude the AWS service principal `export.rds.amazonaws.com`.

   You can use a KMS key within your AWS account, or you can use a cross-account KMS key. For more information, see Using a cross-account AWS KMS key for encrypting Amazon S3 exports.

4. Export the snapshot to Amazon S3 using the console or the `start-export-task` CLI command. For more information, see Exporting a DB snapshot to an Amazon S3 bucket.

5. To access your exported data in the Amazon S3 bucket, see Uploading, downloading, and managing objects in the *Amazon Simple Storage Service User Guide*.

Setting up access to an Amazon S3 bucket

To export DB snapshot data to an Amazon S3 file, you first give the snapshot permission to access the Amazon S3 bucket. You then create an IAM role to allow the Amazon RDS service to write to the Amazon S3 bucket.

Topics

- Identifying the Amazon S3 bucket for export
- Providing access to an Amazon S3 bucket using an IAM role
- Using a cross-account Amazon S3 bucket
- Using a cross-account AWS KMS key for encrypting Amazon S3 exports

Identifying the Amazon S3 bucket for export

Identify the Amazon S3 bucket to export the DB snapshot to. Use an existing S3 bucket or create a new S3 bucket.

> **Note**
>
> The S3 bucket to export to must be in the same AWS Region as the snapshot.

For more information about working with Amazon S3 buckets, see the following in the *Amazon Simple Storage Service User Guide*:

- How do I view the properties for an S3 bucket?
- How do I enable default encryption for an Amazon S3 bucket?
- How do I create an S3 bucket?

Providing access to an Amazon S3 bucket using an IAM role

Before you export DB snapshot data to Amazon S3, give the snapshot export tasks write-access permission to the Amazon S3 bucket.

To grant this permission, create an IAM policy that provides access to the bucket, then create an IAM role and attach the policy to the role. You later assign the IAM role to your snapshot export task.

For information about other Amazon S3 access management tools, see Access control in Amazon S3 in the *Amazon S3 User Guide*.

> **Important**
>
> If you plan to use the AWS Management Console to export your snapshot, you can choose to create the IAM policy and the role automatically when you export the snapshot. For instructions, see Exporting a DB snapshot to an Amazon S3 bucket.

To give DB snapshot tasks access to Amazon S3

1. Create an IAM policy. This policy provides the bucket and object permissions that allow your snapshot export task to access Amazon S3.

   In the policy, include the following required actions to allow the transfer of files from Amazon RDS to an S3 bucket:

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

   In the policy, include the following resources to identify the S3 bucket and objects in the bucket. The following list of resources shows the Amazon Resource Name (ARN) format for accessing Amazon S3.

   - `arn:aws:s3:::amzn-s3-demo-bucket`
   - `arn:aws:s3:::amzn-s3-demo-bucket/*`

   For more information on creating an IAM policy for Amazon RDS, see Creating and using an IAM policy for IAM database access. See also Tutorial: Create and attach your first customer managed policy in the *IAM User Guide*.

   The following AWS CLI command creates an IAM policy named `ExportPolicy` with these options. It grants access to a bucket named `amzn-s3-demo-bucket`.

> **Note**
>
> After you create the policy, note the ARN of the policy. You need the ARN for a subsequent step when you attach the policy to an IAM role.

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}'
```

2. Create an IAM role, so that Amazon RDS can assume this IAM role on your behalf to access your Amazon S3 buckets. For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.

   The following example shows using the AWS CLI command to create a role named `rds-s3-export-role`.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
```

```
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. Attach the IAM policy that you created to the IAM role that you created.

   The following AWS CLI command attaches the policy created earlier to the role named `rds-s3-export-role`. Replace *`your-policy-arn`* with the policy ARN that you noted in an earlier step.

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

Using a cross-account Amazon S3 bucket

You can use Amazon S3 buckets across AWS accounts. To use a cross-account bucket, add a bucket policy to allow access to the IAM role that you're using for the S3 exports. For more information, see Example 2: Bucket owner granting cross-account bucket permissions.

Attach a bucket policy to your bucket, as shown in the following example.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Admin"
            },
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-destination-bucket",
                "arn:aws:s3:::amzn-s3-demo-destination-bucket/*"
            ]
        }
    ]
}
```

Using a cross-account AWS KMS key for encrypting Amazon S3 exports

You can use a cross-account AWS KMS key to encrypt Amazon S3 exports. First, you add a key policy to the local account, then you add IAM policies in the external account. For more information, see Allowing users in other accounts to use a KMS key.

To use a cross-account KMS key

1. Add a key policy to the local account.

   The following example gives `ExampleRole` and `ExampleUser` in the external account 444455556666 permissions in the local account 123456789012.

```
{
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "*"
```

```
}
```

2. Add IAM policies to the external account.

   The following example IAM policy allows the principal to use the KMS key in account 123456789012 for cryptographic operations. To give this permission to `ExampleRole` and `ExampleUser` in account 444455556666, attach the policy to them in that account.

```
{
    "Sid": "Allow use of KMS key in account 123456789012",
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

Exporting a DB snapshot to an Amazon S3 bucket

You can have up to five concurrent DB snapshot export tasks in progress per AWS account.

> **Note**
>
> Exporting RDS snapshots can take a while depending on your database type and size. The export task first restores and scales the entire database before extracting the data to Amazon S3. The task's progress during this phase displays as **Starting**. When the task switches to exporting data to S3, progress displays as **In progress**.
> The time it takes for the export to complete depends on the data stored in the database. For example, tables with well-distributed numeric primary key or index columns export the fastest. Tables that don't contain a column suitable for partitioning and tables with only one index on a string-based column take longer. This longer export time occurs because the export uses a slower single-threaded process.

You can export a DB snapshot to Amazon S3 using the AWS Management Console, the AWS CLI, or the RDS API. To export a DB snapshot to a cross-account Amazon S3 bucket, use the AWS CLI or the RDS API.

If you use a Lambda function to export a snapshot, add the `kms:DescribeKey` action to the Lambda function policy. For more information, see AWS Lambda permissions.

Console

The **Export to Amazon S3** console option appears only for snapshots that can be exported to Amazon S3. A snapshot might not be available for export because of the following reasons:

- The DB engine isn't supported for S3 export.
- The DB engine version isn't supported for S3 export.
- S3 export isn't supported in the AWS Region where the snapshot was created.

To export a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. From the tabs, choose the type of snapshot that you want to export.
4. In the list of snapshots, choose the snapshot that you want to export.
5. For **Actions**, choose **Export to Amazon S3**.

   The **Export to Amazon S3** window appears.
6. For **Export identifier**, enter a name to identify the export task. This value is also used for the name of the file created in the S3 bucket.
7. Choose the data to be exported:

   - Choose **All** to export all data in the snapshot.
   - Choose **Partial** to export specific parts of the snapshot. To identify which parts of the snapshot to export, enter one or more databases, schemas, or tables for **Identifiers**, separated by spaces.

     Use the following format:

```
database[.schema][.table] database2[.schema2][.table2] ... databasen[.schemaN]
[.tablen]
```

For example:

```
mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
 mydatabase2.myschema2.mytable2
```

8. For **S3 bucket**, choose the bucket to export to.

   To assign the exported data to a folder path in the S3 bucket, enter the optional path for **S3 prefix**.

9. For **IAM role**, either choose a role that grants you write access to your chosen S3 bucket, or create a new role.

   - If you created a role by following the steps in Providing access to an Amazon S3 bucket using an IAM role, choose that role.
   - If you didn't create a role that grants you write access to your chosen S3 bucket, then choose **Create a new role** to create the role automatically. Next, enter a name for the role in **IAM role name**.

10. For **AWS KMS key**, enter the ARN for the key to use for encrypting the exported data.
11. Choose **Export to Amazon S3**.

AWS CLI

To export a DB snapshot to Amazon S3 using the AWS CLI, use the start-export-task command with the following required options:

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

In the following examples, the snapshot export task is named *my-snapshot-export*, which exports a snapshot to an S3 bucket named *amzn-s3-demo-bucket*.

Example

For Linux, macOS, or Unix:

```
aws rds start-export-task \
    --export-task-identifier my-snapshot-export \
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
    --s3-bucket-name amzn-s3-demo-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

For Windows:

```
aws rds start-export-task ^
    --export-task-identifier my-snapshot-export ^
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
    --s3-bucket-name amzn-s3-demo-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

Sample output follows.

```
{
    "Status": "STARTING",
    "IamRoleArn": "iam-role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "my-export-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "my-key",
    "ExportTaskIdentifier": "my-snapshot-export",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"
}
```

To provide a folder path in the S3 bucket for the snapshot export, include the `--s3-prefix` option in the start-export-task command.

RDS API

To export a DB snapshot to Amazon S3 using the Amazon RDS API, use the StartExportTask operation with the following required parameters:

- `ExportTaskIdentifier`
- `SourceArn`
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

Region and version availability

Feature availability and support varies across specific versions of each database engine and across AWS Regions. For more information on version and Region availability with exporting snapshots to S3, see Supported Regions and DB engines for exporting snapshots to S3 in Amazon RDS.

Limitations

Exporting DB snapshot data to Amazon S3 has the following limitations:

- You can't run multiple export tasks for the same DB snapshot simultaneously. This applies to both full and partial exports.
- Exporting snapshots from databases that use magnetic storage isn't supported.
- Exports to S3 don't support S3 prefixes containing a colon (:).
- The following characters in the S3 file path are converted to underscores (_) during export:

```
\ ` " (space)
```

- If a database, schema, or table has characters in its name other than the following, partial export isn't supported. However, you can export the entire DB snapshot.
  - Latin letters (A–Z)
  - Digits (0–9)
  - Dollar symbol ($)
  - Underscore (_)

- Spaces ( ) and certain characters aren't supported in database table column names. Tables with the following characters in column names are skipped during export:

  ```
  , ; { } ( ) \n \t = (space)
  ```

- Tables with slashes (/) in their names are skipped during export.
- RDS for PostgreSQL temporary and unlogged tables are skipped during export.
- If the data contains a large object, such as a BLOB or CLOB, that is close to or greater than 500 MB, then the export fails.
- If a table contains a large row that is close to or greater than 2 GB, then the table is skipped during export.
- For partial exports, the `ExportOnly` list has a maximum size of 200 KB.
- We strongly recommend that you use a unique name for each export task. If you don't use a unique task name, you might receive the following error message:

  ExportTaskAlreadyExistsFault: An error occurred (ExportTaskAlreadyExists) when calling the StartExportTask operation: The export task with the ID *xxxxx* already exists.
- You can delete a snapshot while you're exporting its data to S3, but you're still charged for the storage costs for that snapshot until the export task has completed.
- You can't restore exported snapshot data from S3 to a new DB instance or import snapshot data from S3 into an existing DB instance.
- You can have up to five concurrent DB snapshot export tasks in progress per AWS account.
- To export a DB snapshot to a cross-account Amazon S3 bucket, you must use the AWS CLI or the RDS API.
- After Amazon RDS completes an export task, you might have to wait a short time to start another export task from the same DB snapshot.
- You can't export views or materialized views.
- RDS Export to S3 doesn't support tag-based access control for GuardDuty Malware Protection for S3.

Monitoring snapshot exports for Amazon RDS

You can monitor DB snapshot exports using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To monitor DB snapshot exports

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. To view the list of snapshot exports, choose the **Exports in Amazon S3** tab.
4. To view information about a specific snapshot export, choose the export task.

AWS CLI

To monitor DB snapshot exports using the AWS CLI, use the describe-export-tasks command.

The following example shows how to display current information about all of your snapshot exports.

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "amzn-s3-demo-bucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-
groups-test"
        },
{
            "Status": "COMPLETE",
```

```
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "amzn-s3-demo-bucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket edgcuc-export isn't located in the current
 AWS Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "amzn-s3-demo-bucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
}
```

To display information about a specific snapshot export, include the `--export-task-identifier` option with the `describe-export-tasks` command. To filter the output, include the `--Filters` option. For more options, see the describe-export-tasks command.

RDS API

To display information about DB snapshot exports using the Amazon RDS API, use the DescribeExportTasks operation.

To track completion of the export workflow or to initiate another workflow, you can subscribe to Amazon Simple Notification Service topics. For more information on Amazon SNS, see Working with Amazon RDS event notification.

Canceling a snapshot export task for Amazon RDS

You can cancel a DB snapshot export task using the AWS Management Console, the AWS CLI, or the RDS API.

> **Note**
>
> Canceling a snapshot export task doesn't remove any data that was exported to Amazon S3. For information about how to delete the data using the console, see How do I delete objects from an S3 bucket? To delete the data using the CLI, use the delete-object command.

Console

To cancel a snapshot export task

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the **Exports in Amazon S3** tab.
4. Choose the snapshot export task that you want to cancel.
5. Choose **Cancel**.
6. Choose **Cancel export task** on the confirmation page.

AWS CLI

To cancel a snapshot export task using the AWS CLI, use the cancel-export-task command. The command requires the `--export-task-identifier` option.

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "amzn-s3-demo-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

RDS API

To cancel a snapshot export task using the Amazon RDS API, use the CancelExportTask operation with the `ExportTaskIdentifier` parameter.

Failure messages for Amazon S3 export tasks for Amazon RDS

The following table describes the messages that are returned when Amazon S3 export tasks fail.

Failure message	Description
An unknown internal error occurred.	The task has failed because of an unknown error, exception, or failure.
An unknown internal error occurred writing the export task's metadata to the S3 bucket [bucket name].	The task has failed because of an unknown error, exception, or failure.
The RDS export failed to write the export task's metadata because it can't assume the IAM role [role ARN].	The export task assumes your IAM role to validate whether it is allowed to write metadata to your S3 bucket. If the task can't assume your IAM role, it fails.
The RDS export failed to write the export task's metadata to the S3	One or more permissions are missing, so the export task can't access the S3 bucket. This failure message

Failure message	Description
bucket [bucket name] using the IAM role [role ARN] with the KMS key [key ID]. Error code: [error code]	is raised when receiving one of the following error codes: • `AWSSecurityTokenServiceException` with the error code `AccessDenied` • `AmazonS3Exception` with the error code `NoSuchBucket`, `AccessDenied`, `KMS.KMSInvalidStateException`, `403 Forbidden`, or `KMS.DisabledException` These error codes indicate settings are misconfigured for the IAM role, S3 bucket, or KMS key.
The IAM role [role ARN] isn't authorized to call [S3 action] on the S3 bucket [bucket name]. Review your permissions and retry the export.	The IAM policy is misconfigured. Permission for the specific S3 action on the S3 bucket is missing, which causes the export task to fail.
KMS key check failed. Check the credentials on your KMS key and try again.	The KMS key credential check failed.
S3 credential check failed. Check the permissions on your S3 bucket and IAM policy.	The S3 credential check failed.
The S3 bucket [bucket name] isn't valid. Either it isn't located in the current AWS Region or it doesn't exist. Review your S3 bucket name and retry the export.	The S3 bucket is invalid.
The S3 bucket [bucket name] isn't located in the current AWS Region. Review your S3 bucket name and retry the export.	The S3 bucket is in the wrong AWS Region.

Troubleshooting RDS for PostgreSQL permissions errors

When exporting PostgreSQL databases to Amazon S3, you might see a `PERMISSIONS_DO_NOT_EXIST` error stating that certain tables were skipped. This error usually occurs when the superuser, which you specified when creating the database, doesn't have permissions to access those tables.

To fix this error, run the following command:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

For more information on superuser privileges, see Master user account privileges.

File naming conventions for exports to Amazon S3 for Amazon RDS

Exported data for specific tables is stored in the format *base_prefix*/*files*, where the base prefix is the following:

```
export_identifier/database_name/schema_name.table_name/
```

For example:

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

There are two conventions for how files are named.

- Current convention:

  ```
  batch_index/part-partition_index-random_uuid.format-based_extension
  ```

  The batch index is a sequence number that represents a batch of data read from the table. If we can't partition your table into small chunks to be exported in parallel, there will be multiple batch indexes. The same thing happens if your table is partitioned into multiple tables. There will be multiple batch indexes, one for each of the table partitions of your main table.

  If we can partition your table into small chunks to be read in parallel, there will be only the batch index 1 folder.

Inside the batch index folder, there are one or more Parquet files that contain your table's data. The prefix of the Parquet filename is `part-`*`partition_index`*. If your table is partitioned, there will be multiple files starting with the partition index `00000`.

There can be gaps in the partition index sequence. This happens because each partition is obtained from a ranged query in your table. If there is no data in the range of that partition, then that sequence number is skipped.

For example, suppose that the `id` column is the table's primary key, and its minimum and maximum values are `100` and `1000`. When we try to export this table with nine partitions, we read it with parallel queries such as the following:

```
SELECT * FROM table WHERE id <= 100 AND id < 200
SELECT * FROM table WHERE id <= 200 AND id < 300
```

This should generate nine files, from `part-00000-`*`random_uuid`*`.gz.parquet` to `part-00008-`*`random_uuid`*`.gz.parquet`. However, if there are no rows with IDs between `200` and `350`, one of the completed partitions is empty, and no file is created for it. In the previous example, `part-00001-`*`random_uuid`*`.gz.parquet` isn't created.

- Older convention:

```
part-partition_index-random_uuid.format-based_extension
```

This is the same as the current convention, but without the *`batch_index`* prefix, for example:

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

The file naming convention is subject to change. Therefore, when reading target tables, we recommend that you read everything inside the base prefix for the table.

Data conversion when exporting to an Amazon S3 bucket for Amazon RDS

When you export a DB snapshot to an Amazon S3 bucket, Amazon RDS converts data to, exports data in, and stores data in the Parquet format. For more information about Parquet, see the Apache Parquet website.

Parquet stores all data as one of the following primitive types:

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY – A variable-length byte array, also known as binary
- FIXED_LEN_BYTE_ARRAY – A fixed-length byte array used when the values have a constant size

The Parquet data types are few to reduce the complexity of reading and writing the format. Parquet provides logical types for extending primitive types. A *logical type* is implemented as an annotation with the data in a `LogicalType` metadata field. The logical type annotation explains how to interpret the primitive type.

When the `STRING` logical type annotates a `BYTE_ARRAY` type, it indicates that the byte array should be interpreted as a UTF-8 encoded character string. After an export task completes, Amazon RDS notifies you if any string conversion occurred. The underlying data exported is always the same as the data from the source. However, due to the encoding difference in UTF-8, some characters might appear different from the source when read in tools such as Athena.

For more information, see Parquet logical type definitions in the Parquet documentation.

Topics

- MySQL and MariaDB data type mapping to Parquet
- PostgreSQL data type mapping to Parquet

MySQL and MariaDB data type mapping to Parquet

The following table shows the mapping from MySQL and MariaDB data types to Parquet data types when data is converted and exported to Amazon S3.

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
Numeric data types			
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN _BYTE_ARRAY(9)	DECIMAL(20,0)	Parquet supports only signed types, so the mapping requires an additional byte (8 plus 1) to store the BIGINT_UNSIGNED type.
BIT	BYTE_ARRAY		
DECIMAL	INT32	DECIMAL(p,s)	If the source value is less than 2^{31}, it's stored as INT32.
	INT64	DECIMAL(p,s)	If the source value is 2^{31} or greater, but less than 2^{63}, it's stored as INT64.
	FIXED_LEN _BYTE_ARRAY(N)	DECIMAL(p,s)	If the source value is 2^{63} or greater, it's stored as FIXED_LEN _BYTE_ARRAY(N).
	BYTE_ARRAY	STRING	Parquet doesn't support Decimal precision greater

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
			than 38. The Decimal value is converted to a string in a BYTE_ARRAY type and encoded as UTF8.
DOUBLE	DOUBLE		
FLOAT	DOUBLE		
INT	INT32		
INT UNSIGNED	INT64		
MEDIUMINT	INT32		
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL(p,s)	If the source value is less than 2^{31}, it's stored as INT32.
	INT64	DECIMAL(p,s)	If the source value is 2^{31} or greater, but less than 2^{63}, it's stored as INT64.
	FIXED_LEN_ARRAY(N)	DECIMAL(p,s)	If the source value is 2^{63} or greater, it's stored as FIXED_LEN_BYTE_ARRAY(N).

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
	BYTE_ARRAY	STRING	Parquet doesn't support Numeric precision greater than 38. This Numeric value is converted to a string in a BYTE_ARRAY type and encoded as UTF8.
SMALLINT	INT32		
SMALLINT UNSIGNED	INT32		
TINYINT	INT32		
TINYINT UNSIGNED	INT32	INT(16, true)	
String data types			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		
ENUM	BYTE_ARRAY	STRING	
LINESTRING	BYTE_ARRAY		
LONGBLOB	BYTE_ARRAY		
LONGTEXT	BYTE_ARRAY	STRING	
MEDIUMBLOB	BYTE_ARRAY		
MEDIUMTEXT	BYTE_ARRAY	STRING	

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
MULTILINESTRING	BYTE_ARRAY		
SET	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		
VARCHAR	BYTE_ARRAY	STRING	
Date and time data types			
DATE	BYTE_ARRAY	STRING	A date is converted to a string in a BYTE_ARRAY type and encoded as UTF8.
DATETIME	INT64	TIMESTAMP_MICROS	
TIME	BYTE_ARRAY	STRING	A TIME type is converted to a string in a BYTE_ARRAY and encoded as UTF8.
TIMESTAMP	INT64	TIMESTAMP_MICROS	
YEAR	INT32		
Geometric data types			
GEOMETRY	BYTE_ARRAY		

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		
POLYGON	BYTE_ARRAY		
JSON data type			
JSON	BYTE_ARRAY	STRING	

PostgreSQL data type mapping to Parquet

The following table shows the mapping from PostgreSQL data types to Parquet data types when data is converted and exported to Amazon S3.

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
Numeric data types			
BIGINT	INT64		
BIGSERIAL	INT64		
DECIMAL	BYTE_ARRAY	STRING	A DECIMAL type is converted to a string in a BYTE_ARRAY type and encoded as UTF8.

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
			This conversion is to avoid complications due to data precision and data values that are not a number (NaN).
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT(16, true)	
SMALLSERIAL	INT32	INT(16, true)	
String and related data types			
ARRAY	BYTE_ARRAY	STRING	An array is converted to a string and encoded as BINARY (UTF8). This conversion is to avoid complications due to data precision, data values that are not a number (NaN), and time data values.

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
BIT	BYTE_ARRAY	STRING	
BIT VARYING	BYTE_ARRAY	STRING	
BYTEA	BINARY		
CHAR	BYTE_ARRAY	STRING	
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TEXT SEARCH	BYTE_ARRAY	STRING	
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	
Date and time data types			
DATE	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	
TIME	BYTE_ARRAY	STRING	
TIME WITH TIME ZONE	BYTE_ARRAY	STRING	
TIMESTAMP	BYTE_ARRAY	STRING	
TIMESTAMP WITH TIME ZONE	BYTE_ARRAY	STRING	
Geometric data types			

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
BOX	BYTE_ARRAY	STRING	
CIRCLE	BYTE_ARRAY	STRING	
LINE	BYTE_ARRAY	STRING	
LINESEGMENT	BYTE_ARRAY	STRING	
PATH	BYTE_ARRAY	STRING	
POINT	BYTE_ARRAY	STRING	
POLYGON	BYTE_ARRAY	STRING	
JSON data types			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	
Other data types			
BOOLEAN	BOOLEAN		
CIDR	BYTE_ARRAY	STRING	Network data type
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	
INET	BYTE_ARRAY	STRING	Network data type
MACADDR	BYTE_ARRAY	STRING	
OBJECT IDENTIFIER	N/A		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
UUID	BYTE_ARRAY	STRING	

Using AWS Backup to manage automated backups for Amazon RDS

AWS Backup is a fully managed backup service that makes it easy to centralize and automate the backup of data across AWS services in the cloud and on premises. You can manage backups of your Amazon RDS databases in AWS Backup.

> **Note**
>
> Backups managed by AWS Backup are considered manual DB snapshots, but don't count toward the DB snapshot quota for RDS. Backups that were created with AWS Backup have names ending in `awsbackup:backup-job-number`.

For more information about AWS Backup, see the *AWS Backup Developer Guide*.

To view backups managed by AWS Backup

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the **Backup service** tab.

   Your AWS Backup backups are listed under **Backup service snapshots**.

Monitoring metrics in an Amazon RDS instance

In the following sections, you can find an overview of Amazon RDS monitoring and an explanation about how to access metrics. To learn how to monitor events, logs, and database activity streams, see Monitoring events, logs, and streams in an Amazon RDS DB instance.

Topics

- Monitoring plan
- Performance baseline
- Performance guidelines
- Monitoring tools for Amazon RDS
- Viewing instance status
- Recommendations from Amazon RDS
- Viewing metrics in the Amazon RDS console
- Viewing combined metrics with the Performance Insights dashboard
- Monitoring Amazon RDS metrics with Amazon CloudWatch
- Monitoring Amazon RDS databases with CloudWatch Database Insights
- Monitoring DB load with Performance Insights on Amazon RDS
- Analyzing performance anomalies with Amazon DevOps Guru for Amazon RDS
- Monitoring OS metrics with Enhanced Monitoring
- Metrics reference for Amazon RDS

Monitoring plan

Before you start monitoring Amazon RDS, create a monitoring plan. This plan should answer the following questions:

- What are your monitoring goals?
- Which resources will you monitor?
- How often will you monitor these resources?
- Which monitoring tools will you use?
- Who will perform the monitoring tasks?

- Whom should be notified when something goes wrong?

Performance baseline

To achieve your monitoring goals, you need to establish a baseline. To do this, measure performance under different load conditions at various times in your Amazon RDS environment. You can monitor metrics such as the following:

- Network throughput
- Client connections
- I/O for read, write, or metadata operations
- Burst credit balances for your DB instances

We recommend that you store historical performance data for Amazon RDS. Using the stored data, you can compare current performance against past trends. You can also distinguish normal performance patterns from anomalies, and devise techniques to address issues.

Performance guidelines

In general, acceptable values for performance metrics depend on what your application is doing relative to your baseline. Investigate consistent or trending variances from your baseline. The following metrics are often the source of performance issues:

- **High CPU or RAM consumption** – High values for CPU or RAM consumption might be appropriate, if they're in keeping with your goals for your application (like throughput or concurrency) and are expected.
- **Disk space consumption** – Investigate disk space consumption if space used is consistently at or above 85 percent of the total disk space. See if it is possible to delete data from the instance or archive data to a different system to free up space.
- **Network traffic** – For network traffic, talk with your system administrator to understand what expected throughput is for your domain network and internet connection. Investigate network traffic if throughput is consistently lower than expected.
- **Database connections** – If you see high numbers of user connections and also decreases in instance performance and response time, consider constraining database connections. The best number of user connections for your DB instance varies based on your instance class and

the complexity of the operations being performed. To determine the number of database connections, associate your DB instance with a parameter group where the `User Connections` parameter is set to a value other than 0 (unlimited). You can either use an existing parameter group or create a new one. For more information, see Parameter groups for Amazon RDS.

- **IOPS metrics** – The expected values for IOPS metrics depend on disk specification and server configuration, so use your baseline to know what is typical. Investigate if values are consistently different than your baseline. For best IOPS performance, make sure that your typical working set fits into memory to minimize read and write operations.

When performance falls outside your established baseline, you might need to make changes to optimize your database availability for your workload. For example, you might need to change the instance class of your DB instance. Or you might need to change the number of DB instances and read replicas that are available for clients.

Monitoring tools for Amazon RDS

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon RDS and your other AWS solutions. AWS provides various monitoring tools to watch Amazon RDS, report when something is wrong, and take automatic actions when appropriate.

Topics

- Automated monitoring tools
- Manual monitoring tools

Automated monitoring tools

We recommend that you automate monitoring tasks as much as possible.

Topics

- Amazon RDS instance status and recommendations
- Amazon CloudWatch metrics for Amazon RDS
- Amazon RDS Performance Insights and operating-system monitoring
- Integrated services

Amazon RDS instance status and recommendations

You can use the following automated tools to watch Amazon RDS and report when something is wrong:

- **Amazon RDS instance status** — View details about the current status of your instance by using the Amazon RDS console, the AWS CLI, or the RDS API.
- **Amazon RDS recommendations** — Respond to automated recommendations for database resources, such as DB instances, read replicas, and DB parameter groups. For more information, see Recommendations from Amazon RDS.

Amazon CloudWatch metrics for Amazon RDS

Amazon RDS integrates with Amazon CloudWatch for additional monitoring capabilities.

- **Amazon CloudWatch** – This service monitors your AWS resources and the applications you run on AWS in real time. You can use the following Amazon CloudWatch features with Amazon RDS:
  - **Amazon CloudWatch metrics** – Amazon RDS automatically sends metrics to CloudWatch every minute for each active database. You don't get additional charges for Amazon RDS metrics in CloudWatch. For more information, see Monitoring Amazon RDS metrics with Amazon CloudWatch.
  - **Amazon CloudWatch alarms** – You can watch a single Amazon RDS metric over a specific time period. You can then perform one or more actions based on the value of the metric relative to a threshold that you set. For more information, see Monitoring Amazon RDS metrics with Amazon CloudWatch.

Amazon RDS Performance Insights and operating-system monitoring

You can use the following automated tools to monitor Amazon RDS performance:

- **Amazon RDS Performance Insights** – Assess the load on your database, and determine when and where to take action. For more information, see Monitoring DB load with Performance Insights on Amazon RDS.
- **Amazon RDS Enhanced Monitoring** – Look at metrics in real time for the operating system. For more information, see Monitoring OS metrics with Enhanced Monitoring.

Integrated services

The following AWS services are integrated with Amazon RDS:

- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. For more information, see Monitoring Amazon RDS events.
- *Amazon CloudWatch Logs* lets you monitor, store, and access your log files from Amazon RDS instances, CloudTrail, and other sources. For more information, see Monitoring Amazon RDS log files.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. For more information, see Monitoring Amazon RDS API calls in AWS CloudTrail.

- *Database Activity Streams* is an Amazon RDS feature that provides a near-real-time stream of the activity in your Oracle DB instance. For more information, see Monitoring Amazon RDS with Database Activity Streams.

Manual monitoring tools

You need to manually monitor those items that the CloudWatch alarms don't cover. The Amazon RDS, CloudWatch, AWS Trusted Advisor and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your DB instance.

- From the Amazon RDS console, you can monitor the following items for your resources:
  - The number of connections to a DB instance
  - The amount of read and write operations to a DB instance
  - The amount of storage that a DB instance is currently using
  - The amount of memory and CPU being used for a DB instance
  - The amount of network traffic to and from a DB instance
- From the Trusted Advisor dashboard, you can review the following cost optimization, security, fault tolerance, and performance improvement checks:
  - Amazon RDS Idle DB Instances
  - Amazon RDS Security Group Access Risk
  - Amazon RDS Backups
  - Amazon RDS Multi-AZ

  For more information on these checks, see Trusted Advisor best practices (checks).
- CloudWatch home page shows:
  - Current alarms and status
  - Graphs of alarms and resources
  - Service health status

  In addition, you can use CloudWatch to do the following:
  - Create customized dashboards to monitor the services that you care about.
  - Graph metric data to troubleshoot issues and discover trends.
  - Search and browse all your AWS resource metrics.

- Create and edit alarms to be notified of problems.

Viewing instance status

Using the Amazon RDS console, you can quickly access the status of your DB instance.

Topics

- Viewing Amazon RDSDB instance status

Viewing Amazon RDSDB instance status

The status of a DB instance indicates its current operational state. You can use the following procedures to view the DB instance status in the Amazon RDS console, the AWS CLI command, or the API operation.

> **Note**
>
> Amazon RDS also uses another status called *maintenance status*, which is shown in the **Maintenance** column of the Amazon RDS console. This value indicates the status of any maintenance patches that need to be applied to a DB instance. Maintenance status is independent of DB instance status. For more information about maintenance status, see Applying updates to a DB instance .

Find the possible status values for DB instances in the following table. This table also shows whether you will be billed for the DB instance and storage, billed only for storage, or not billed. For all DB instance statuses, you are always billed for backup usage.

DB instance status	Billed	Description
available	Billed	The DB instance is available for modifications.
backing-up	Billed	The DB instance is currently being backed up.
configuring-enhanced-monitoring	Billed	Enhanced Monitoring is being enabled or disabled for this DB instance.
configuring-iam-database-auth	Billed	AWS Identity and Access Management (IAM) database authentication is being enabled or disabled for this DB instance.
configuring-log-exports	Billed	Publishing log files to Amazon CloudWatch Logs is being enabled or disabled for this DB instance.
converting-to-vpc	Billed	The DB instance is being converted from a DB instance that is not in an Amazon Virtual Private Cloud (Amazon VPC) to a DB instance that is in an Amazon VPC.

DB instance status	Billed	Description
creating	Not billed (non-PITR) Billed (PITR only)	The DB instance is being created. The DB instance is inaccessible while it is being created. If you restore a database during point-in-time recovery (PITR), you're billed while the database is in the **creating** state. This is the only scenario in which the **creating** state incurs charges.
delete-precheck	Not billed	Amazon RDS is validating that read replicas are safe to delete.
deleting	Not billed	The DB instance is being deleted.
failed	Not billed	The DB instance has failed and Amazon RDS can't recover it. Perform a point-in-time restore to the latest restorable time of the DB instance to recover the data.
inaccessible-encryption-credentials	Not billed	The AWS KMS key used to encrypt or decrypt the DB instance can't be accessed or recovered.
inaccessible-encryption-credentials-recoverable	Billed for storage	The KMS key used to encrypt or decrypt the DB instance can't be accessed. However, if the KMS key is active, restarting the DB instance can recover it. For more information, see Encrypting a DB instance.
incompatible-create	Not billed	Amazon RDS is attempting to create a DB instance but can't do so because resources are incompatible with your DB instance. This status can occur if, for example, the instance profile for your DB instance doesn't have the correct permissions.

DB instance status	Billed	Description
incompatible-network	Not billed	Amazon RDS is attempting to perform a recovery action on a DB instance but can't do so because the VPC is in a state that prevents the action from being completed. This status can occur if, for example, all available IP addresses in a subnet are in use and Amazon RDS can't get an IP address for the DB instance.
incompatible-option-group	Billed	Amazon RDS attempted to apply an option group change but can't do so, and Amazon RDS can't roll back to the previous option group state. For more information, check the **Recent Events** list for the DB instance. This status can occur if, for example, the option group contains an option such as TDE and the DB instance doesn't contain encrypted i nformation.
incompatible-parameters	Billed	Amazon RDS can't start the DB instance because the parameters specified in the DB instance's DB parameter group aren't compatible with the DB instance. Revert the parameter changes or make them compatible with the DB instance to regain access to your DB instance. For more information about the incompatible parameters, check the **Recent Events** list for the DB instance.
incompatible-restore	Not billed	Amazon RDS can't do a point-in-time restore. Common causes for this status include using temp tables, using MyISAM tables with MySQL, or using Aria tables with MariaDB.

DB instance status	Billed	Description
insufficient-capacity	Not billed	Amazon RDS can't create your instance because sufficient capacity isn't currently available. To create your DB instance in the same AZ with the same instance type, delete your DB instance, wait a few hours, and try to create again. Alternatively, create a new instance using a different instance class or AZ.
maintenance	Billed	Amazon RDS is applying a maintenance update to the DB instance. This status is used for instance-level maintenance that RDS schedules well in advance.
modifying	Billed	The DB instance is being modified because of a customer request to modify the DB instance.
moving-to-vpc	Billed	The DB instance is being moved to a new Amazon Virtual Private Cloud (Amazon VPC).
rebooting	Billed	The DB instance is being rebooted because of a customer request or an Amazon RDS process that requires the rebooting of the DB instance.
resetting-master-credentials	Billed	The master credentials for the DB instance are being reset because of a customer request to reset them.
renaming	Billed	The DB instance is being renamed because of a customer request to rename it.
restore-error	Billed	The DB instance encountered an error attempting to restore to a point-in-time or from a snapshot.
starting	Billed for storage	The DB instance is starting.

DB instance status	Billed	Description
stopped	Billed for storage	The DB instance is stopped.
stopping	Billed for storage	The DB instance is being stopped.
storage-config-upgrade	Billed	The storage file system configuration of the DB instance is being upgraded. This status only applies to green databases within a blue/green deployment, or to DB instance read replicas.
storage-full	Billed	The DB instance has reached its storage capacity allocation. This is a critical status, and we recommend that you fix this issue immediately. To do so, scale up your storage by modifying the DB instance. To avoid this situation, set Amazon CloudWatch alarms to warn you when storage space is getting low.
storage-initialization	Billed	The DB instance is loading data blocks from Amazon S3 to optimize volume performance after being restored from a snapshot. It remains available for operations, but performance mights not be at its fullest until initialization completes.
storage-optimization	Billed	Amazon RDS is optimizing the storage of your DB instance. The storage optimization process is usually short, but can sometimes take up to and even beyond 24 hours. During storage optimization, the DB instance remains available. Storage optimization is a background process that doesn't affect the instance's availability.

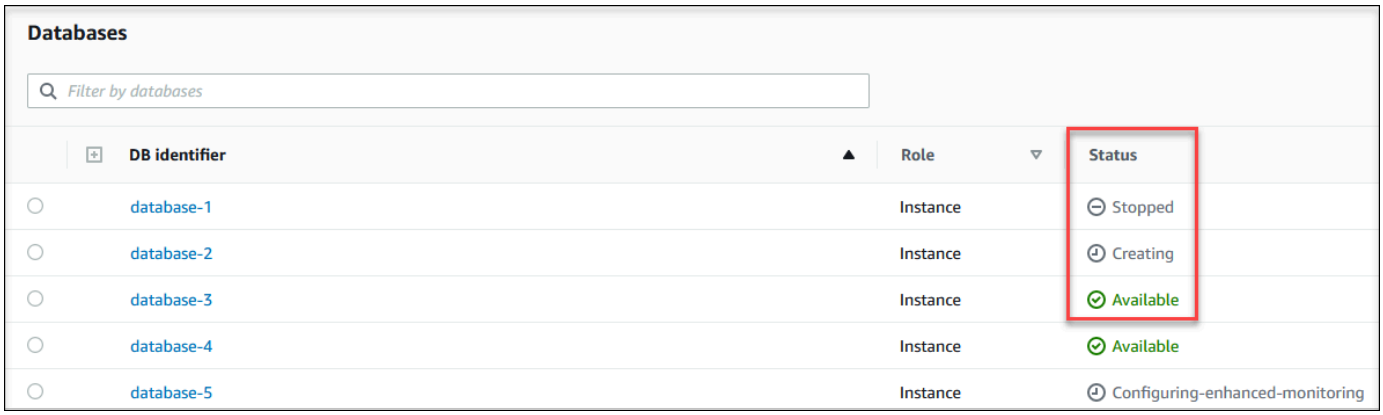
DB instance status	Billed	Description
upgrading	Billed	The database engine or operating system version is being upgraded.
upgrade_failed	Not billed	The DB instance has failed to upgrade to a supported version. Aurora creates a final snapshot with the prefix `rds-final` .

Console

To view the status of a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.

   The **Databases page** appears with the list of DB instances. For each DB instance , the status value is displayed.



CLI

To view DB instance and its status information by using the AWS CLI, use the describe-db-instances command. For example, the following AWS CLI command lists all the DB instances information .

```
aws rds describe-db-instances
```

To view a specific DB instance and its status, call the describe-db-instances command with the following option:

- `DBInstanceIdentifier` – The name of the DB instance.

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

To view just the status of all the DB instances, use the following query in AWS CLI.

```
aws rds describe-db-instances --query 'DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]' --output table
```

API

To view the status of the DB instance using the Amazon RDS API, call the DescribeDBInstances operation.

Recommendations from Amazon RDS

Amazon RDS provides automated recommendations for database resources, such as DB instances, read replicas, and DB parameter groups. These recommendations provide best practice guidance by analyzing DB instance configuration, usage, and performance data.

Amazon RDS Performance Insights monitors specific metrics and automatically creates thresholds by analyzing what levels are considered potentially problematic for a specified resource. When new metric values cross a predefined threshold over a given period of time, Performance Insights generates a proactive recommendation. This recommendation helps to prevent future database performance impact. For example, the "Idle In Transaction" recommendation is generated for RDS for PostgreSQL instances when the sessions connected to the database are not performing active work, but can keep database resources blocked. To receive proactive recommendations, you must turn on Performance Insights with a paid tier retention period. For information about turning on Performance Insights, see Turning Performance Insights on and off for Amazon RDS. For information about pricing and data retention for Performance Insights see Pricing and data retention for Performance Insights.

DevOps Guru for RDS monitors certain metrics to detect when the metric's behavior becomes highly unusual or anomalous. These anomalies are reported as reactive insights with recommendations. For example, DevOps Guru for RDS might recommend you to consider increasing CPU capacity or investigate wait events that are contributing to DB load. DevOps Guru for RDS also provides threshold based proactive recommendations. For these recommendations, you must turn on DevOps Guru for RDS. For information about turning on DevOps Guru for RDS, see Turning on DevOps Guru and specifying resource coverage.

Recommendations will be in any of the following status: active, dismissed, pending, or resolved. Resolved recommendations are available for 365 days.

You can view or dismiss the recommendations. You can apply a configuration based active recommendation immediately, schedule it in the next maintenance window, or dismiss it. For threshold based proactive and machine learning based reactive recommendations, you need to review the suggested cause of the issue and then perform the recommended actions to fix the issue.

Recommendations are supported in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)

- US West (N. California)
- US West (Oregon)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- South America (São Paulo)

Learn to view, apply, dismiss, and modify recommendations from Amazon RDS in the following sections.

Topics

- Viewing Amazon RDS recommendations
- Applying Amazon RDS recommendations
- Dismissing Amazon RDS recommendations
- Modifying dismissed Amazon RDS recommendations to active recommendations
- Recommendations from Amazon RDS reference

Viewing Amazon RDS recommendations

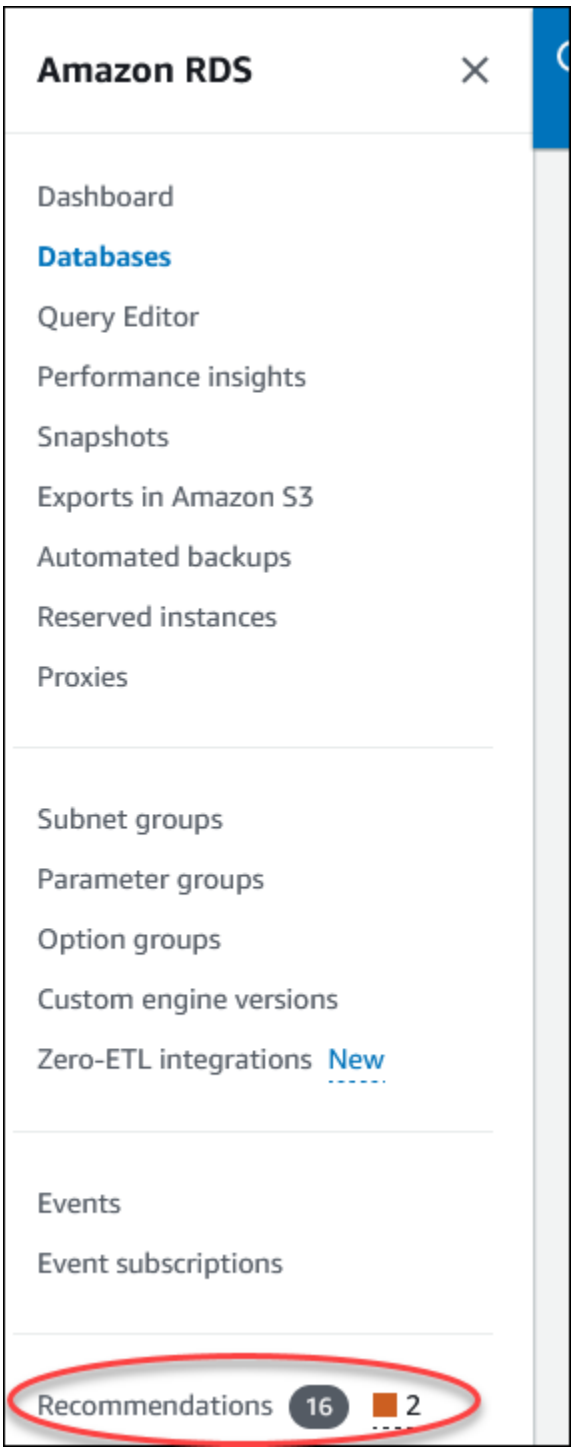
Using the Amazon RDS console, you can view Amazon RDS recommendations for your database resources.

Console

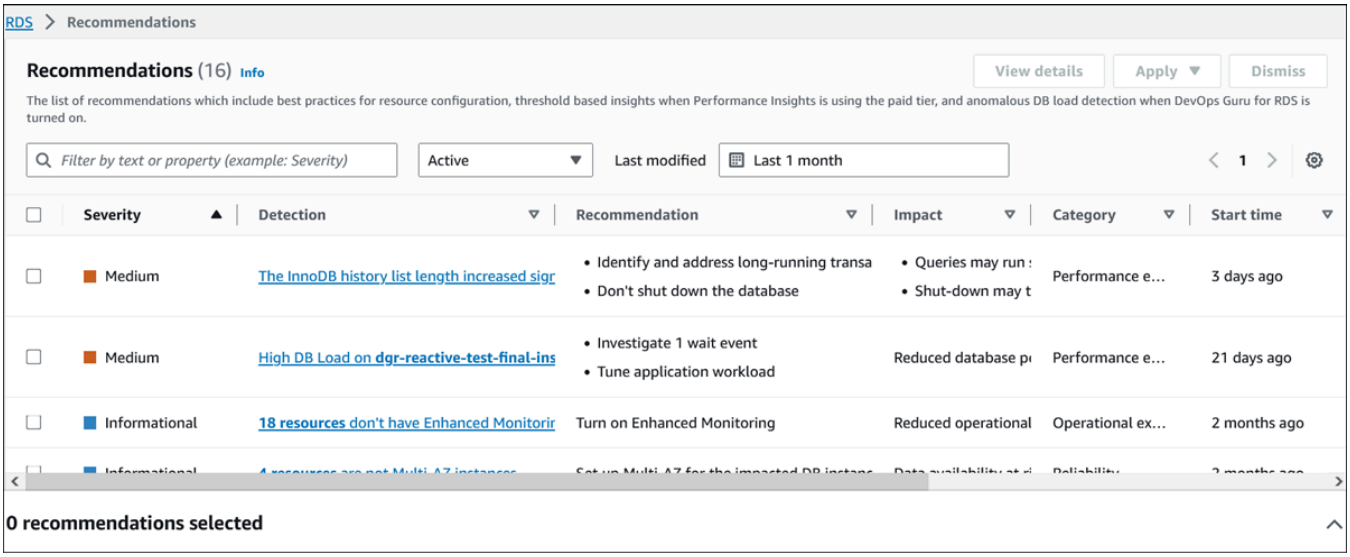
To view the Amazon RDS recommendations

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, do any of the following:

   - Choose **Recommendations**. The number of active recommendations for your resources and the number of recommendations with the highest severity generated in the last month are available next to **Recommendations**. To find the number of active recommendations for each severity, choose the number that shows the highest severity.

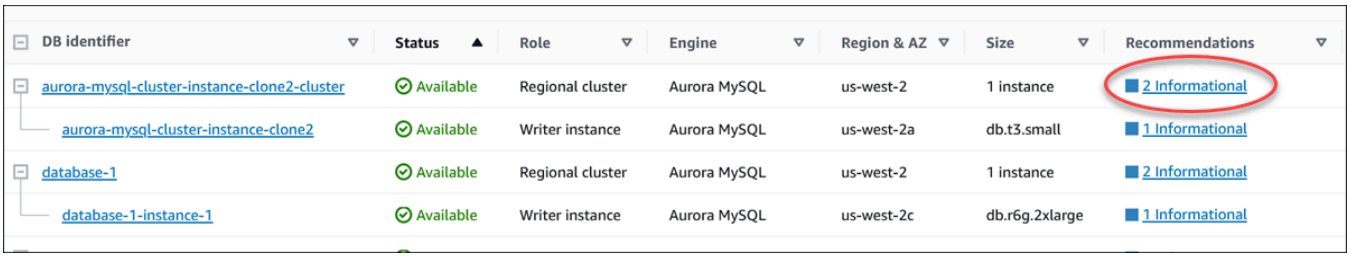


By default, the **Recommendations** page displays a list of new recommendations in the last month. Amazon RDS gives recommendations for all the resources in your account and sorts the recommendations by their severity.

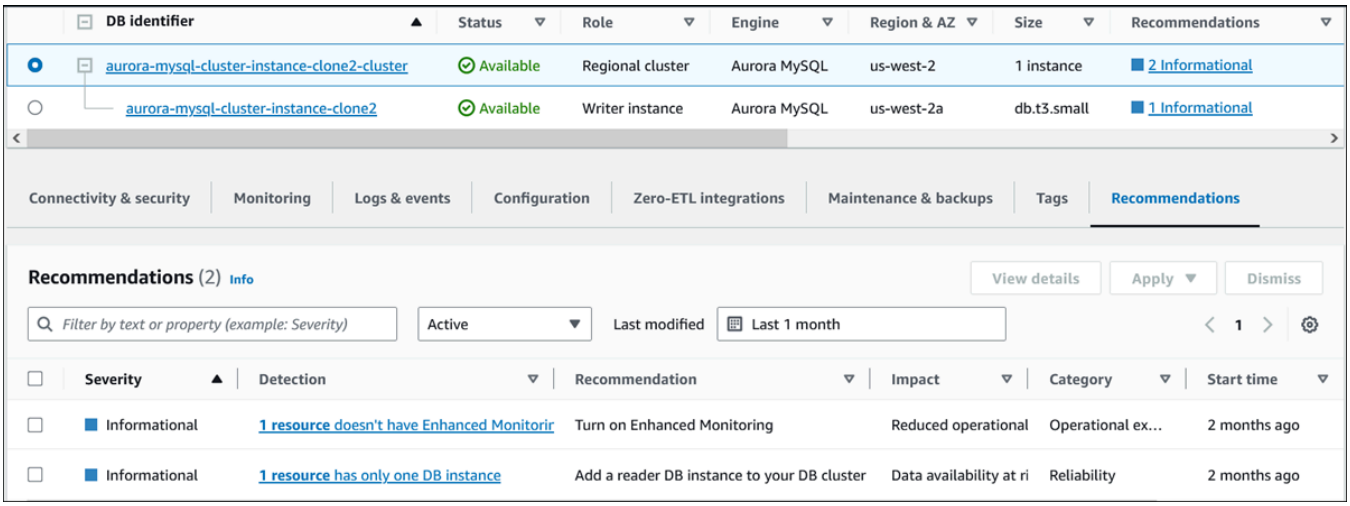


You can choose a recommendation to view a section at the bottom of the page which contains the affected resources and details of how the recommendation will be applied.

- In the **Databases** page, choose **Recommendations** for a resource.



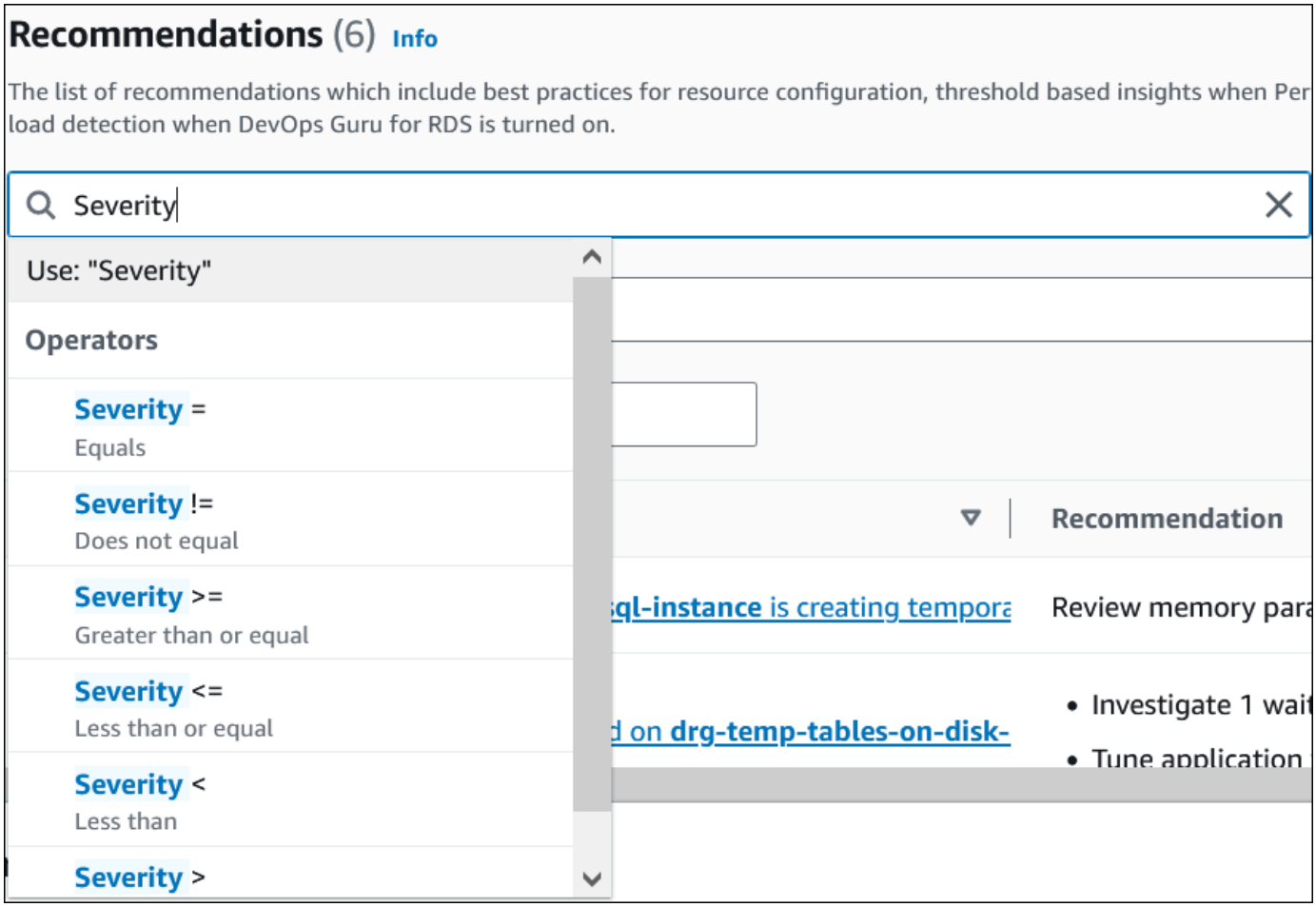
The **Recommendations** tab displays the recommendations and its details for the selected resource.



The following details are available for the recommendations:

- **Severity** – The implication level of the issue. The severity levels are **High**, **Medium**, **Low**, and **Informational**.
- **Detection** – The number of affected resources and a short description of the issue. Choose this link to view the recommendation and the analysis details.
- **Recommendation** – A short description of the recommended action to apply.
- **Impact** – A short description of the possible impact when the recommendation isn't applied.
- **Category** – The type of recommendation. The categories are **Performance efficiency**, **Security**, **Reliability**, **Cost optimization**, **Operational excellence**, and **Sustainability**.
- **Status** – The current status of the recommendation. The possible statuses are **All**, **Active**, **Dismissed**, **Resolved**, and **Pending**.
- **Start time** – The time when the issue began. For example, **18 hours ago**.
- **Last modified** – The time when the recommendation was last updated by the system because of a change in the **Severity**, or the time you responded to the recommendation. For example, **10 hours ago**.
- **End time** – The time when the issue ended. The time won't display for any continuing issues.
- **Resource identifier** – The name of one or more resources.

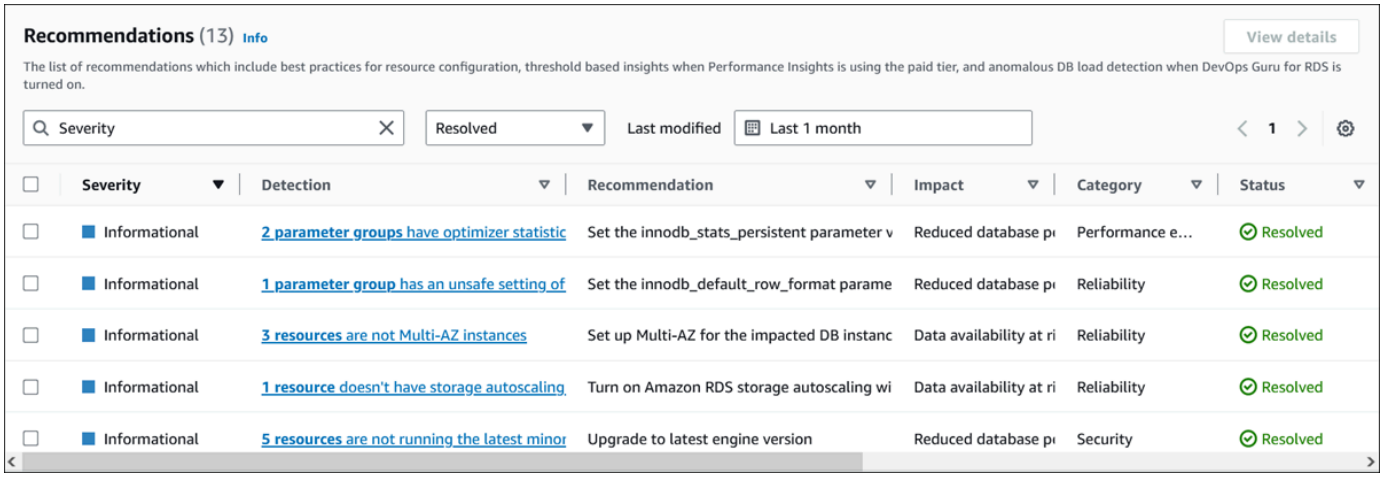
3. (Optional) Choose **Severity** or **Category** operators in the field to filter the list of recommendations.



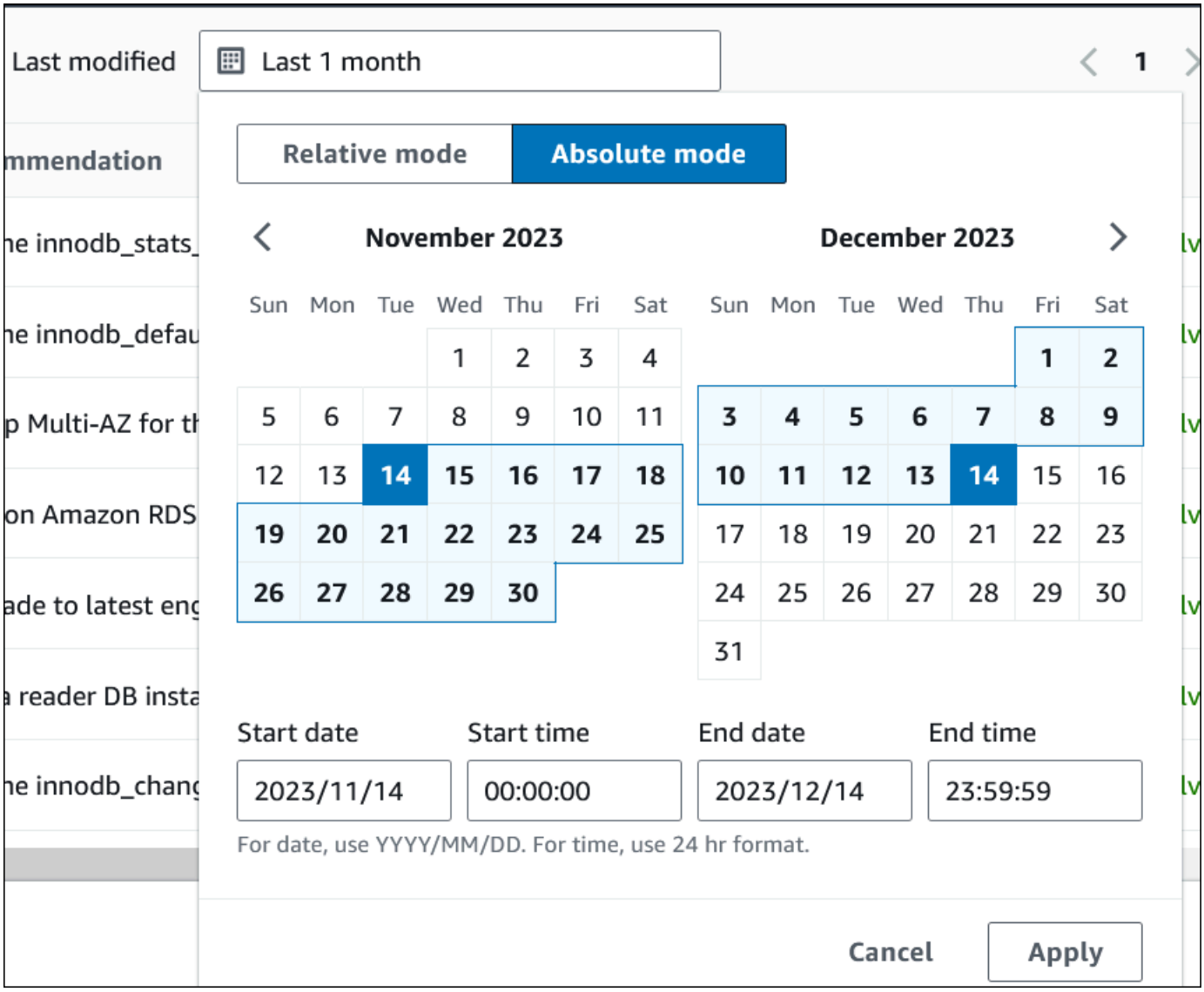
The recommendations for the selected operation appear.

4. (Optional) Choose any of the following recommendation status:

   - **Active** (default) – Shows the current recommendations that you can apply, schedule it for the next maintenance window, or dismiss.
   - **All** – Shows all the recommendations with the current status.
   - **Dismissed** – Shows the dismissed recommendations.
   - **Resolved** – Shows the recommendations that are resolved.
   - **Pending** – Shows the recommendations whose recommended actions are in progress or scheduled for the next maintenance window.



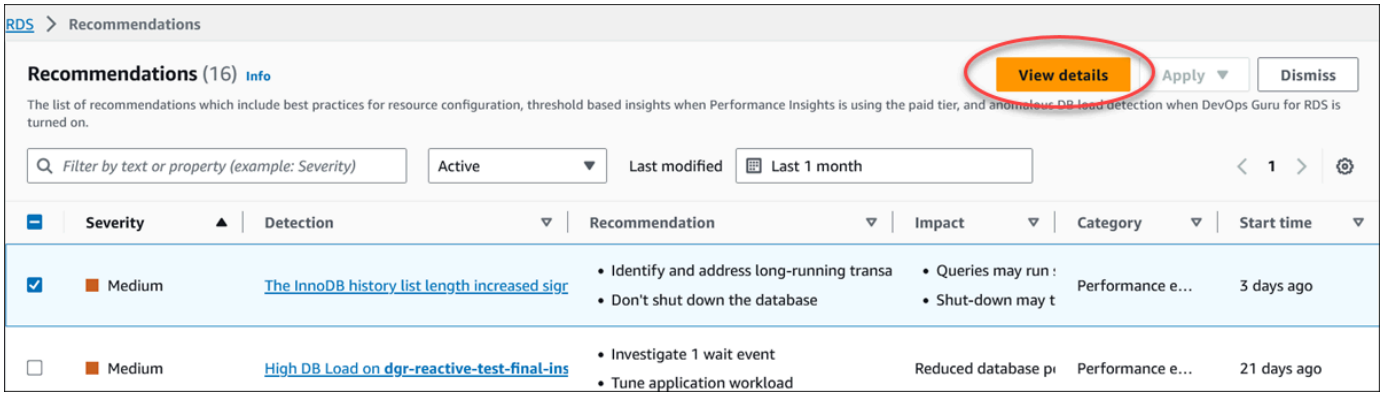
5. (Optional) Choose **Relative mode** or **Absolute mode** in **Last modified** to modify the time period. The **Recommendations** page displays the recommendations generated in the time period. The default time period is the last month. In the **Absolute mode**, you can choose the time period, or enter the time in **Start date** and **End date** fields.



The recommendations for the set time period display.

Note that you can see all recommendations for resources in your account by setting the range to **All**.

6. (Optional) Choose **Preferences** in the right to customize the details to display. You can choose a page size, wrap the lines of the text, and allow or hide the columns.
7. (Optional) Choose a recommendation and then choose **View details**.



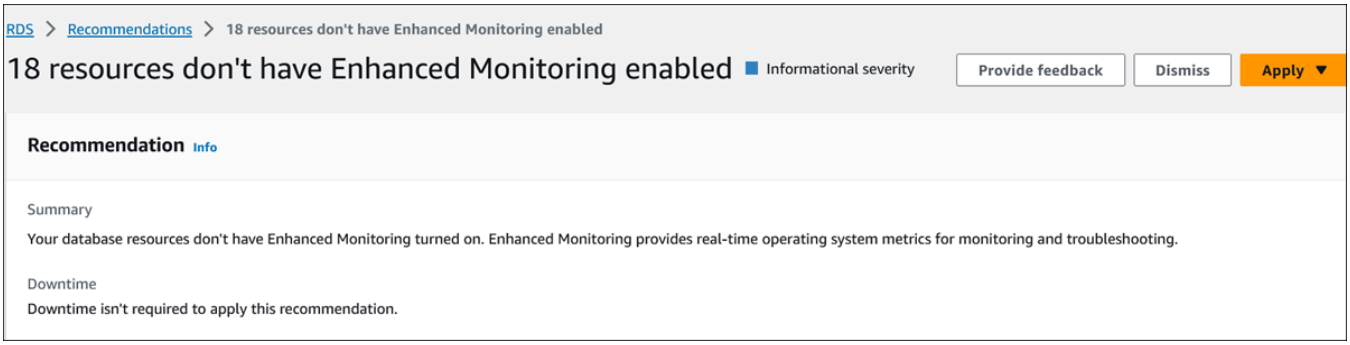
The recommendation details page appears. The title provides the total count of the resources with the issue detected and the severity.

For information about the components on the details page for an anomaly based reactive recommendation, see Viewing reactive anomalies in the *Amazon DevOps Guru User Guide*.

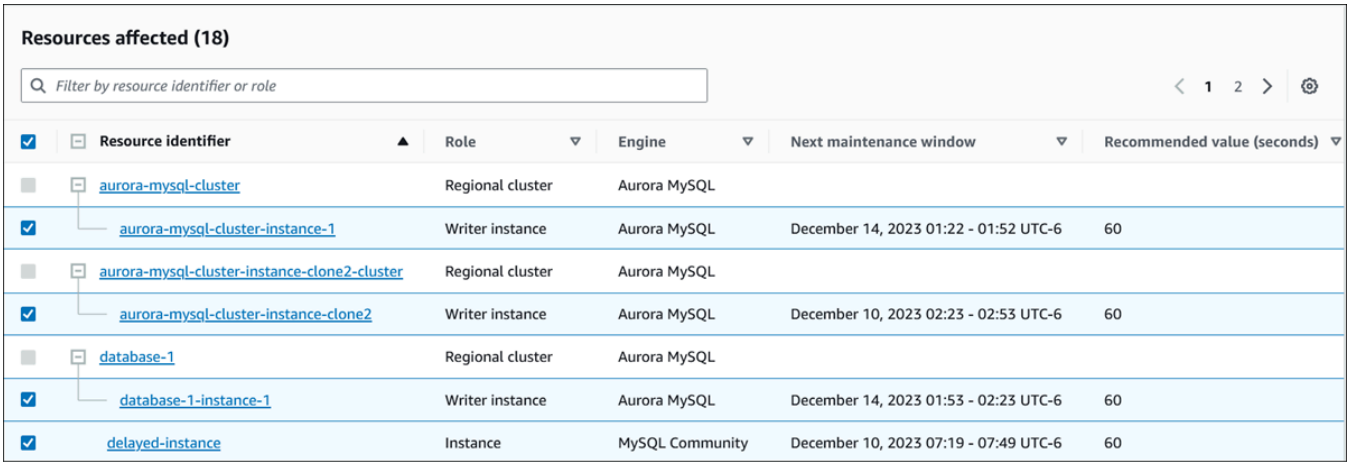
For information about the components on the details page for a threshold based proactive recommendation, see Viewing Performance Insights proactive recommendations.

The other automated recommendations display the following components on the recommendation details page:

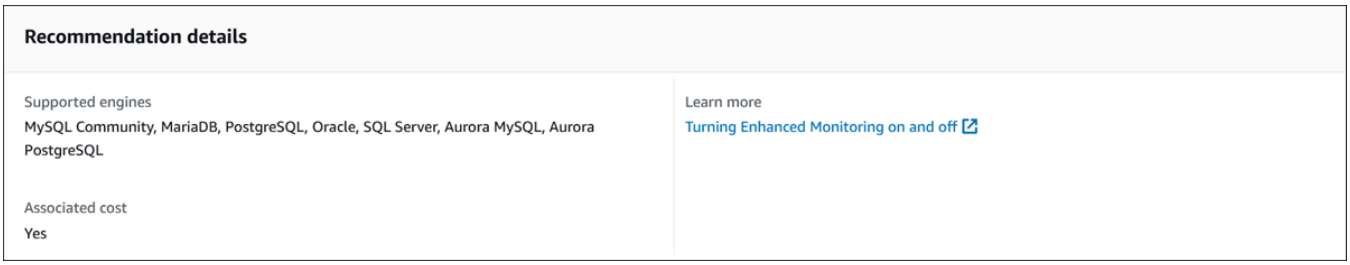
- **Recommendation** – A summary of the recommendation and whether downtime is required to apply the recommendation.



- **Resources affected** – Details of the affected resources.



- **Recommendation details** – Supported engine information, any required associated cost to apply the recommendation, and documentation link to learn more.



CLI

To view Amazon RDS recommendations of the DB instances, use the following command in AWS CLI.

```
aws rds describe-db-recommendations
```

RDS API

To view Amazon RDS recommendations using the Amazon RDS API, use the DescribeDBRecommendations operation.

Applying Amazon RDS recommendations

To apply Amazon RDS recommendations using the Amazon RDS console, select a configuration based recommendation or an affected resource in the details page. Then, choose to apply the recommendation immediately or schedule it for the next maintenance window. The resource might need to restart for the change to take effect. For a few DB parameter group recommendations, you might need to restart the resources.

The threshold based proactive or anomaly based reactive recommendations won't have the apply option and might need additional review.

Console

To apply a configuration based recommendation

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, perform any of the following:

   - Choose **Recommendations**.

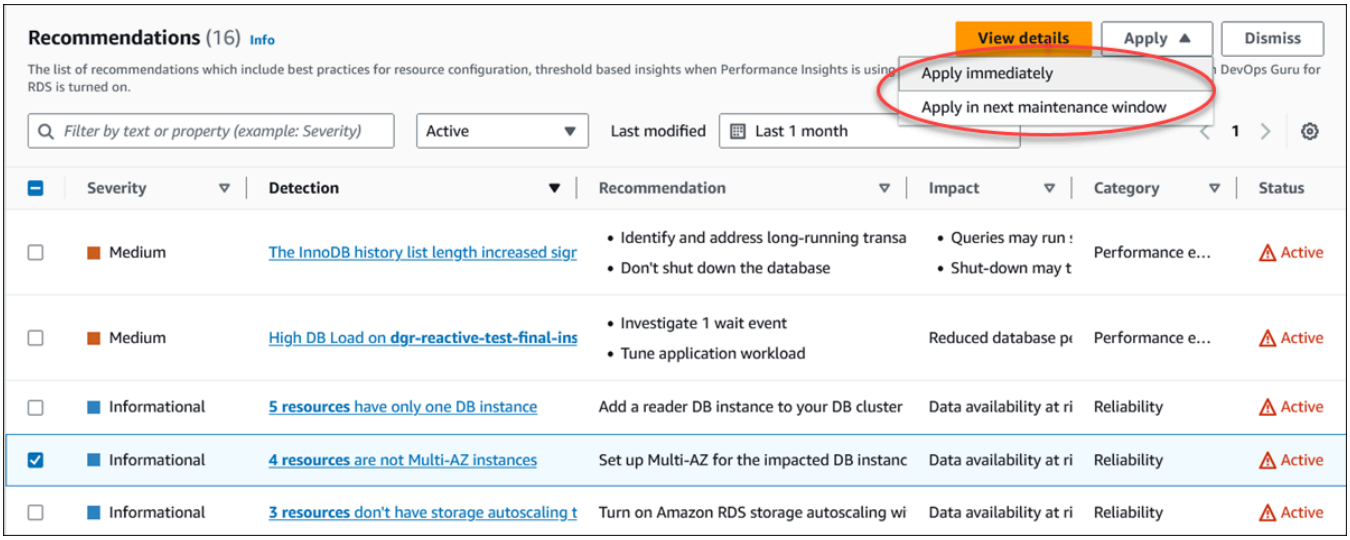
     The **Recommendations** page appears with the list of all recommendations.
   - Choose **Databases** and then choose **Recommendations** for a resource in the databases page.

     The details appear in the **Recommendations** tab for the selected recommendation.
   - Choose **Detection** for an active recommendation in the **Recommendations** page or the **Recommendations** tab in the **Databases** page.

     The recommendation details page appears.
3. Choose a recommendation, or one or more affected resources in the recommendation details page, and do any of the following:

   - Choose **Apply** and then choose **Apply immediately** to apply the recommendation immediately.
   - Choose **Apply** and then choose **Apply in next maintenance window** to schedule in the next maintenance window.

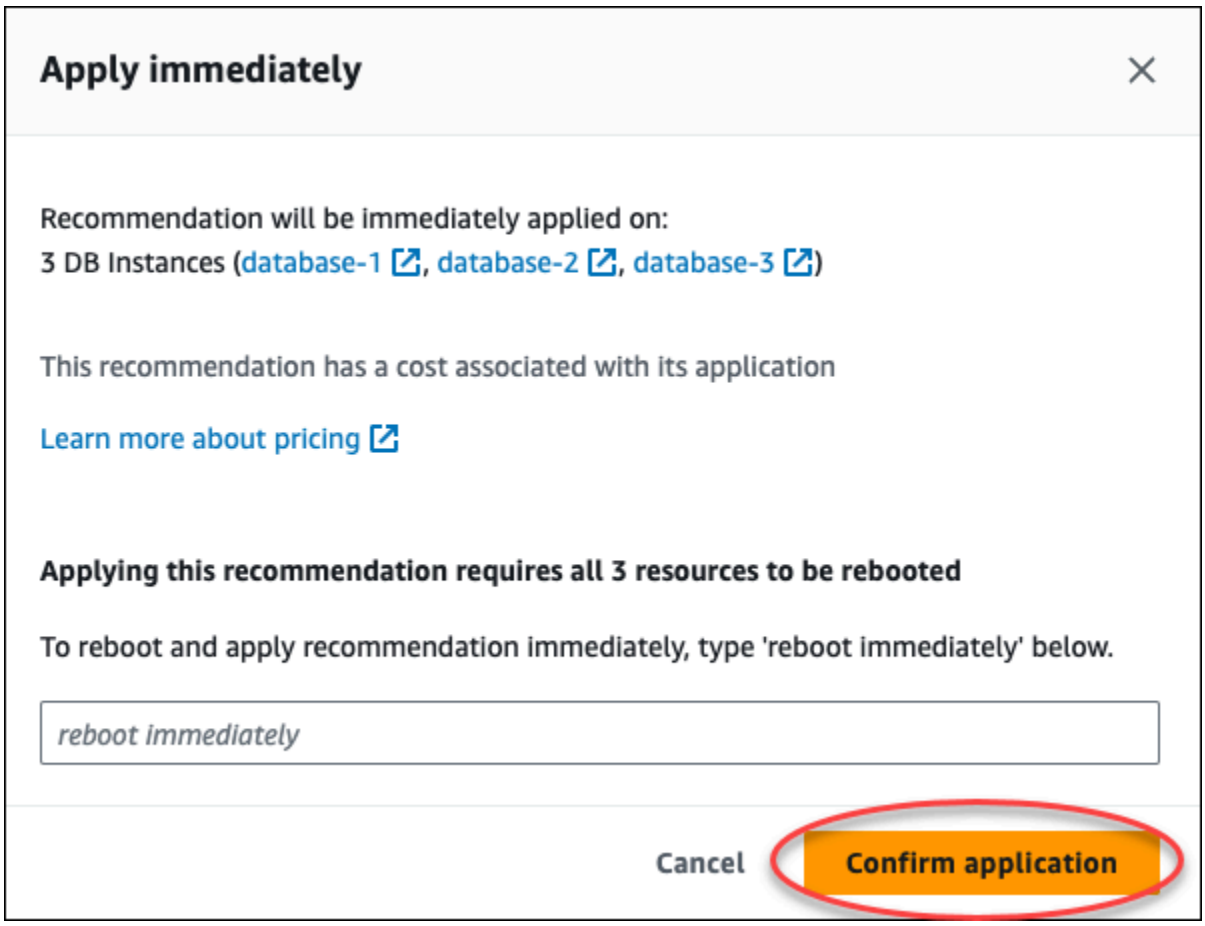
     The selected recommendation status is updated to pending until the next maintenance window.



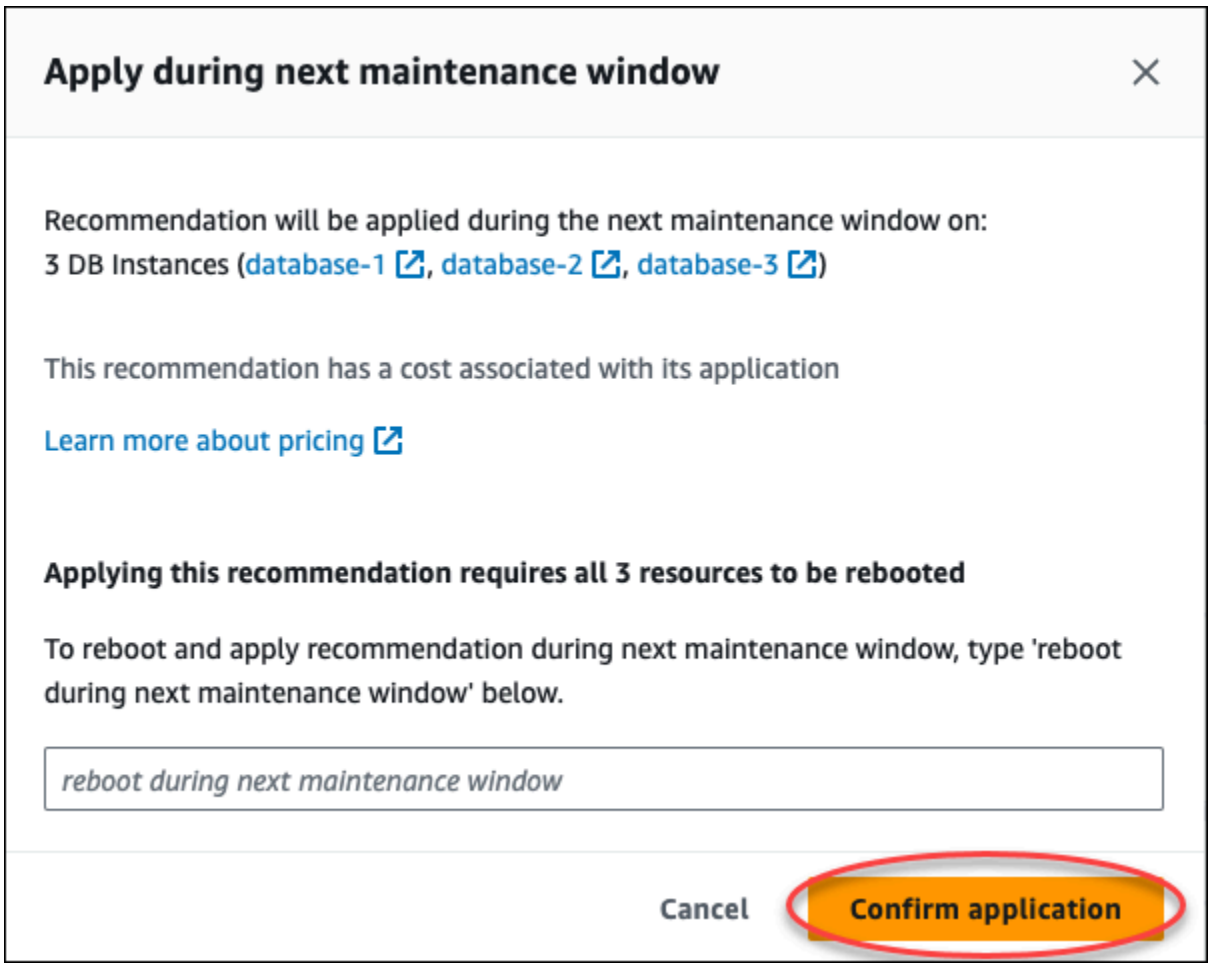
A confirmation window appears.

4. Choose **Confirm application** to apply the recommendation. This window confirms whether the resources need an automatic or manual restart for the changes to take effect.

   The following example shows the confirmation window to apply the recommendation immediately.



The following example shows the confirmation window to schedule applying the recommendation in the next maintenance window.



A banner displays a message when the recommendation applied is successful or has failed.

The following example shows the banner with the successful message.



The following example shows the banner with the failure message.



RDS API

To apply a configuration based RDS recommendation using the Amazon RDS API

1. Use the DescribeDBRecommendations operation. The `RecommendedActions` in the output can have one or more recommended actions.
2. Use the RecommendedAction object for each recommended action from step 1. The output contains `Operation` and `Parameters`.

   The following example shows the output with one recommended action.

```
   "RecommendedActions": [
       {
           "ActionId": "0b19ed15-840f-463c-a200-b10af1b552e3",
           "Title": "Turn on auto backup", // localized
           "Description": "Turn on auto backup for my-mysql-instance-1", //
localized
           "Operation": "ModifyDbInstance",
           "Parameters": [
               {
                   "Key": "DbInstanceIdentifier",
                   "Value": "my-mysql-instance-1"
               },
               {
                   "Key": "BackupRetentionPeriod",
                   "Value": "7"
               }
           ],
           "ApplyModes": ["immediately", "next-maintenance-window"],
           "Status": "applied"
       },
       ... // several others
   ],
```

3. Use the `operation` for each recommended action from the output in step 2 and input the `Parameters` values.
4. After the operation in step 2 is successful, use the ModifyDBRecommendation operation to modify the recommendation status.

Dismissing Amazon RDS recommendations

You can dismiss one or more Amazon RDS recommendations using the Amazon RDS console, AWS CLI, or Amazon RDS API.

Console

To dismiss one or more recommendations

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, perform any of the following:

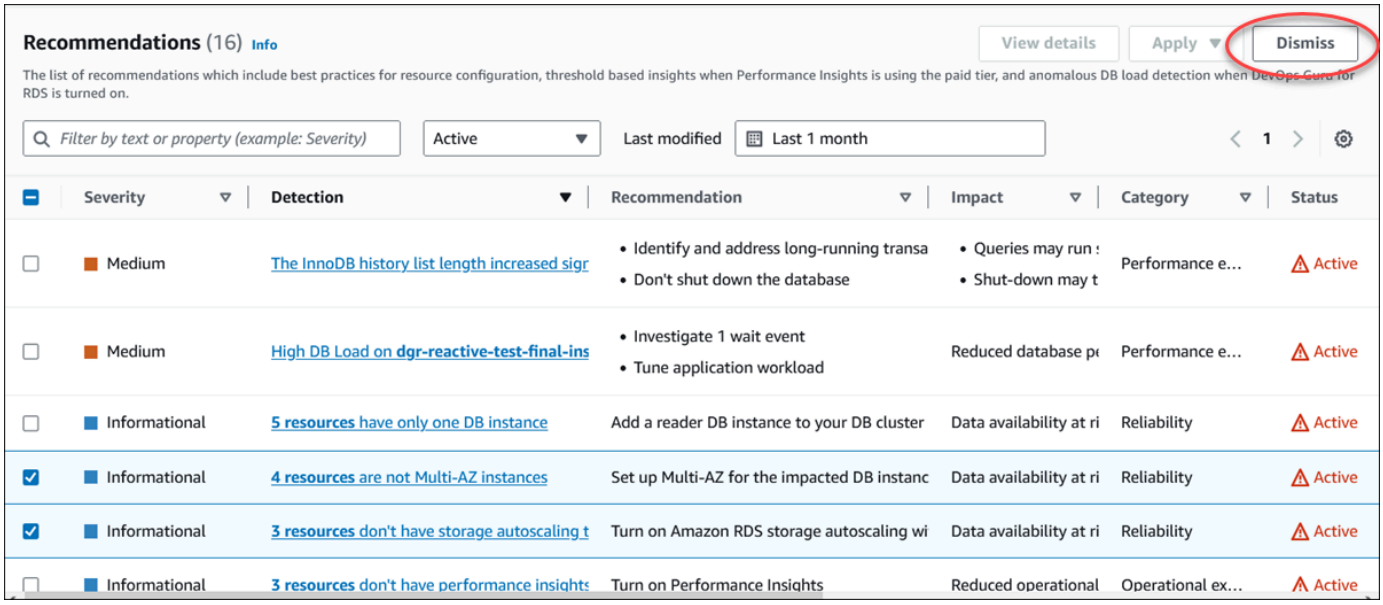
   - Choose **Recommendations**.

     The **Recommendations** page appears with the list of all recommendations.
   - Choose **Databases** and then choose **Recommendations** for a resource in the databases page.

     The details appear in the **Recommendations** tab for the selected recommendation.
   - Choose **Detection** for an active recommendation in the **Recommendations** page or the **Recommendations** tab in the **Databases** page.

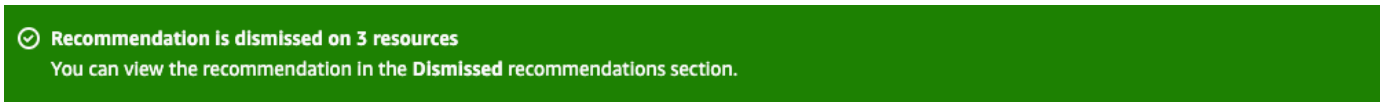
     The recommendation details page displays the list of affected resources.
3. Choose one or more recommendation, or one or more affected resources in the recommendation details page, and then choose **Dismiss**.

   The following example shows the **Recommendations** page with multiple active recommendations selected to dismiss.



A banner displays a message when the selected one or more recommendations are dismissed.

The following example shows the banner with the successful message.



The following example shows the banner with the failure message.



CLI

To dismiss a RDS recommendation using the AWS CLI

1. Run the command `aws rds describe-db-recommendations --filters "Name=status,Values=active"`.

   The output provides a list of recommendations in `active` status.
2. Find the `recommendationId` for the recommendation that you want to dismiss from step 1.
3. Run the command `>aws rds modify-db-recommendation --status dismissed --recommendationId <ID>` with the `recommendationId` from step 2 to dismiss the recommendation.

RDS API

To dismiss a RDS recommendation using the Amazon RDS API, use the ModifyDBRecommendation operation.

Modifying dismissed Amazon RDS recommendations to active recommendations

You can move one or more dismissed Amazon RDS recommendations to active recommendations using the Amazon RDS console, AWS CLI, or Amazon RDS API.

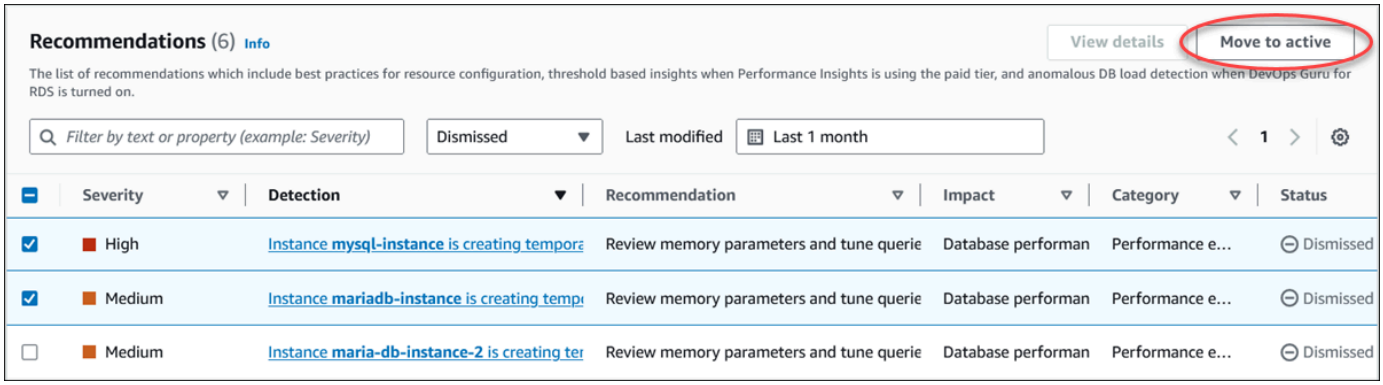
Console

To move one or more dismissed recommendations to active recommendations

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, perform any of the following:
   - Choose **Recommendations**.

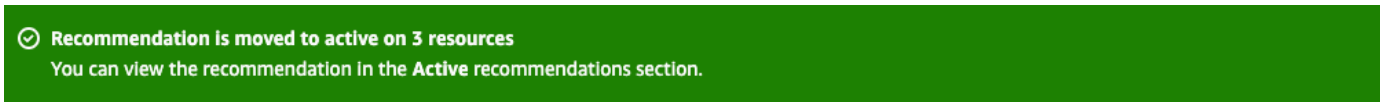
     The **Recommendations** page displays a list of recommendations sorted by the severity for all the resources in your account.
   - Choose **Databases** and then choose **Recommendations** for a resource in the databases page.

     The **Recommendations** tab displays the recommendations and its details for the selected resource.
3. Choose one or more dismissed recommendations from the list and then choose **Move to active**.



A banner displays a successful or failure message when the moving the selected recommendations from dismissed to active status.

The following example shows the banner with the successful message.



The following example shows the banner with the failure message.



CLI

To change a dismissed RDS recommendation to active recommendation using the AWS CLI

1. Run the command `aws rds describe-db-recommendations --filters "Name=status,Values=dismissed"`.

   The output provides a list of recommendations in `dismissed` status.
2. Find the `recommendationId` for the recommendation that you want to change the status from step 1.
3. Run the command `>aws rds modify-db-recommendation --status active --recommendationId <ID>` with the `recommendationId` from step 2 to change to active recommendation.

RDS API

To change a dismissed RDS recommendation to active recommendation using the Amazon RDS API, use the ModifyDBRecommendation operation.

Recommendations from Amazon RDS reference

Amazon RDS generates recommendations for a resource when the resource is created or modified. You can find examples of recommendations from Amazon RDS in the following table.

Type	Description	Recommendation	Downt require	Additional informati on
Magnetic volume is in use	Your DB instances are using magnetic storage. Magnetic storage isn't recommended for most of the DB instances. Choose a different storage type: General Purpose (SSD) or Provisioned IOPS.	Choose a different storage type: General Purpose (SSD) or Provisioned IOPS.	Yes	Previous generatio n volumes in the Amazon EC2 documentation.
Resource Automated backups is turned off	Automated backups aren't turned on for your DB instances. Automated backups are recommend ed because they enable point-in-time recovery of your DB instances.	Turn on automated backups with a retention period of up to 14 days.	Yes	Enabling automated backups Demystifying Amazon RDS backup storage costs on the AWS Database Blog
Engine minor version upgrade is required	Your database resources aren't running the latest minor DB engine version. The latest minor version contains the latest security fixes and other improvements.	Upgrade to latest engine version.	Yes	Upgrading a DB instance engine version

Type	Description	Recommendation	Downt require	Additional informati on
Enhanced Monitoring is turned off	Your database resources don't have Enhanced Monitoring turned on. Enhanced Monitoring provides real-time operating system metrics for monitoring and troubleshooting.	Turn on Enhanced Monitoring.	No	Monitoring OS metrics with Enhanced Monitoring

Type	Description	Recommendation	Downt require	Additional informati on
Storage encryption is turned off	Amazon RDS supports encryptio n at rest for all the database engines by using the keys that you manage in AWS Key Management Service (AWS KMS). On an active DB instance with Amazon RDS encryption, the data stored at rest in the storage is encrypted, similar to automated backups, read replicas, and snapshots. If encryption isn't turned on while creating a DB instance, you will need to create and restore an encrypted copy of the decrypted snapshot of the DB instance before you turn on the encryptio n.	Turn on encryption of data at rest for your DB instance.	Yes	Security in Amazon RDS Copying a DB snapshot for Amazon RDS

Type	Description	Recommendation	Downt require	Additional informati on
Performan ce Insights is turned off	Performance Insights monitors your DB instance load to help you analyze and resolve database performance issues. We recommend that you turn on Performance Insights.	Turn on Performance Insights.	No	Monitoring DB load with Performance Insights on Amazon RDS
DB instances have storage autoscaling turned off	Storage autoscaling isn't turned on for your DB instance. When the database workload increases, RDS storage autoscali ng automatically scales the storage capacity with zero downtime.	Turn on Amazon RDS storage autoscali ng with a specified maximum storage threshold	No	Managing capacity automatically with Amazon RDS storage autoscaling
RDS resources major versions update is required	Databases with the current major version for the DB engine won't be supported. We recommend that you upgrade to the latest major version which includes new functionality and enhancements.	Upgrade to the latest major version for the DB engine.	Yes	Upgrading a DB instance engine version Using Amazon RDS Blue/Green Deployments for database updates

Type	Description	Recommendation	Downt require	Additional informati on
RDS resources instance class update is required	Your DB instance is running an earlier generation DB instance class. We have replaced DB instance classes from an earlier generatio n with DB instance classes with better cost, performance, or both. We recommend that you run your DB instance with a DB instance class from a newer generation.	Upgrade the DB instance class.	Yes	Supported DB engines for DB instance classes
RDS resources using end of support engine edition under license-i ncluded	We recommend that you upgrade the major version to the latest engine version supported by Amazon RDS to continue with the current license support. The engine version of your database won't be supported with the current license.	We recommend that you upgrade your database to the latest supported version in Amazon RDS to continue using the licensed model.	Yes	Oracle major version upgrades

Type	Description	Recommendation	Downt require	Additional informati on
DB instances not using Multi-AZ deployment	We recommend that you use Multi-AZ deployment. The Multi-AZ deploymen ts enhance the availability and durability of the DB instance.	Set up Multi-AZ for the impacted DB instances	No Downti doesn't occur during this change Howev there is a possibl perfor ce impact For more inform on, see Conver g a DB instanc to a Multi- AZ deploy t for Amazo RDS	Pricing for Amazon RDS Multi-AZ

Type	Description	Recommendation	Downt require	Additional informati on
DB memory parameters are diverging from default	The memory parameters of the DB instances are significa ntly different from the default values. These settings can impact performance and cause errors. We recommend that you reset the custom memory parameters for the DB instance to their default values in the DB parameter group.	Reset the memory parameters to their default values.	No	Best practices for configuring performance parameters for Amazon RDS for MySQL on the AWS Database Blog

Type	Description	Recommendation	Downt require	Additional informati on
`InnoDB_Change_Buffering` parameter using less than optimum value	Change buffering allows a MySQL DB instance to defer a few writes, which are required to maintain secondary indexes. This feature was useful in environments with slow disks. The change buffering configuration improved the DB performance slightly but caused a delay in crash recovery and long shutdown times during upgrade. Set to `OFF` by default in MySQL version 8.4.	Set `InnoDB_Change_Buffering` parameter value to `NONE` in your DB parameter groups.	No	Best practices for configuring performance parameters for Amazon RDS for MySQL on the AWS Database Blog

Type	Description	Recommendation	Downt require	Additional informati on
Query cache parameter is turned on	When changes require that your query cache is purged, your DB instance will appear to stall. Most workloads don't benefit from a query cache. The query cache was removed from MySQL 8.0 and higher versions. We recommend that you set the query_cac he_type parameter to 0.	Set the `query_cac he_type` parameter value to `0` in your DB parameter groups.	Yes	Best practices for configuring performance parameters for Amazon RDS for MySQL on the AWS Database Blog
`log_outpu t` parameter is set to table	When `log_output` is set to `TABLE`, more storage is used than when `log_outpu t` is set to `FILE`. We recommend that you set the parameter to `FILE`, to avoid reaching the storage size limit. Set to `FILE` by default in MySQL 8.4 and higher versions.	Set the `log_outpu t` parameter value to `FILE` in your DB parameter groups.	No	MySQL database log files

Type	Description	Recommendation	Downt require	Additional informati on
Parameter groups not using huge pages	Large pages can increase database scalability, but your DB instance isn't using large pages. We recommend that you set the `use_large_pages` parameter value to `ONLY` in the DB parameter group for your DB instance.	Set the `use_large_pages` parameter value to `ONLY` in your DB parameter groups.	Yes	Turning on HugePages for an RDS for Oracle instance
`autovacuum` parameter is turned off	The autovacuum parameter is turned off for your DB instances. Turning autovacuum off increases the table and index bloat and impacts the performance. We recommend that you turn on autovacuum in your DB parameter groups.	Turn on the autovacuum parameter in your DB parameter groups.	No	Understanding autovacuum in Amazon RDS for PostgreSQL environments on the AWS Database Blog

Type	Description	Recommendation	Downt require	Additional informati on
synchrono us_commit parameter is turned off	When synchrono us_commit parameter is turned off, data can be lost in a database crash. The durability of the database is at risk. We recommend that you turn on the synchrono us_commit parameter.	Turn on synchrono us_commit parameter in your DB parameter groups.	Yes	Amazon Aurora PostgreSQL parameters: Replicati on, security, and logging on the AWS Database Blog
track_cou nts parameter is turned off	When the track_counts parameter is turned off, the database doesn't collect the database activity statistics. Autovacuu m requires these statistics to work correctly. We recommend that you set track_cou nts parameter to 1.	Set track_counts parameter to 1.	No	Run-time Statistics for PostgreSQL

Type	Description	Recommendation	Downt require	Additional informati on
enable_in dexonlysc an parameter is turned off	The query planner or optimizer can't use the index-only scan plan type when it is turned off. We recommend that you set the enable_in dexonlyscan parameter value to 1.	Set the enable_in dexonlyscan parameter value to 1.	No	Planner Method Configuration for PostgreSQL
enable_in dexscan parameter is turned off	The query planner or optimizer can't use the index scan plan type when it is turned off. We recommend that you set the enable_in dexscan value to 1.	Set the enable_in dexscan parameter value to 1.	No	Planner Method Configuration for PostgreSQL

Type	Description	Recommendation	Downt require	Additional informati on
innodb_fl ush_log_a t_trx parameter is turned off	The value of the innodb_fl ush_log_at_trx parameter of your DB instance isn't safe value. This parameter controls the persistence of commit operations to disk. We recommend that you set the innodb_fl ush_log_at_trx parameter to 1.	Set the innodb_fl ush_log_at_trx parameter value to 1.	No	Best practices for configuring performance parameters for Amazon RDS for MySQL on the AWS Database Blog
sync_binl og parameter is turned off	The synchronization of the binary log to disk isn't enforced before the transacti on commits are acknowledged in your DB instance. We recommend that you set the sync_binlog parameter value to 1.	Set the sync_binl og parameter value to 1.	No	Best practices for configuring replicati on parameters for Amazon RDS for MySQL on the AWS Database Blog

Type	Description	Recommendation	Downt require	Additional informati on
innodb_st ats_persi stent parameter is turned off	Your DB instance isn't configured to persist the InnoDB statistics to the disk. When the statistics aren't stored, they are recalculated each time the instance restarts and the table accessed. This leads to variations in the query execution plan. You can modify the value of this global parameter at the table level. We recommend that you set the innodb_st ats_persistent parameter value to ON.	Set the innodb_st ats_persistent parameter value to ON.	No	Best practices for configuring performance parameters for Amazon RDS for MySQL on the AWS Database Blog

Type	Description	Recommendation	Downt require	Additional informati on
innodb_op en_files parameter is low	The innodb_op en_files parameter controls the number of files InnoDB can open at one time. InnoDB opens all of the log and system tablespac e files when mysqld is running. Your DB instance has a low value for the maximum number of files InnoDB can open at one time. We recommend that you set the innodb_op en_files parameter to a minimum value of 65.	Set the innodb_op en_files parameter to a minimum value of 65.	Yes	InnoDB open files for MySQL

Type	Description	Recommendation	Downt require	Additional informati on
`max_user_ connectio ns` parameter is low	Your DB instance has a low value for the maximum number of simultaneous connections for each database account. We recommend setting the `max_user_ connections` parameter to a number greater than 5.	Increase the value of the `max_user_ connections` parameter to a number greater than 5.	Yes	Setting Account Resource Limits for MySQL
Read Replicas are open in writable mode	Your DB instance has a read replica in writable mode, which allows updates from clients. We recommend that you set the `read_only` parameter to `TrueIfReplica` so that the read replicas isn't in writable mode.	Set the `read_only` parameter value to `TrueIfReplica` .	No	Best practices for configuring replicati on parameters for Amazon RDS for MySQL on the AWS Database Blog

Type	Description	Recommendation	Downt require	Additional informati on
innodb_de fault_row _format parameter setting is unsafe	Your DB instance encounters a known issue: A table created in a MySQL version lower than 8.0.26 with the row_forma t set to COMPACT or REDUNDANT will be inaccessible and unrecoverable when the index exceeds 767 bytes. We recommend that you set the innodb_de fault_row _format parameter value to DYNAMIC.	Set the innodb_de fault_row _format parameter value to DYNAMIC.	No	Changes in MySQL 8.0.26

Type	Description	Recommendation	Downt requir	Additional informati on
general_l ogging parameter is turned on	The general logging is turned on for your DB instance. This setting is useful while troubleshooting the database issues. However, turning on general logging increases the amount of I/O operations and allocated storage space, which might result in contentio n and performance degradation. Check your requireme nts for general logging usage. We recommend that you set the general_l ogging parameter value to 0.	Check your requireme nts for general logging usage. If it isn't mandatory , we recommend that you to set the general_logging parameter value to 0.	No	Overview of RDS for MySQL database logs

Type	Description	Recommendation	Downt require	Additional informati on
RDS instance under-pro visioned for system memory capacity	We recommend that you tune your queries to use lesser memory or use a DB instance type with higher allocated memory. When the instance is running low on memory, then the database performan ce is impacted.	Use a DB instance with higher memory capacity	Yes	Scaling Your Amazon RDS Instance Verticall y and Horizontally on the AWS Database Blog Amazon RDS instance types Amazon RDS pricing
RDS instance under-pro visioned for system CPU capacity	We recommend that you tune your queries to use less CPU or modify your DB instance to use a DB instance class with higher allocated vCPUs. Database performance might decline when a DB instance is running low on CPU.	Use a DB instance with higher CPU capacity	Yes	Scaling Your Amazon RDS Instance Verticall y and Horizontally on the AWS Database Blog Amazon RDS instance types Amazon RDS pricing

Type	Description	Recommendation	Downt require	Additional informati on
RDS resources are not utilizing connectio n pooling correctly	We recommend that you enable Amazon RDS Proxy to efficient ly pool and share existing database connections. If you are already using a proxy for your database, configure it correctly to improve connection pooling and load balancing across multiple DB instances. RDS Proxy can help reduce the risk of connectio n exhaustion and downtime while improving availability and scalability.	Enable RDS Proxy or modify your existing proxy configuration	No	Scaling Your Amazon RDS Instance Verticall y and Horizontally on the AWS Database Blog Amazon RDS Proxy Amazon RDS Proxy Pricing

Type	Description	Recommendation	Downt require	Additional informati on
RDS instances are creating excessive temporary objects	We recommend that you tune your workload to prevent creating excessive temporary objects, or switch to RDS instance classes supporting optimized reads. RDS Optimized Reads improves database performan ce for workloads involving a large number of temporary objects and/or large temporary objects. Evaluate your workload to determine if using an instance with RDS Optimized Reads benefits your database workload.	Use a DB instance type with RDS Optimized Reads	Yes	Amazon RDS instance types Improving query performance for RDS for MySQL with Amazon RDS Optimized Reads Improving query performance for RDS for MariaDB with Amazon RDS Optimized Reads Improving query performance for RDS for PostgreSQ L with Amazon RDS Optimized Reads

Type	Description	Recommendation	Downt require	Additional informati on
RDS instances are under-pro visioned for system IOPS capacity	We recommend tuning the database workload to reduce IOPS or scale up the DB instance to a type with a higher default IOPS limit. The current DB instance can't support the Provisioned IOPS, or the database workload has high IOPS utilization.	Use a DB instance type with higher default IOPS limits	Yes	Amazon RDS instance types Amazon RDS DB instance storage Database load
RDS instances have under-provisioned Amazon EBS volumes	We recommend tuning the database workload to reduce IOPS or increase the Provisioned IOPS for the database. When IOPS utilizati on approaches the Provisioned IOPS, database performan ce might decline.	Provision more IOPS for the DB instance	Yes	Amazon RDS instance types Amazon RDS DB instance storage Database load

Type	Description	Recommendation	Downt require	Additional informati on
RDS instances are under-pro visioned for throughput capacity	We recommend tuning the database workload to reduce throughput or increase the provision ed throughput for the database. When throughput utilization approache s the provisioned throughput, database performance might be impacted.	Provision more throughput for the DB instance	Yes	Amazon RDS instance types Amazon RDS DB instance storage Database load

Type	Description	Recommendation	Downt require	Additional informati on
RDS instances are under-pro visioned for EBS I/O	We recommend tuning the database workload to reduce I/O operations or modifying the DB instance to use Amazon RDS io2 Block Express volumes which are designed for database workloads that require high performance, high throughput, and low latency. With the current workload, the database might not be able to process I/O operations at the required rate which can lead to performance degradation.	Use Amazon RDS io2 Block Express volumes for the RDS instance	No	Amazon RDS DB instance storage Amazon CloudWatch metrics for Amazon RDS Provisioned IOPS SSD volumes in the Amazon EBS User Guide

Viewing metrics in the Amazon RDS console

Amazon RDS integrates with Amazon CloudWatch to display a variety of RDS DB instance metrics in the RDS console. For descriptions of these metrics, see Metrics reference for Amazon RDS.

For your DB instance, the following categories of metrics are monitored:

- **CloudWatch** – Shows the Amazon CloudWatch metrics for RDS that you can access in the RDS console. You can also access these metrics in the CloudWatch console. Each metric includes a graph that shows the metric monitored over a specific time span. For a list of CloudWatch metrics, see Amazon CloudWatch metrics for Amazon RDS.
- **Enhanced monitoring** – Shows a summary of operating-system metrics when your RDS DB instance has turned on Enhanced Monitoring. RDS delivers the metrics from Enhanced Monitoring to your Amazon CloudWatch Logs account. Each OS metric includes a graph showing the metric monitored over a specific time span. For an overview, see Monitoring OS metrics with Enhanced Monitoring. For a list of Enhanced Monitoring metrics, see OS metrics in Enhanced Monitoring.
- **OS Process list** – Shows details for each process running in your DB instance.
- **Performance Insights** – Opens the Amazon RDS Performance Insights dashboard for a DB instance. For an overview of Performance Insights, see Monitoring DB load with Performance Insights on Amazon RDS. For a list of Performance Insights metrics, see Amazon CloudWatch metrics for Amazon RDS Performance Insights.

Amazon RDS now provides a consolidated view of Performance Insights and CloudWatch metrics in the Performance Insights dashboard. Performance Insights must be turned on for your DB instance to use this view. You can choose the new monitoring view in the **Monitoring** tab or **Performance Insights** in the navigation pane. To view the instructions for choosing this view, see Viewing combined metrics with the Performance Insights dashboard.

Viewing combined metrics with the Performance Insights dashboard

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

Amazon RDS provides a consolidated view of Performance Insights and CloudWatch metrics for your DB instance in the Performance Insights dashboard. You can use the preconfigured dashboard or create a custom dashboard. The preconfigured dashboard provides the most commonly used metrics to help diagnose performance issues for a database engine. Alternatively, you can create a custom dashboard with the metrics for a database engine that meet your analysis requirements. Then, use this dashboard for all the DB instances of that database engine type in your AWS account.

You can choose the monitoring view in the **Monitoring** tab or **Performance Insights** in the navigation pane.

Performance Insights must be turned on for your DB instance to view the combined metrics in the Performance Insights dashboard. For more information about turning on Performance Insights, see Turning Performance Insights on and off for Amazon RDS.

In the following sections, you can learn to display Performance Insights and CloudWatch metrics.

Topics

- Choosing the new monitoring view from the Monitoring tab
- Choosing the new monitoring view from the Performance Insights page
- Creating a custom dashboard with Performance Insights
- Choosing the preconfigured dashboard with Performance Insights

Choosing the new monitoring view from the Monitoring tab

Important

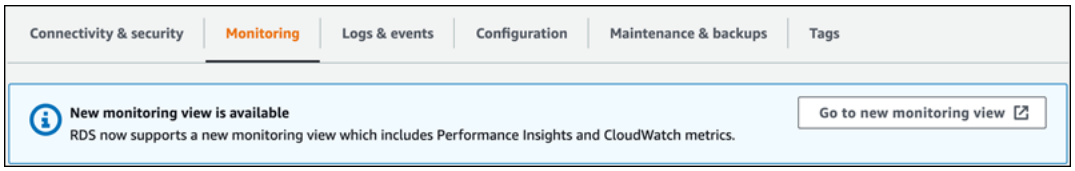
AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

From the Amazon RDS console, you can choose the new monitoring view to view Performance Insights and CloudWatch metrics for your DB instance.

To choose the new monitoring view in the Monitoring tab

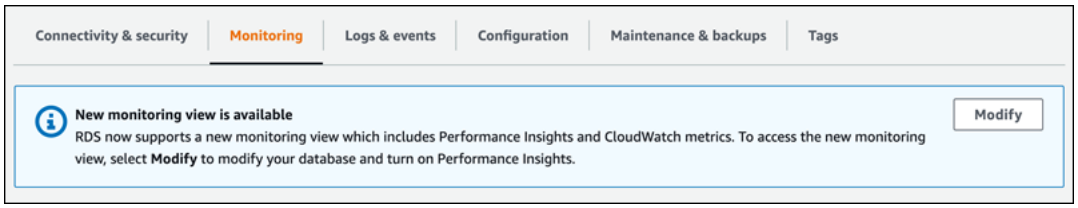
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Databases**.
3. Choose the DB instance that you want to monitor.
4. Scroll down and choose the **Monitoring** tab.

   A banner appears with the option to choose the new monitoring view. The following example shows the banner to choose the new monitoring view.



5. Choose **Go to new monitoring view** to open the Performance Insights dashboard with Performance Insights and CloudWatch metrics for your DB instance.
6. (Optional) If Performance Insights is turned off for your DB instance, a banner appears with the option to modify your DB cluster and turn on Performance Insights.

   The following example shows the banner to modify the DB cluster in the **Monitoring** tab .



   Choose **Modify** to modify your DB cluster and turn on Performance Insights. For more information about turning on Performance Insights, see Turning Performance Insights on and off for Amazon RDS

Choosing the new monitoring view from the Performance Insights page

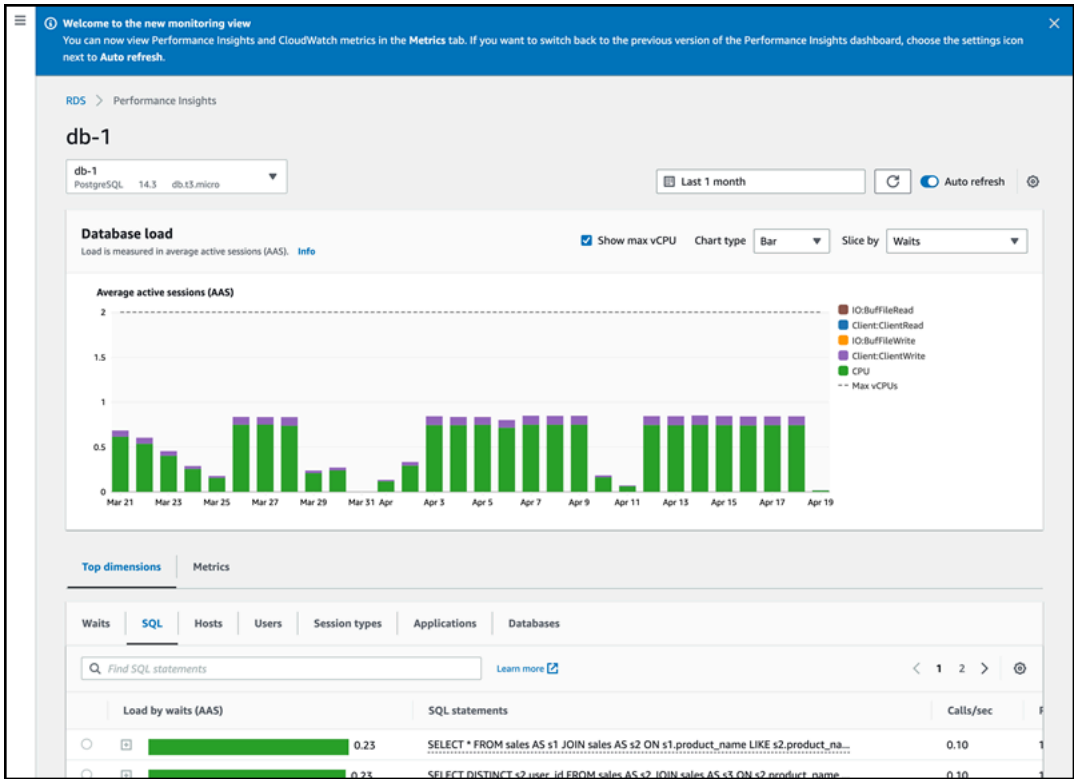
Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

From the Amazon RDS console, you can choose the new monitoring view to view Performance Insights and CloudWatch metrics for your DB instance.

To choose the new monitoring view with Performance Insights in the navigation pane

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance to view the Performance Insights dashboard that shows both Performance Insights and CloudWatch metrics for your DB instance.



Creating a custom dashboard with Performance Insights

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights. We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.

If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.

With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

In the new monitoring view, you can create a custom dashboard with the metrics you need to meet your analysis requirements.

You can create a custom dashboard by selecting Performance Insights and CloudWatch metrics for your DB instance. You can use this custom dashboard for other DB instances of the same database engine type in your AWS account.

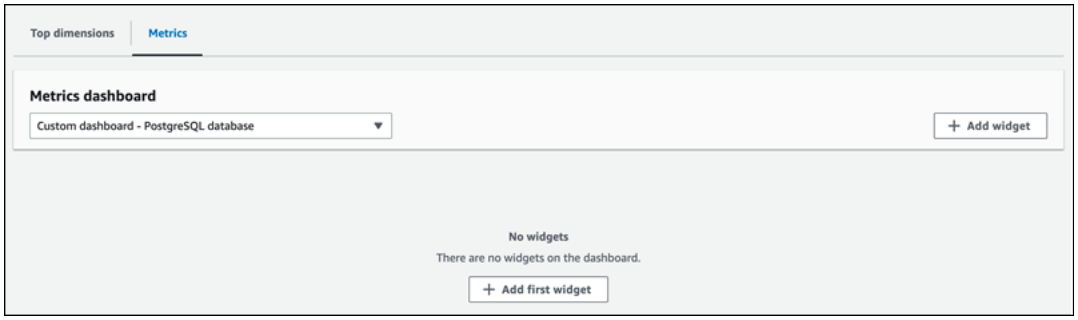
Note

The customized dashboard supports up to 50 metrics.

Use the widget settings menu to edit or delete the dashboard, and move or resize the widget window.

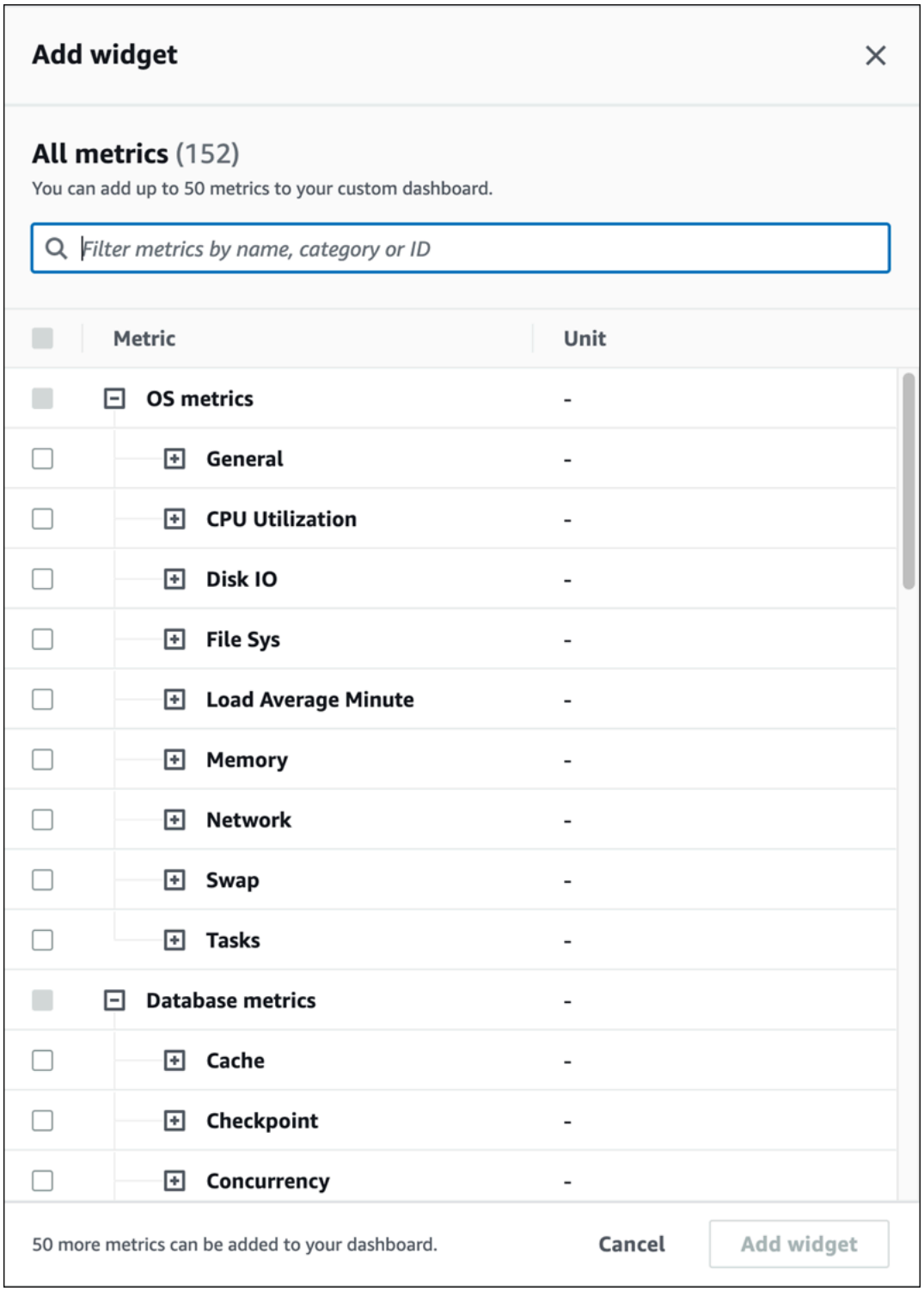
To create a custom dashboard with Performance Insights in the navigation pane

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.
4. Scroll down to the **Metrics tab** in the window.
5. Select the custom dashboard from the drop down list. The following example shows the custom dashboard creation.



6. Choose **Add widget** to open the **Add widget** window. You can open and view the available operating system (OS) metrics, database metrics, and CloudWatch metrics in the window.

   The following example shows the **Add widget** window with the metrics.



7. Select the metrics that you want to view in the dashboard and choose **Add widget**. You can use the search field to find a specific metric.

   The selected metrics appear on your dashboard.

8. (Optional) If you want to modify or delete your dashboard, choose the settings icon on the upper right of the widget, and then select one of the following actions in the menu.

   - **Edit** – Modify the metrics list in the window. Choose **Update widget** after you select the metrics for your dashboard.
   - **Delete** – Deletes the widget. Choose **Delete** in the confirmation window.

Choosing the preconfigured dashboard with Performance Insights

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

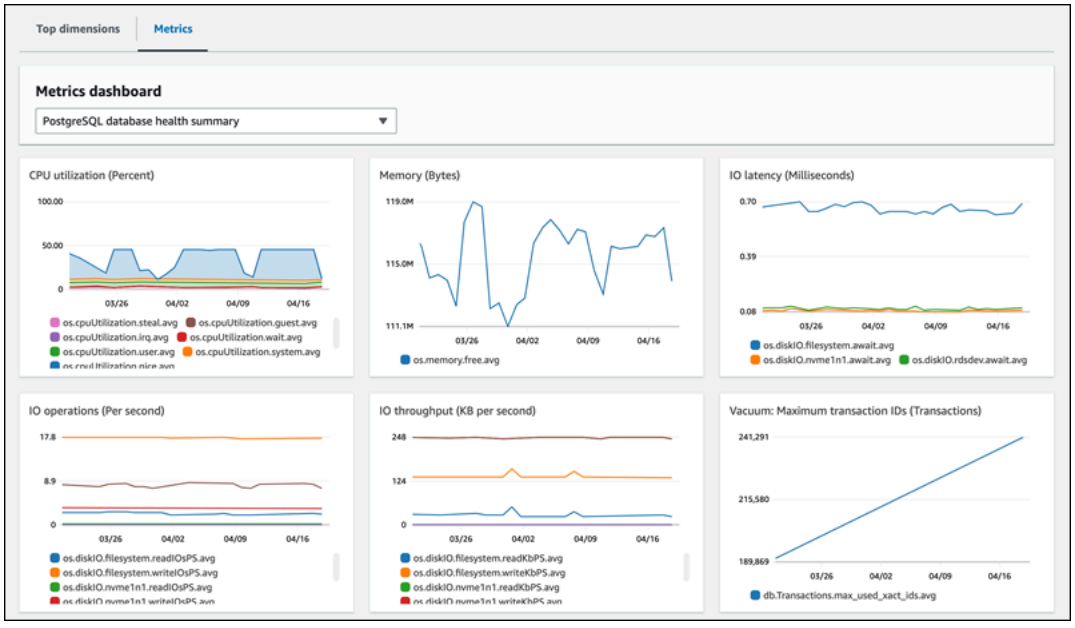
You can view the most commonly used metrics with the preconfigured dashboard. This dashboard helps diagnose performance issues with a database engine and reduce the average recovery time from hours to minutes.

> **Note**
>
> This dashboard can't be edited.

To choose the preconfigured dashboard with Performance Insights in the navigation pane

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.
4. Scroll down to the **Metrics tab** in the window
5. Select a preconfigured dashboard from the drop down list.

   You can view the metrics for the DB instance in the dashboard. The following example shows a preconfigured metrics dashboard.



Monitoring Amazon RDS metrics with Amazon CloudWatch

Amazon CloudWatch is a metrics repository. The repository collects and processes raw data from Amazon RDS into readable, near real-time metrics. For a complete list of Amazon RDS metrics sent to CloudWatch, see Metrics reference for Amazon RDS.

To analyze and troubleshoot the performance of your databases at scale, use CloudWatch Database Insights.

Topics

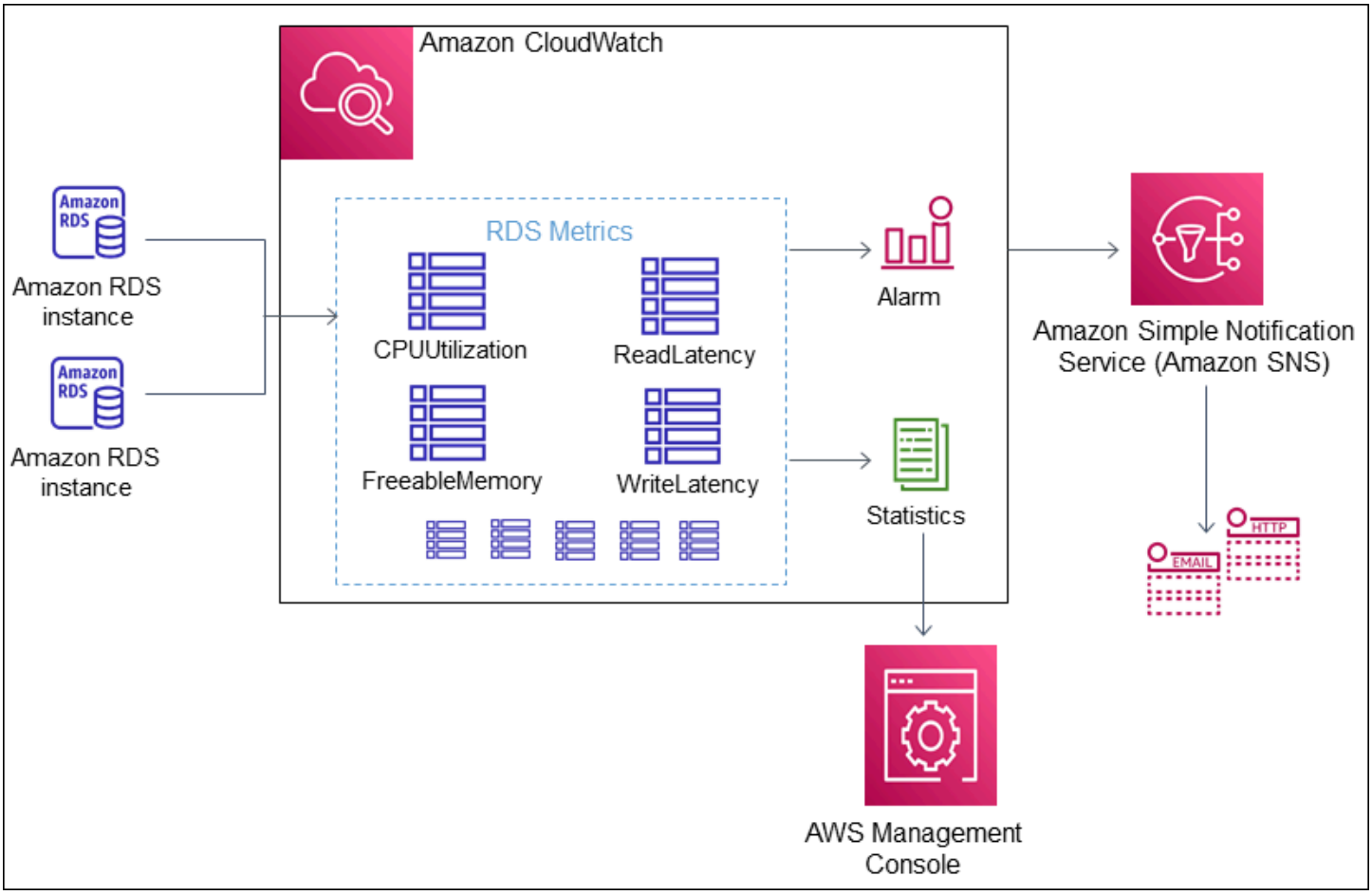
- Overview of Amazon RDS and Amazon CloudWatch
- Viewing DB instance metrics in the CloudWatch console and AWS CLI
- Exporting Performance Insights metrics to CloudWatch
- Creating CloudWatch alarms to monitor Amazon RDS
- Tutorial: Creating an Amazon CloudWatch alarm for Multi-AZ DB cluster replica lag for Amazon RDS

Overview of Amazon RDS and Amazon CloudWatch

By default, Amazon RDS automatically sends metric data to CloudWatch in 1-minute periods. For example, the `CPUUtilization` metric records the percentage of CPU utilization for a DB instance over time. Data points with a period of 60 seconds (1 minute) are available for 15 days. This means that you can access historical information and see how your web application or service is performing.

You can now export Performance Insights metrics dashboards from Amazon RDS to Amazon CloudWatch. You can export either the preconfigured or customized metrics dashboards as a new dashboard or add them to an existing CloudWatch dashboard. The exported dashboard is available to view in the CloudWatch console. For more information on how to export the Performance Insights metrics dashboards to CloudWatch, see Exporting Performance Insights metrics to CloudWatch.

As shown in the following diagram, you can set up alarms for your CloudWatch metrics. For example, you might create an alarm that signals when the CPU utilization for an instance is over 70%. You can configure Amazon Simple Notification Service to email you when the threshold is passed.



Amazon RDS publishes the following types of metrics to Amazon CloudWatch:

- Metrics for your RDS DB instances

  For a table of these metrics, see Amazon CloudWatch metrics for Amazon RDS.

- Performance Insights metrics

  For a table of these metrics, see Amazon CloudWatch metrics for Amazon RDS Performance Insights and Performance Insights counter metrics.

- Enhanced Monitoring metrics (published to Amazon CloudWatch Logs)

  For a table of these metrics, see OS metrics in Enhanced Monitoring.

- Usage metrics for the Amazon RDS service quotas in your AWS account

  For a table of these metrics, see Amazon CloudWatch usage metrics for Amazon RDS. For more information about Amazon RDS quotas, see Quotas and constraints for Amazon RDS.

For more information about CloudWatch, see What is Amazon CloudWatch? in the *Amazon CloudWatch User Guide*. For more information about CloudWatch metrics retention, see Metrics retention.

Viewing DB instance metrics in the CloudWatch console and AWS CLI

Following, you can find details about how to view metrics for your DB instance using CloudWatch. For information on monitoring metrics for your DB instance's operating system in real time using CloudWatch Logs, see Monitoring OS metrics with Enhanced Monitoring.

When you use Amazon RDS resources, Amazon RDS sends metrics and dimensions to Amazon CloudWatch every minute.

You can now export Performance Insights metrics dashboards from Amazon RDS to Amazon CloudWatch and view these metrics in the CloudWatch console. For more information on how to export the Performance Insights metrics dashboards to CloudWatch, see Exporting Performance Insights metrics to CloudWatch.

Use the following procedures to view the metrics for Amazon RDS in the CloudWatch console and CLI.

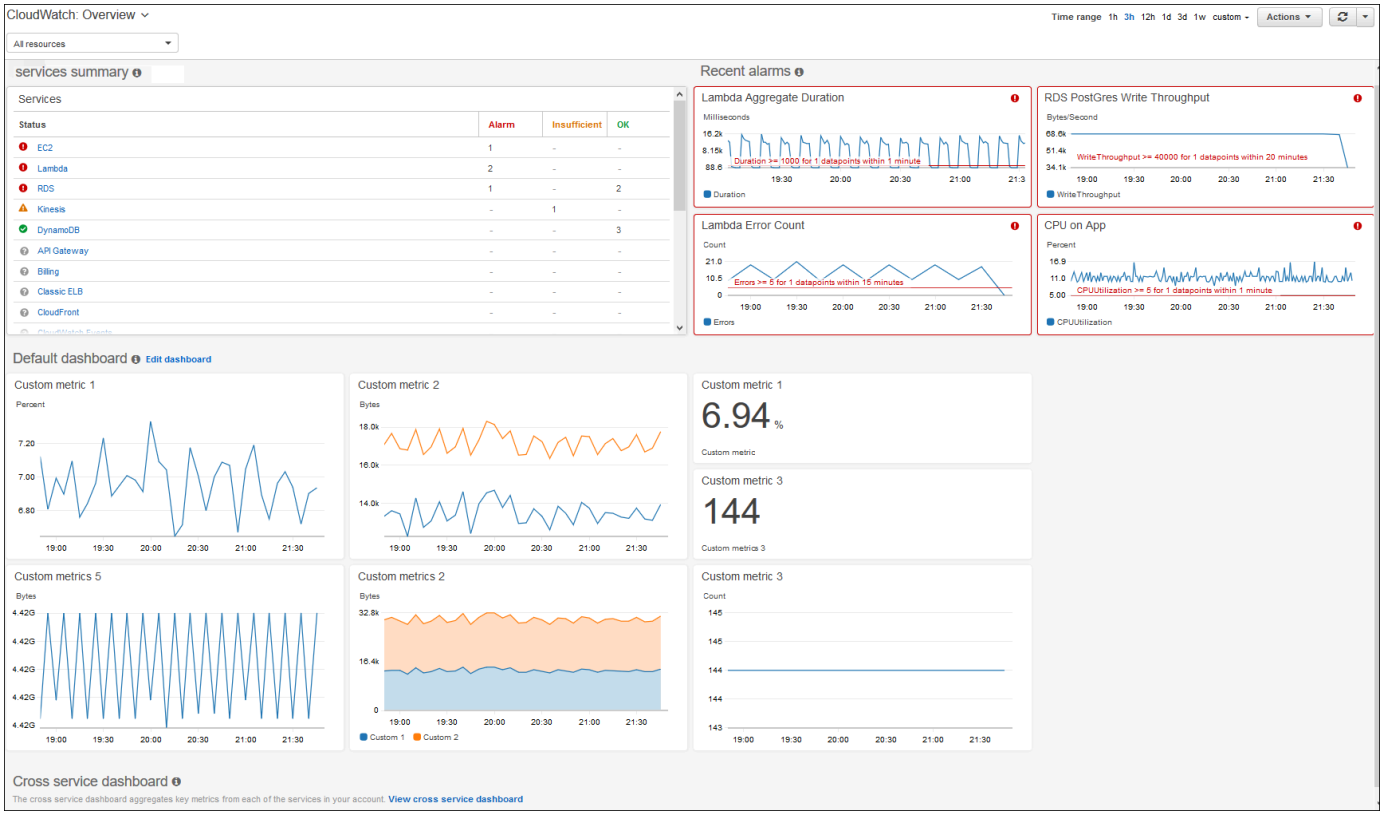
Console

To view metrics using the Amazon CloudWatch console

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

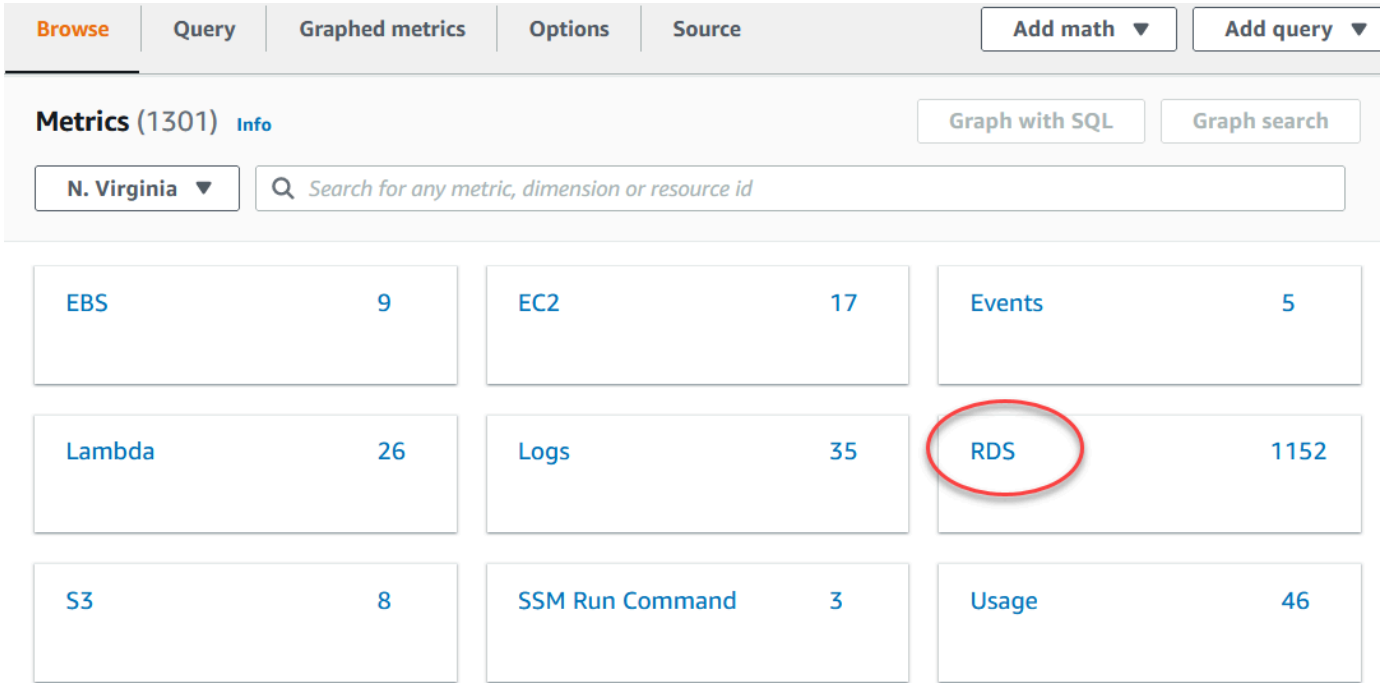
1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

   The CloudWatch overview home page appears.



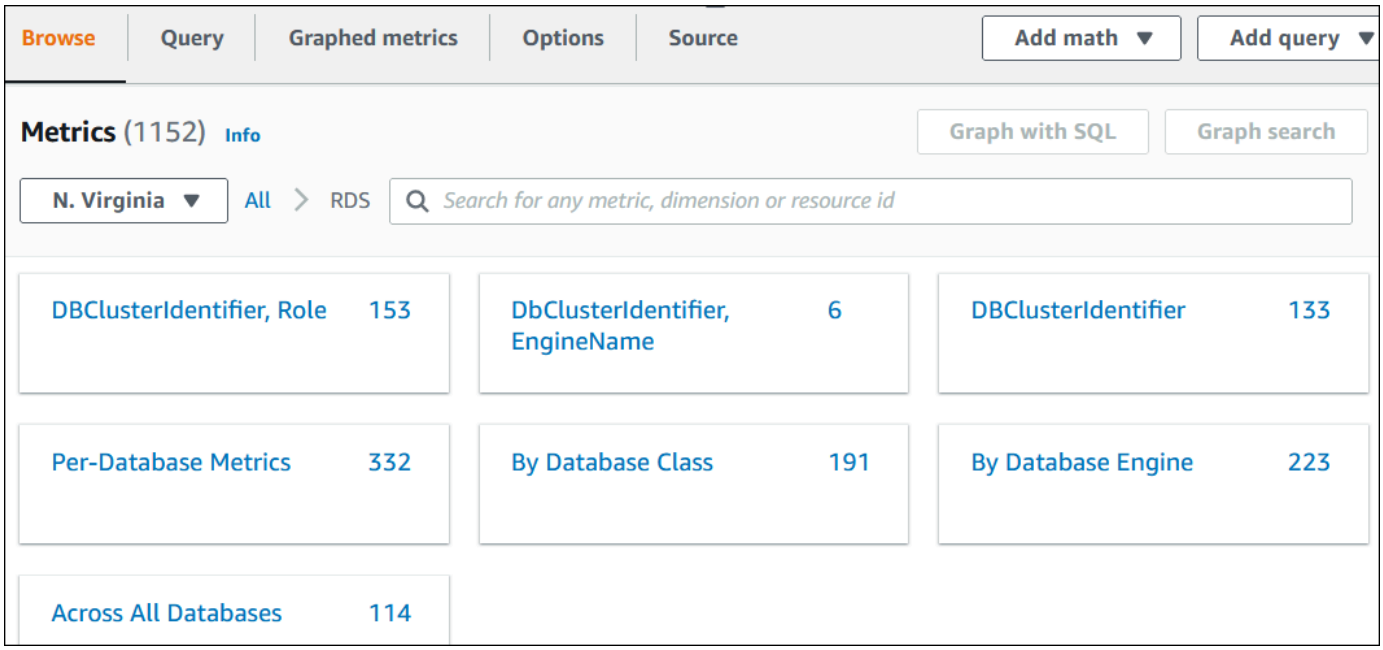
2. If necessary, change the AWS Region. From the navigation bar, choose the AWS Region where your AWS resources are. For more information, see [Regions and endpoints](#).

3. In the navigation pane, choose **Metrics** and then **All metrics**.

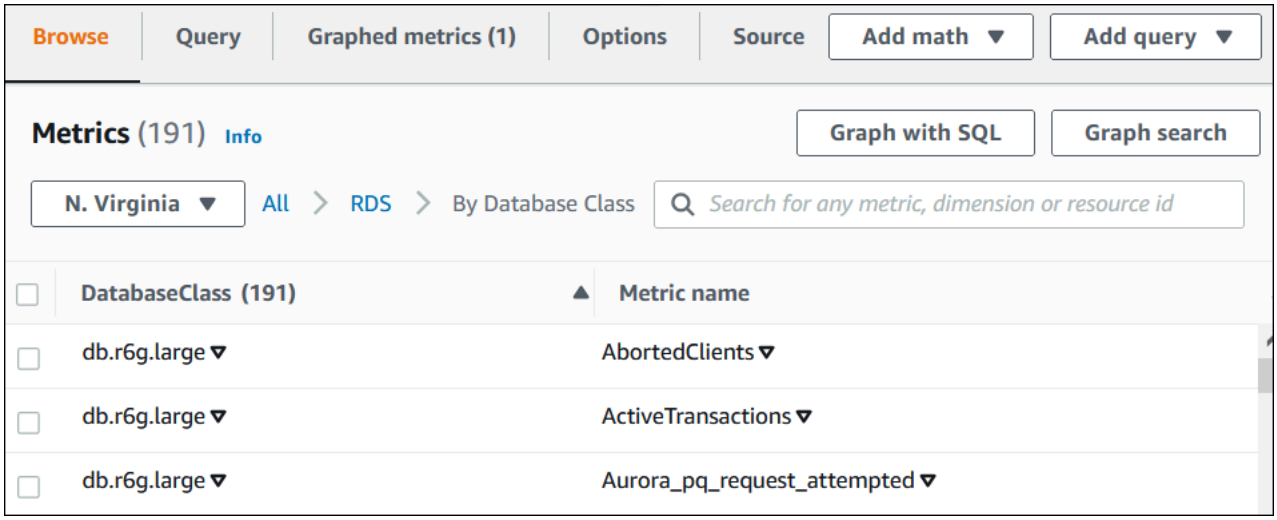


4. Scroll down and choose the **RDS** metric namespace.

The page displays the Amazon RDS dimensions. For descriptions of these dimensions, see [Amazon CloudWatch dimensions for Amazon RDS](#).



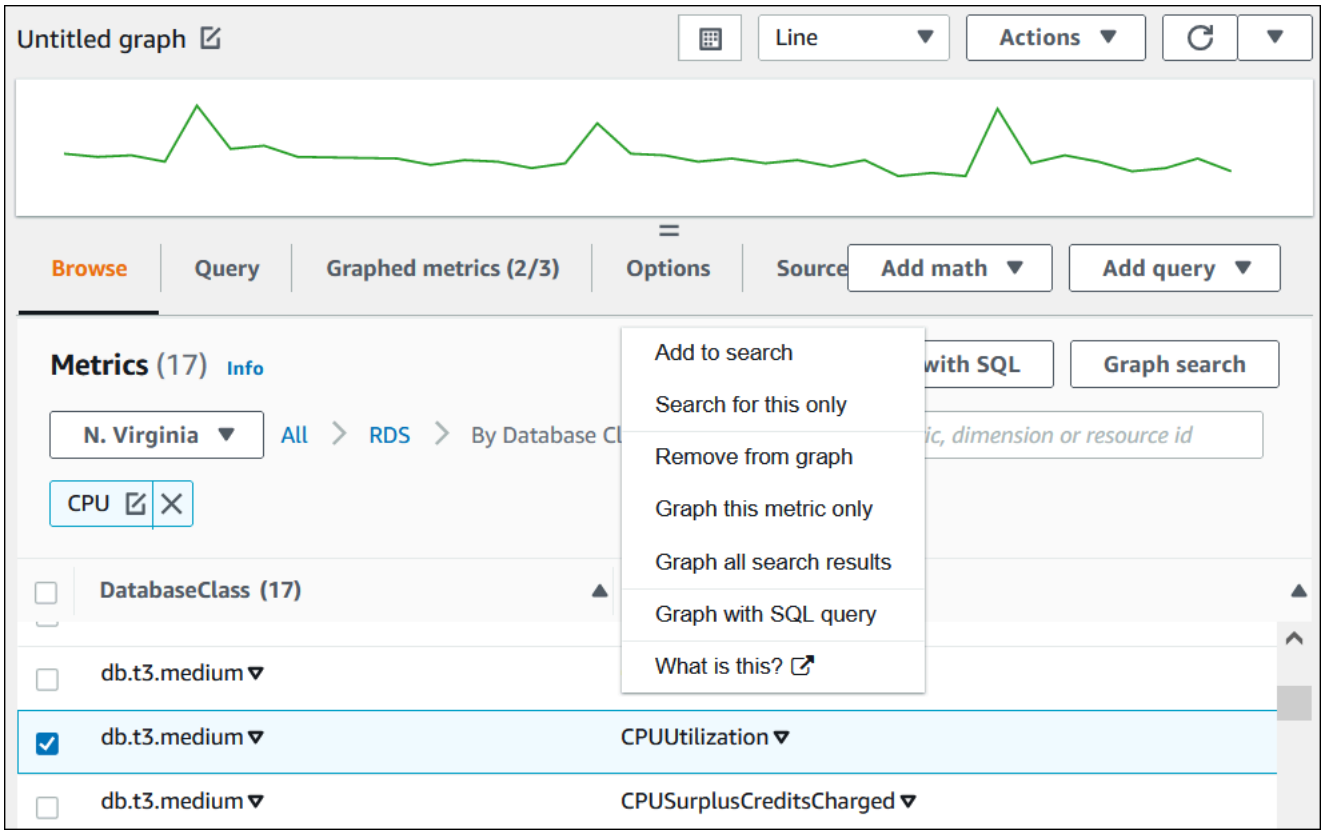
5. Choose a metric dimension, for example **By Database Class**.



6. Do any of the following actions:

- To sort the metrics, use the column heading.
- To graph a metric, select the check box next to the metric.
- To filter by resource, choose the resource ID, and then choose **Add to search**.
- To filter by metric, choose the metric name, and then choose **Add to search**.

The following example filters on the **db.t3.medium** class and graphs the **CPUUtilization** metric.



AWS CLI

To obtain metric information by using the AWS CLI, use the CloudWatch command `list-metrics`. In the following example, you list all metrics in the `AWS/RDS` namespace.

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

To obtain metric data, use the command `get-metric-data`.

The following example gets `CPUUtilization` statistics for instance `my-instance` over the specific 24-hour period, with a 5-minute granularity.

Create a JSON file `CPU_metric.json` with the following contents.

```
{
   "StartTime" : "2023-12-25T00:00:00Z",
```

```
   "EndTime" : "2023-12-26T00:00:00Z",
   "MetricDataQueries" : [{
     "Id" : "cpu",
     "MetricStat" : {
    "Metric" : {
        "Namespace" : "AWS/RDS",
        "MetricName" : "CPUUtilization",
        "Dimensions" : [{ "Name" : "DBInstanceIdentifier" , "Value" : my-instance}]
    },
       "Period" : 360,
       "Stat" : "Minimum"
     }
   }]
}
```

Example

For Linux, macOS, or Unix:

```
aws cloudwatch get-metric-data \
    --cli-input-json file://CPU_metric.json
```

For Windows:

```
aws cloudwatch get-metric-data ^
     --cli-input-json file://CPU_metric.json
```

Sample output appears as follows:

```
{
    "MetricDataResults": [
        {
            "Id": "cpu",
            "Label": "CPUUtilization",
            "Timestamps": [
                "2023-12-15T23:48:00+00:00",
                "2023-12-15T23:42:00+00:00",
                "2023-12-15T23:30:00+00:00",
                "2023-12-15T23:24:00+00:00",
                ...
            ],
            "Values": [
```

```
                13.299778337027714,
                13.677507543049558,
                14.24976250395827,
                13.02521708695145,
                ...
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

For more information, see Getting statistics for a metric in the *Amazon CloudWatch User Guide*.

Exporting Performance Insights metrics to CloudWatch

Performance Insights lets you export the preconfigured or custom metrics dashboard for your DB instance to Amazon CloudWatch. You can export the metrics dashboard as a new dashboard or add it to an existing CloudWatch dashboard. When you choose to add the dashboard to an existing CloudWatch dashboard, you can create a header label so that the metrics appear in a separate section in the CloudWatch dashboard.

You can view the exported metrics dashboard in the CloudWatch console. If you add new metrics to a Performance Insights metrics dashboard after you export it, you must export this dashboard again to view the new metrics in the CloudWatch console.

You can also select a metric widget in the Performance Insights dashboard and view the metrics data in the CloudWatch console.

For more information about viewing the metrics in the CloudWatch console, see Viewing DB instance metrics in the CloudWatch console and AWS CLI.

In the following sections, export Performance Insights metrics to CloudWatch as a new or existing dashboard and view Performance Insights metrics in CloudWatch.

Topics

- Exporting Performance Insights metrics as a new dashboard to CloudWatch
- Adding Performance Insights metrics to an existing CloudWatch dashboard
- Viewing a Performance Insights metric widget in CloudWatch

Exporting Performance Insights metrics as a new dashboard to CloudWatch

Choose a preconfigured or custom metrics dashboard from the Performance Insights dashboard and export it as a new dashboard to CloudWatch. You can view the exported dashboard in the CloudWatch console.

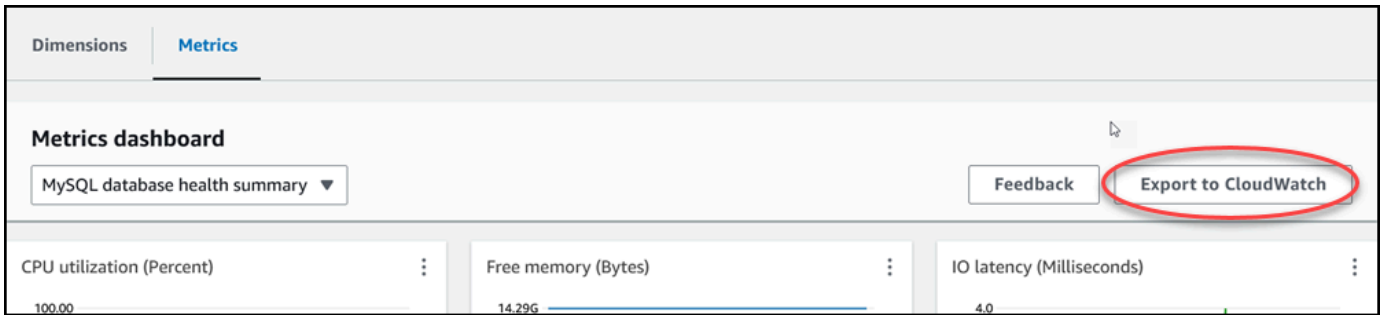
To export a Performance Insights metric dashboard as a new dashboard to CloudWatch

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.

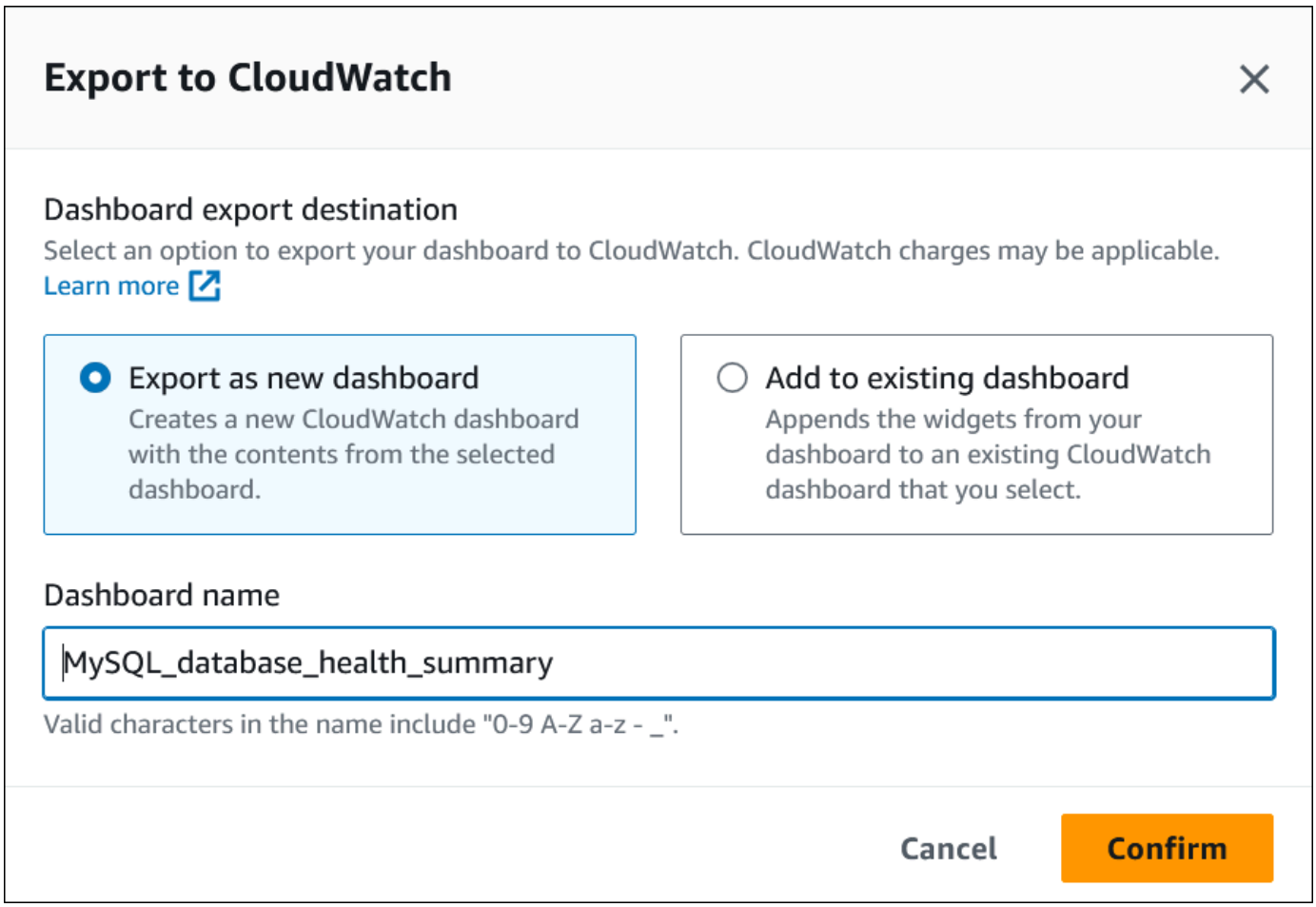
   The Performance Insights dashboard appears for the DB instance.
4. Scroll down and choose **Metrics**.

   By default, the preconfigured dashboard with Performance Insights metrics appears.
5. Choose a preconfigured or custom dashboard and then choose **Export to CloudWatch**.

   The **Export to CloudWatch** window appears.

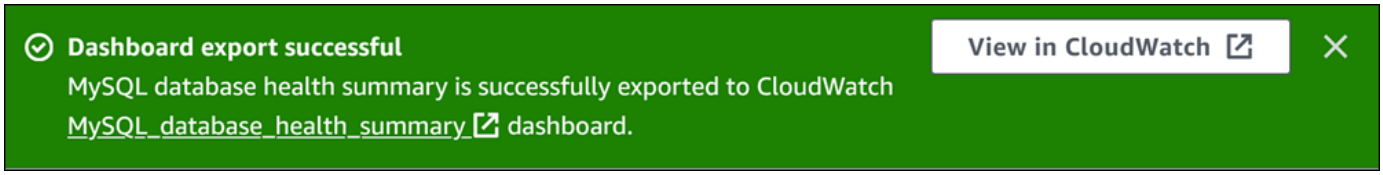


6. Choose **Export as new dashboard**.



7. Enter a name for the new dashboard in the **Dashboard name** field and choose **Confirm**.

   A banner displays a message after the dashboard export is successful.



8. Choose the link or **View in CloudWatch** in the banner to view the metrics dashboard in the CloudWatch console.

Adding Performance Insights metrics to an existing CloudWatch dashboard

Add a preconfigured or custom metrics dashboard to an existing CloudWatch dashboard. You can add a label to the metrics dashboard to appear in a separate section in the CloudWatch dashboard.

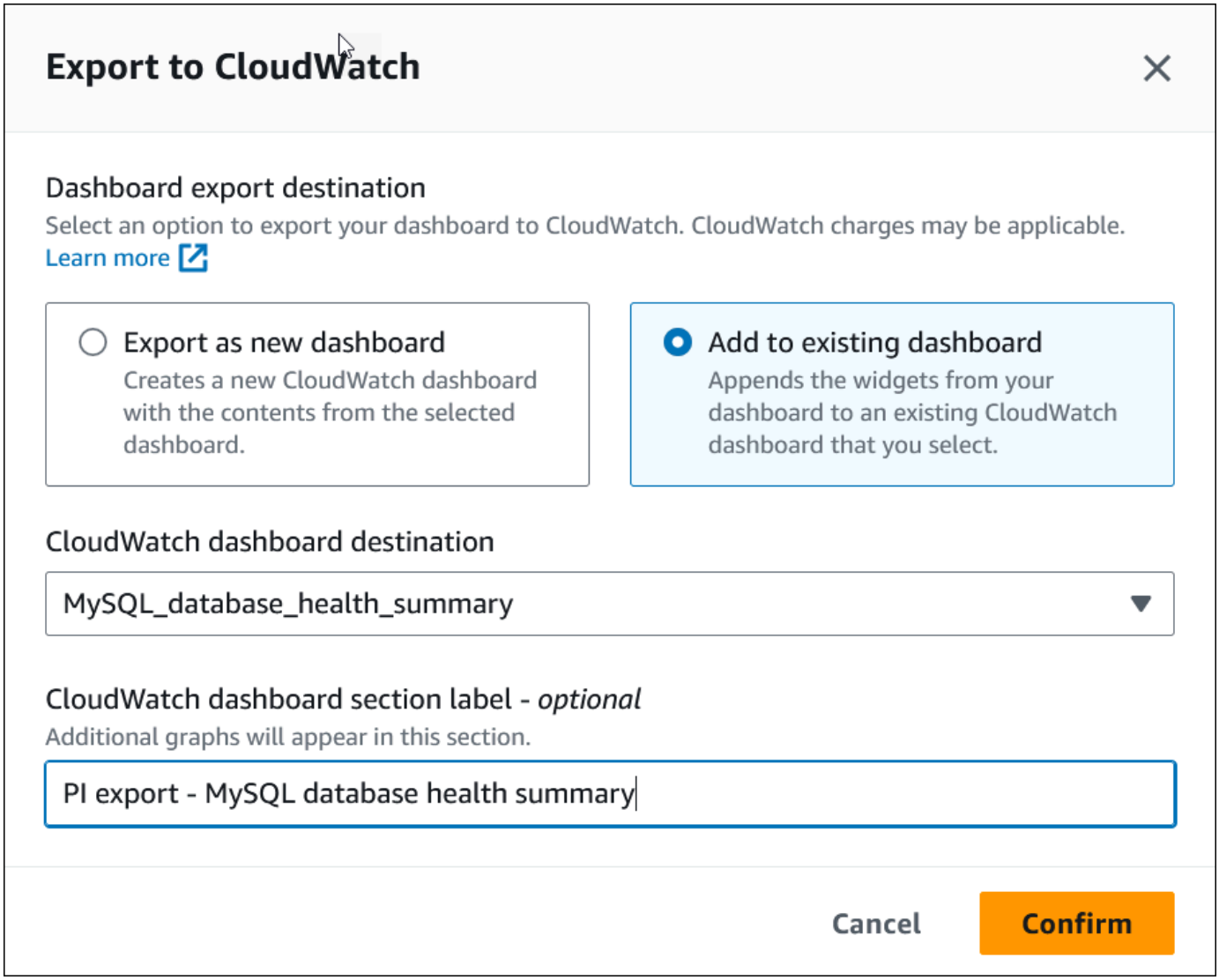
To export the metrics to an existing CloudWatch dashboard

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.

   The Performance Insights dashboard appears for the DB instance.
4. Scroll down and choose **Metrics**.

   By default, the preconfigured dashboard with Performance Insights metrics appears.
5. Choose the preconfigured or custom dashboard and then choose **Export to CloudWatch**.

   The **Export to CloudWatch** window appears.
6. Choose **Add to existing dashboard**.



7. Specify the dashboard destination and label, and then choose **Confirm**.

   - **CloudWatch dashboard destination** - Choose an existing CloudWatch dashboard.
   - **CloudWatch dashboard section label - optional** - Enter a name for the Performance Insights metrics to appear in this section in the CloudWatch dashboard.

   A banner displays a message after the dashboard export is successful.

8. Choose the link or **View in CloudWatch** in the banner to view the metrics dashboard in the CloudWatch console.

Viewing a Performance Insights metric widget in CloudWatch

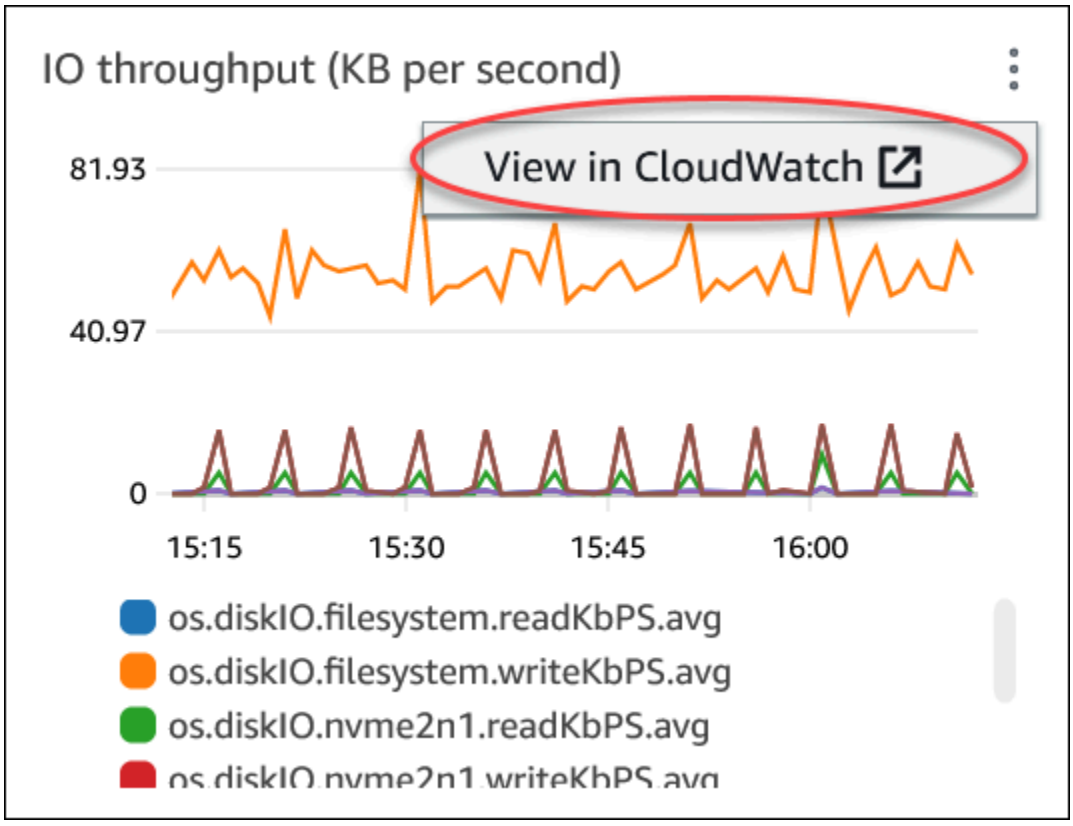
Select a Performance Insights metric widget in the Amazon RDS Performance Insights dashboard and view the metric data in the CloudWatch console.

To export a metric widget and view the metrics data in the CloudWatch console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.

   The Performance Insights dashboard appears for the DB instance.
4. Scroll down to **Metrics**.

   By default, the preconfigured dashboard with Performance Insights metrics appears.
5. Choose a metric widget and then choose **View in CloudWatch** in the menu.



   The metric data appears in the CloudWatch console.

Creating CloudWatch alarms to monitor Amazon RDS

You can create a CloudWatch alarm that sends an Amazon SNS message when the alarm changes state. An alarm watches a single metric over a time period that you specify. The alarm can also perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Amazon EC2 Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms don't invoke actions simply because they are in a particular state. The state must have changed and have been maintained for a specified number of time periods.

You can use the **DB_PERF_INSIGHTS** metric math function in the CloudWatch console to query Amazon RDS for Performance Insights counter metrics. The **DB_PERF_INSIGHTS** function also includes the DBLoad metric at sub-minute intervals. You can set CloudWatch alarms on these metrics.

For more details on how to create an alarm, see Create an alarm on Performance Insights counter metrics from an AWS database.

To set an alarm using the AWS CLI

- Call `put-metric-alarm`. For more information, see *AWS CLI Command Reference*.

To set an alarm using the CloudWatch API

- Call `PutMetricAlarm`. For more information, see *Amazon CloudWatch API Reference*

For more information about setting up Amazon SNS topics and creating alarms, see Using Amazon CloudWatch alarms.

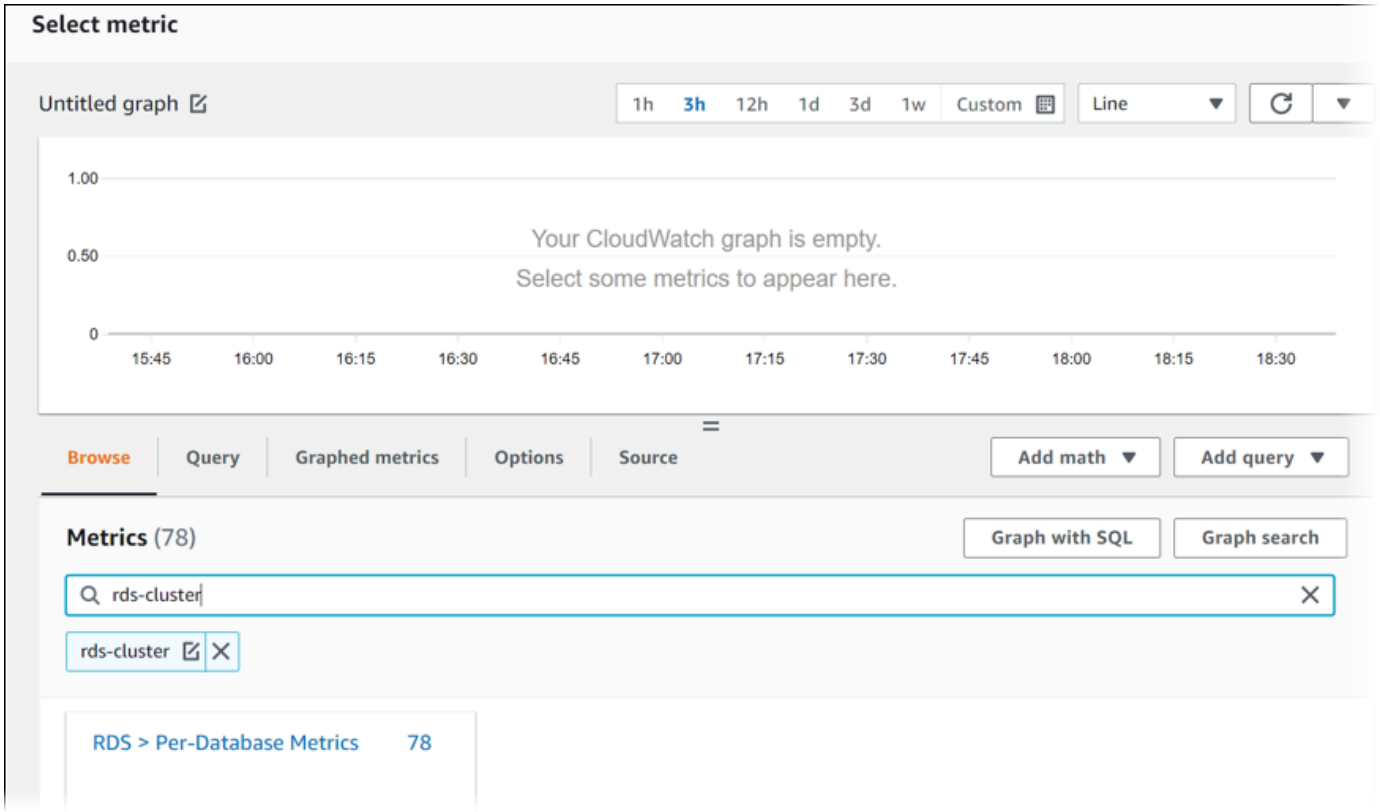
Tutorial: Creating an Amazon CloudWatch alarm for Multi-AZ DB cluster replica lag for Amazon RDS

You can create an Amazon CloudWatch alarm that sends an Amazon SNS message when replica lag for a Multi-AZ DB cluster has exceeded a threshold. An alarm watches the `ReplicaLag` metric over a time period that you specify. The action is a notification sent to an Amazon SNS topic or Amazon EC2 Auto Scaling policy.

To set a CloudWatch alarm for Multi-AZ DB cluster replica lag

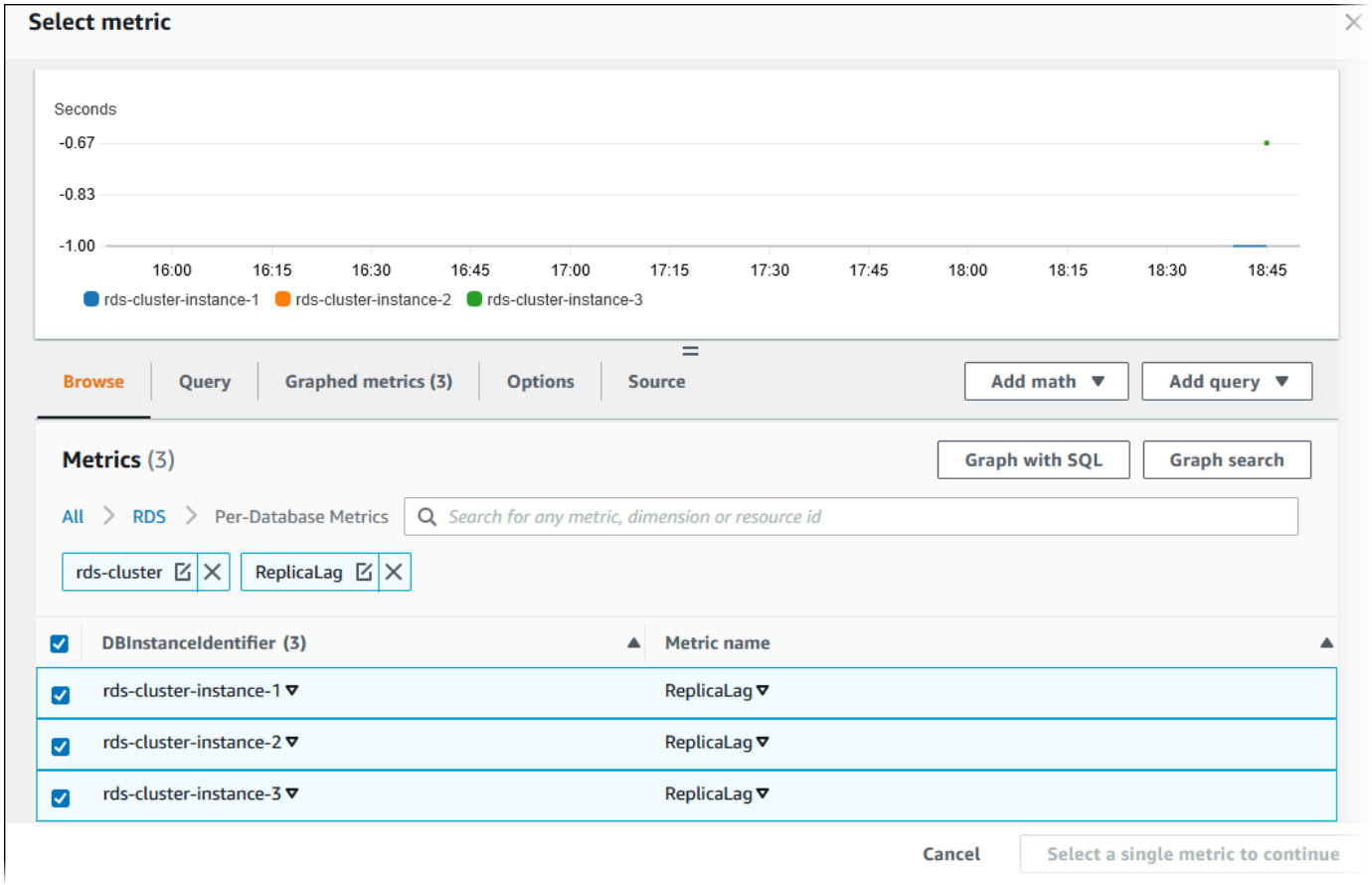
1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Alarms**, **All alarms**.
3. Choose **Create alarm**.
4. On the **Specify metric and conditions** page, choose **Select metric**.
5. In the search box, enter the name of your Multi-AZ DB cluster and press Enter.

   The following image shows the **Select metric** page with a Multi-AZ DB cluster named `rds-cluster` entered.



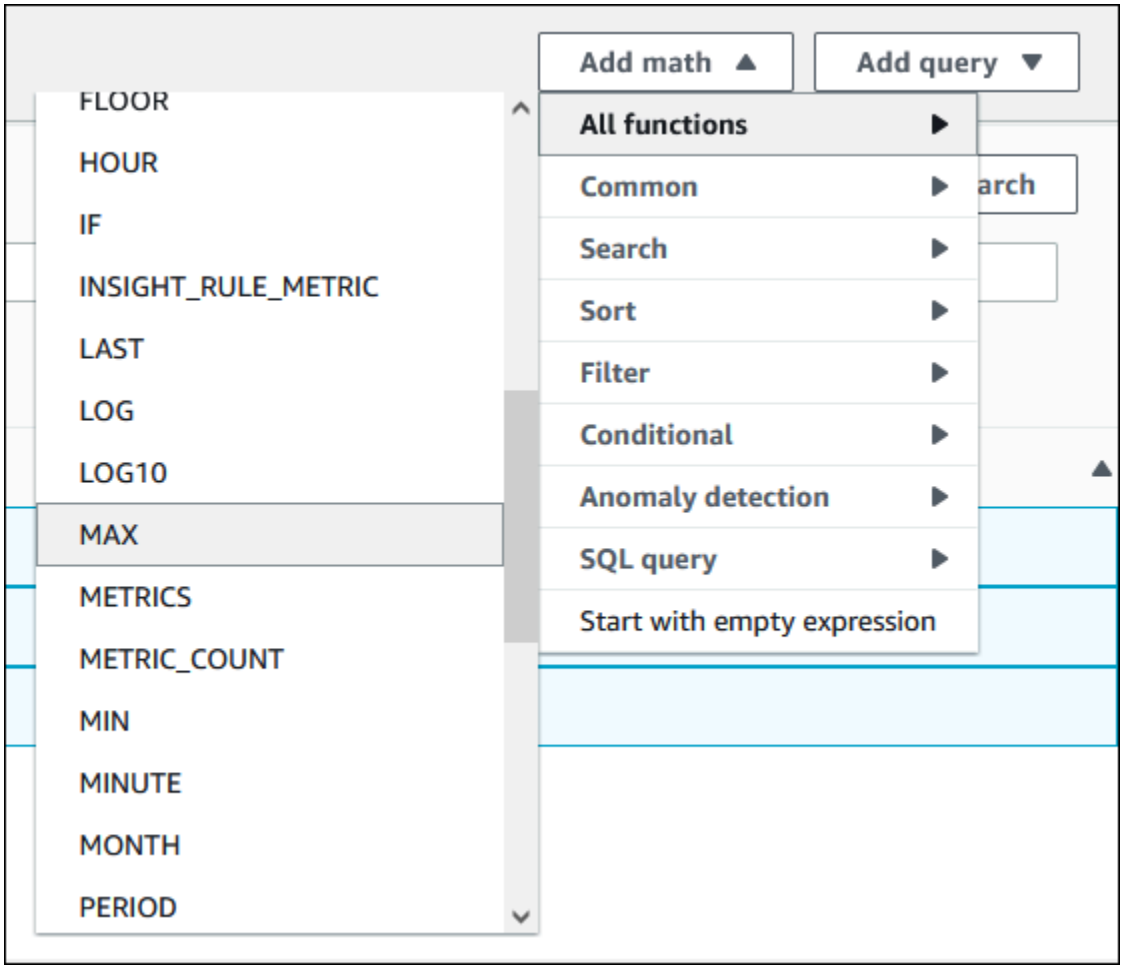
6. Choose **RDS**, **Per-Database Metrics**.
7. In the search box, enter **`ReplicaLag`** and press Enter, then select each DB instance in the DB cluster.

   The following image shows the **Select metric** page with the DB instances selected for the **ReplicaLag** metric.



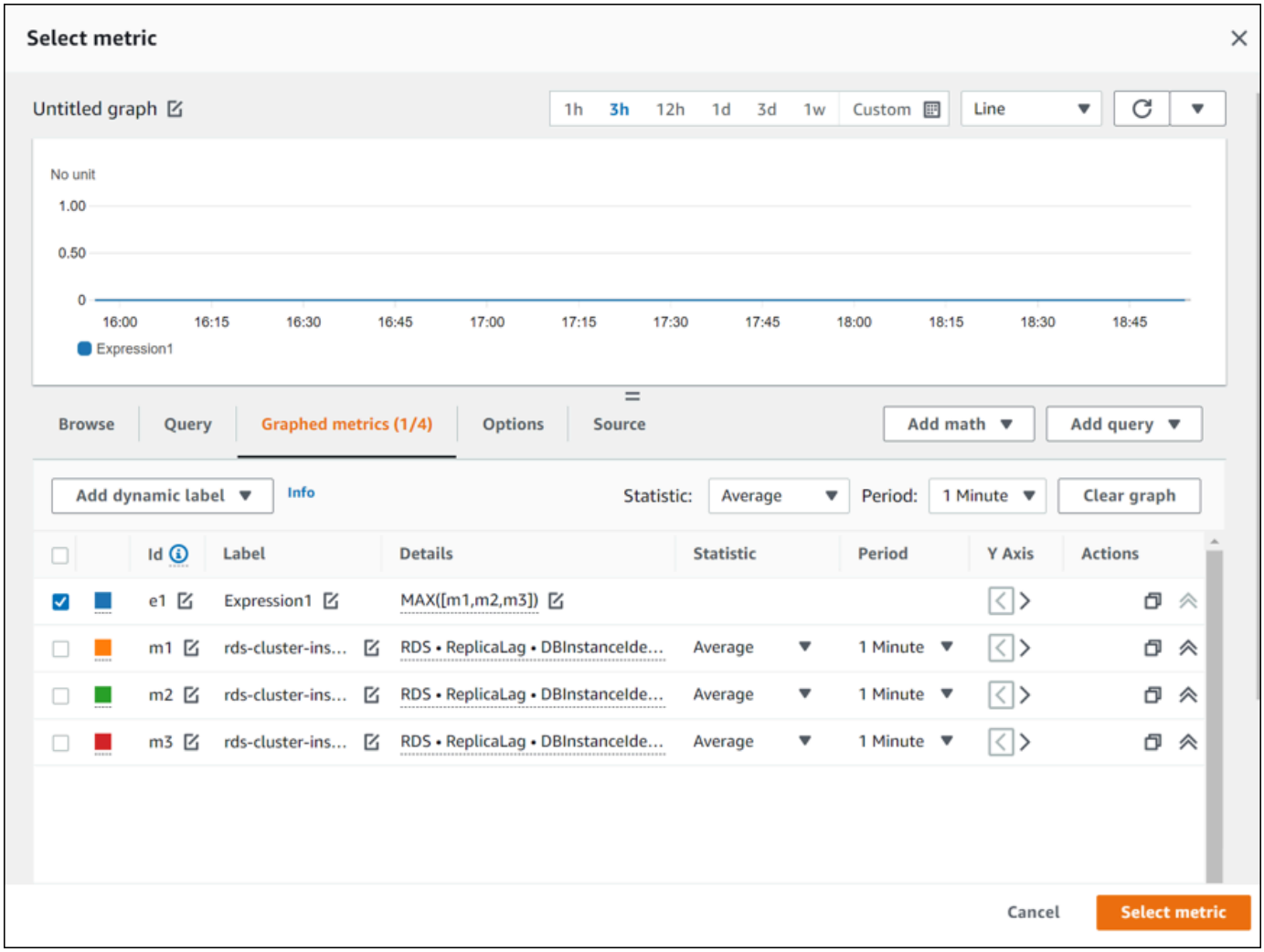
This alarm considers the replica lag for all three of the DB instances in the Multi-AZ DB cluster. The alarm responds when any DB instance exceeds the threshold. It uses a math expression that returns the maximum value of the three metrics. Start by sorting by metric name, and then choose all three **ReplicaLag** metrics.

8. From **Add math**, choose **All functions**, **MAX**.



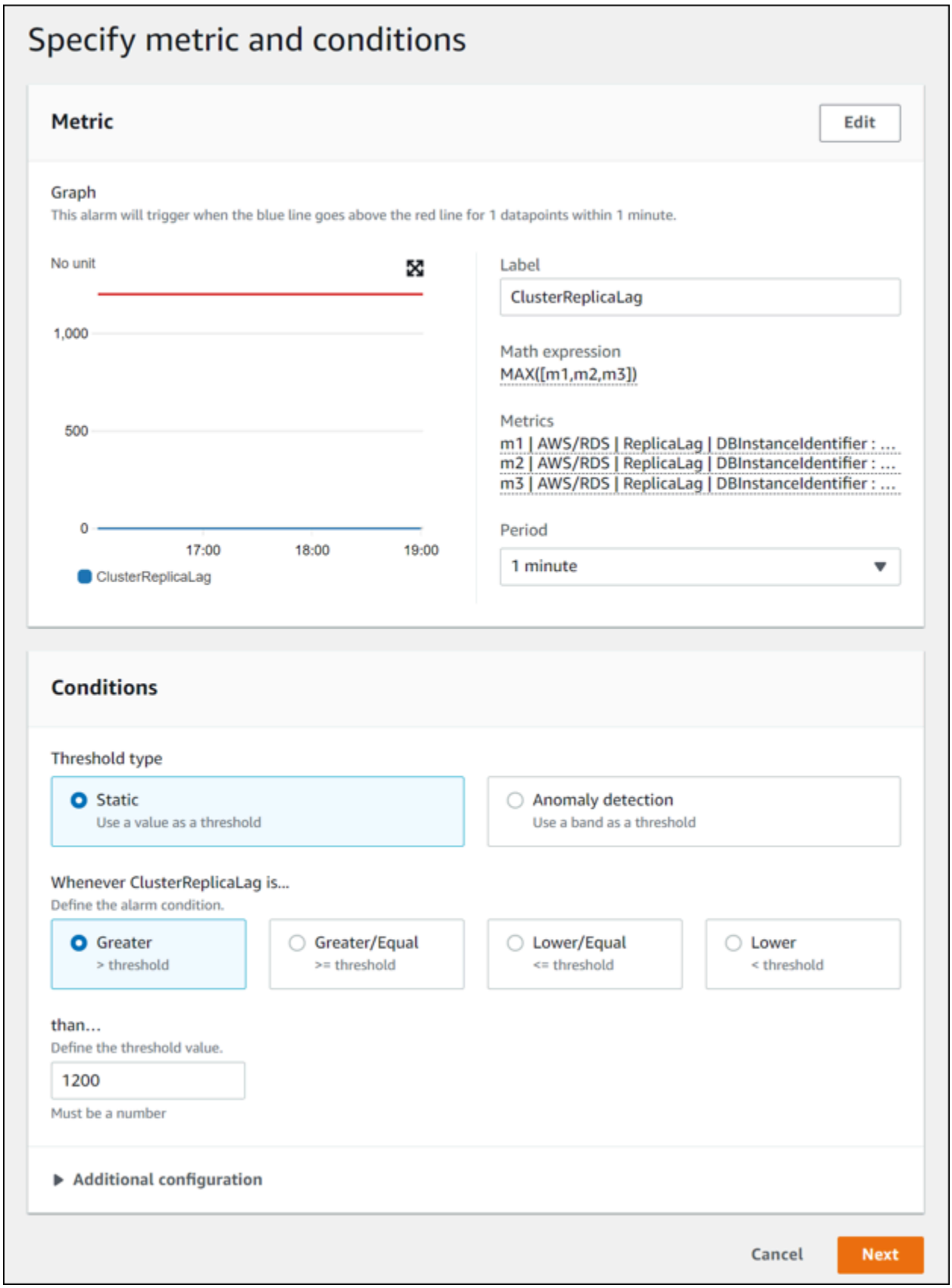
9. Choose the **Graphed metrics** tab, and edit the details for **Expression1** to **`MAX([m1,m2,m3])`**.
10. For all three **ReplicaLag** metrics, change the **Period** to **1 minute**.
11. Clear selection from all metrics except for **Expression1**.

    The **Select metric** page should look similar to the following image.



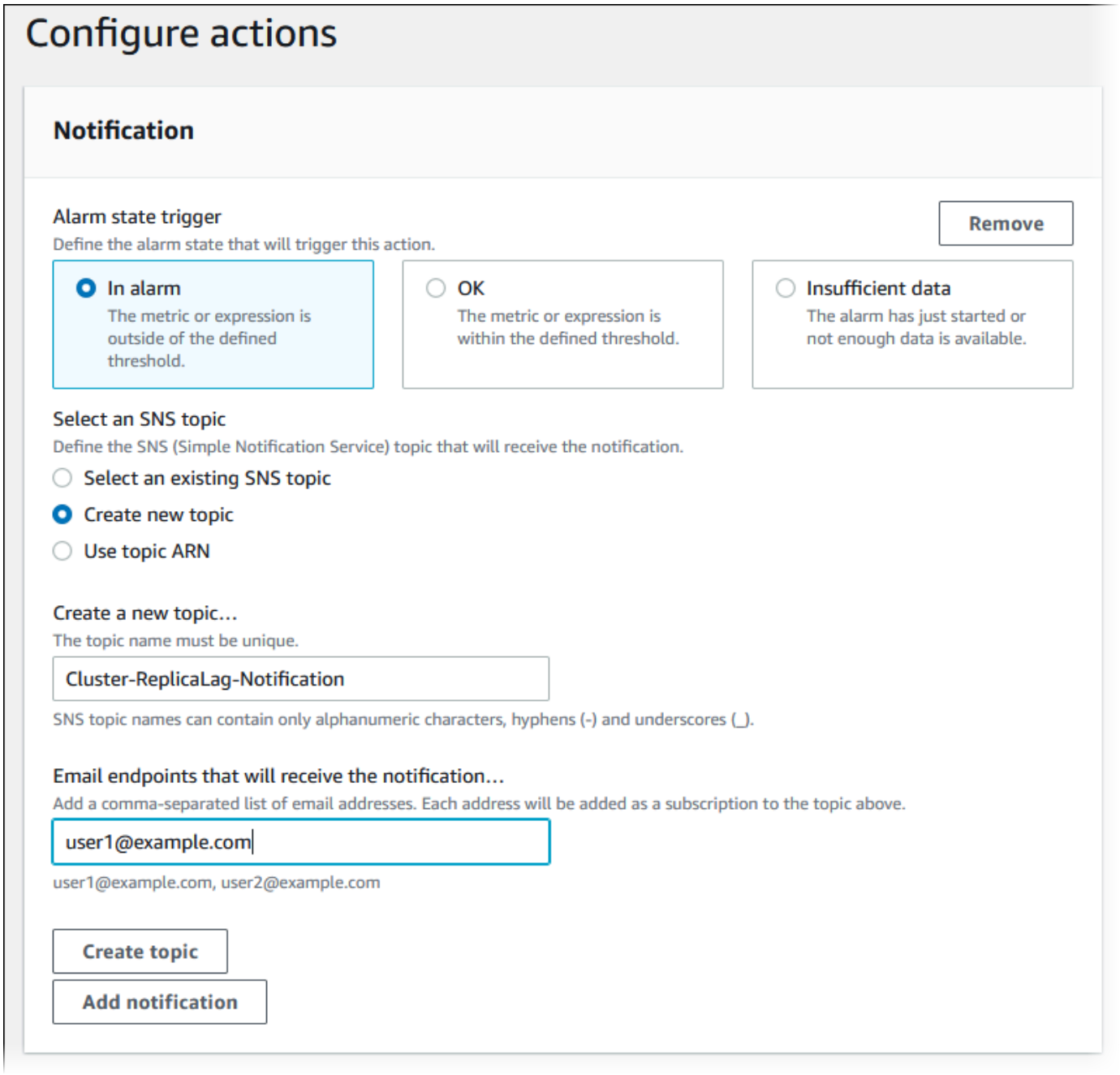
12. Choose **Select metric**.
13. On the **Specify metric and conditions** page, change the label to a meaningful name, such as **`ClusterReplicaLag`**, and enter a number of seconds in **Define the threshold value**. For this tutorial, enter **`1200`** seconds (20 minutes). You can adjust this value for your workload requirements.

    The **Specify metric and conditions** page should look similar to the following image.

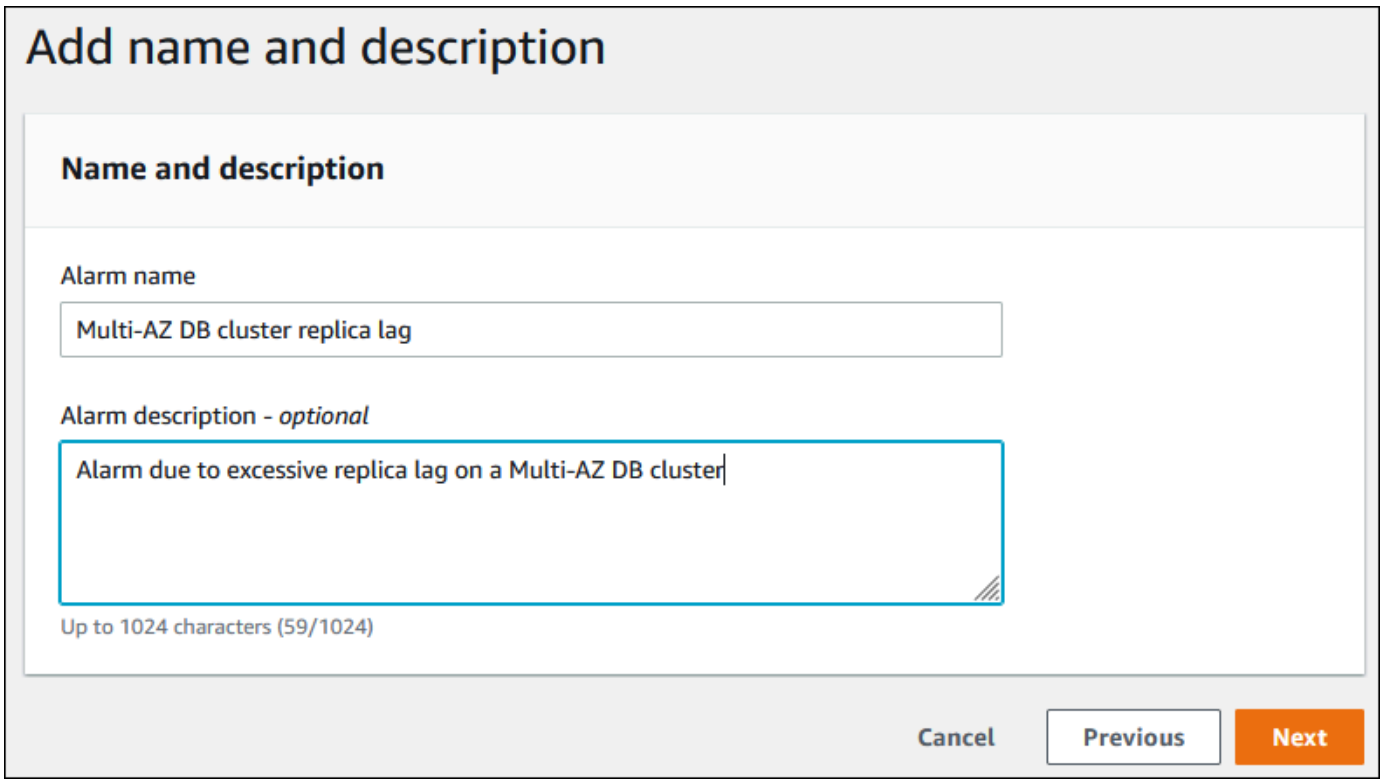


14. Choose **Next**, and the **Configure actions** page appears.

15. Keep **In alarm** selected, choose **Create new topic**, and enter the topic name and a valid email address.



16. Choose **Create topic**, and then choose **Next**.
17. On the **Add name and description** page, enter the **Alarm name** and **Alarm description**, and then choose **Next**.



18. Preview the alarm that you're about to create on the **Preview and create** page, and then choose **Create alarm**.

Monitoring Amazon RDS databases with CloudWatch Database Insights

Monitor the database load (DB Load) for your fleet of Amazon RDS DB instances with Database Insights. DB Load measures the level of session activity in your database. You can use Database Insights to analyze and troubleshoot the performance of your Amazon RDS databases at scale.

With Database Insights, you can visualize the DB Load on your fleet of databases and filter the load by waits, SQL statements, hosts, or users.

By default, RDS enables the Standard mode of Database Insights for your Amazon RDS databases.

For information about using Database Insights in the Amazon CloudWatch console, see CloudWatch Database Insights in the *Amazon CloudWatch User Guide*.

Pricing

For information about pricing, see Amazon CloudWatch Pricing.

Topics

- Amazon RDS DB engine, Region, and instance class support for Database Insights
- Turning on the Advanced mode of Database Insights for Amazon RDS
- Turning on the Standard mode of Database Insights for Amazon RDS
- Configuring your database to monitor slow SQL queries with Database Insights for Amazon RDS
- Considerations for Database Insights for Amazon RDS

Amazon RDS DB engine, Region, and instance class support for Database Insights

The following table provides Amazon RDS DB engines that support Database Insights.

Amazon RDS DB engine	Supported engine versions and Regions	Instance class restrictions
Amazon RDS for MariaDB	For more information on version and Region availability of	Database Insights isn't supported for the following instance classes:

Amazon RDS DB engine	Supported engine versions and Regions	Instance class restrictions
	Database Insights with RDS for MariaDB, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	• db.t2.micro • db.t2.small • db.t3.micro • db.t3.small • db.t4g.micro • db.t4g.small
RDS for MySQL	For more information on version and Region availability of Database Insights with RDS for MySQL, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	Database Insights isn't supported for the following instance classes: • db.t2.micro • db.t2.small • db.t3.micro • db.t3.small • db.t4g.micro • db.t4g.small
Amazon RDS for Microsoft SQL Server	For more information on version and Region availability of Database Insights with RDS for SQL Server, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	N/A

Amazon RDS DB engine	Supported engine versions and Regions	Instance class restrictions
Amazon RDS for PostgreSQL	For more information on version and Region availability of Database Insights with RDS for PostgreSQL, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	N/A
Amazon RDS for Oracle	For more information on version and Region availability of Database Insights with RDS for Oracle, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	N/A

Amazon RDS DB engine, Region, and instance class support for Database Insights features

The following table provides Amazon RDS DB engines that support Database Insights features.

Feature	Pricing tier	Supported regions	Supported DB engines	Supported instance classes
SQL statistics for Performance Insights	All	All	All	All
Analyzing Oracle execution plans using the Performance Insights	All	All	RDS for Oracle	All

Feature	Pricing tier	Supported regions	Supported DB engines	Supported instance classes
dashboard for Amazon RDS				
Analyzing database performance for a period of time	Paid tier only	All	• RDS for MariaDB • RDS for MySQL • RDS for PostgreSQL	All

Feature	Pricing tier	Supported regions	Supported DB engines	Supported instance classes
Viewing Performance Insights proactive recommendations	Paid tier only	• US East (Ohio) • US East (N. Virginia) • US West (N. California) • US West (Oregon) • Asia Pacific (Mumbai) • Asia Pacific (Seoul) • Asia Pacific (Singapore) • Asia Pacific (Sydney) • Asia Pacific (Tokyo) • Canada (Central) • Europe (Frankfurt) • Europe (Ireland) • Europe (London) • Europe (Paris) • Europe (Stockholm) • South America (São Paulo)	All	All

Amazon RDS Region support for Database Insights

Amazon RDS supports Database Insights in the following AWS Regions.

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Hyderabad)
- Asia Pacific (Jakarta)
- Asia Pacific (Malaysia)
- Asia Pacific (Melbourne)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Canada West (Calgary)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Spain)
- Europe (Stockholm)
- Europe (Zurich)
- Israel (Tel Aviv)

- Middle East (Bahrain)
- Middle East (UAE)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Turning on the Advanced mode of Database Insights for Amazon RDS

To turn on the Advanced mode of Database Insights for Amazon RDS, use the following procedures.

Turning on the Advanced mode of Database Insights when creating a DB instance or Multi-AZ DB cluster

Turn on the Advanced mode of Database Insights when creating a database for Amazon RDS.

Console

In the console, you can turn on the Advanced mode of Database Insights when you create a DB instance or Multi-AZ DB cluster.

To turn on the Advanced mode of Database Insights when creating a DB instance or Multi-AZDB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Databases**.
3. Choose **Create database**.
4. In the **Database Insights** section, select **Advanced mode**. Then, choose the following options:

   - **Retention** – The amount of time to retain Performance Insights data. The retention period must be 15 months for the Advanced mode of Database Insights.
   - **AWS KMS key** – Specify your KMS key. Performance Insights encrypts all potentially sensitive data using your KMS key. Data is encrypted in flight and at rest. For more information, see Encrypting Amazon RDS resources.

5. Choose **Create database**.

AWS CLI

To turn on the Advanced mode of Database Insights when creating a DB instance or Multi-AZ DB cluster, call the create-db-instance or create-db-cluster AWS CLI command and supply the following values:

- `--database-insights-mode advanced` to turn on the Advanced mode of Database Insights.
- `--engine` – The database engine for the DB instance.
- `--db-instance-identifier` – The identifier for the DB instance or `--db-cluster-identifier` – The identifier for the Multi-AZ DB cluster.
- `--enable-performance-insights` to turn on Performance Insights for Database Insights.
- `--performance-insights-retention-period` – The retention period for data for your DB instance or Multi-AZ DB cluster. To turn on Database Insights, the retention period must be at least 465 days.

The following example enables the Advanced mode of Database Insights when creating a DB instance.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --database-insights-mode advanced \
    --engine postgresql \
    --db-instance-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 465
```

For Windows:

```
aws rds create-db-instance ^
    --database-insights-mode advanced ^
    --engine postgresql ^
    --db-instance-identifier sample-db-identifier ^
    --enable-performance-insights ^
```

```
    --performance-insights-retention-period 465
```

RDS API

To turn on the Advanced mode of Database Insights when you create a DB instance or Multi-AZ DB cluster, specify the following parameters for your CreateDBInstance or CreateDBCluster Amazon RDS API operation.

- `DatabaseInsightsMode` to `advanced`
- `EnablePerformanceInsights` to `True`
- `PerformanceInsightsRetentionPeriod` to at least 465 days

Turning on the Advanced mode of Database Insights when modifying a DB instance or Multi-AZ DB cluster

Turn on Database Insights when modifying a database for Amazon RDS. Modifying a DB instance to enable the Advanced mode of Database Insights doesn't cause downtime.

> **Note**
>
> To enable Database Insights, each DB instance in a Multi-AZ DB cluster must have the same Performance Insights and Enhanced Monitoring settings.

Console

In the console, you can turn on the Advanced mode of Database Insights when you modify a DB instance or Multi-AZ DB cluster.

To turn on the Advanced mode of Database Insights when modifying a DB instance or Multi-AZDB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Databases**.
3. Choose a DB instance or Multi-AZ DB cluster, and choose **Modify**.
4. In the **Database Insights** section, select **Advanced mode**. Then, choose the following options:

- **Retention** – The amount of time to retain Performance Insights data. The retention period must be 15 months for the Advanced mode of Database Insights.
- **AWS KMS key** – Specify your KMS key. Performance Insights encrypts all potentially sensitive data using your KMS key. Data is encrypted in flight and at rest. For more information, see Encrypting Amazon RDS resources.

5. Choose **Continue**.
6. For **Scheduling of Modifications**, choose **Apply immediately**. If you choose **Apply during the next scheduled maintenance window**, your database ignores this setting and turns on the Advanced mode of Database Insights immediately.
7. Choose **Modify instance**.

AWS CLI

To turn on the Advanced mode of Database Insights when modifying a DB instance or Multi-AZ DB cluster, call the modify-db-instance or modify-db-cluster AWS CLI command and supply the following values:

- `--database-insights-mode advanced` to turn on the Advanced mode of Database Insights.
- `--db-instance-identifier` – The identifier for the DB instance or `--db-cluster-identifier` – The identifier for the Multi-AZ DB cluster.
- `--enable-performance-insights` to turn on Performance Insights for Database Insights.
- `--performance-insights-retention-period` – The retention period for data for your DB instance. To turn on the Advanced mode of Database Insights, the retention period must be at least 465 days.

The following example enables the Advanced mode of Database Insights when modifying a DB instance.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --database-insights-mode advanced \
    --db-instance-identifier sample-db-identifier \
    --enable-performance-insights \
```

```
    --performance-insights-retention-period 465
```

For Windows:

```
aws rds modify-db-instance ^
    --database-insights-mode advanced ^
    --db-instance-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 465
```

RDS API

To turn on the Advanced mode of Database Insights when you modify a DB instance or Multi-AZ DB cluster, specify the following parameters for your ModifyDBInstance or ModifyDBCluster Amazon RDS API operation.

- `DatabaseInsightsMode` to `advanced`
- `EnablePerformanceInsights` to `True`
- `PerformanceInsightsRetentionPeriod` to at least 465 days

Turning on the Standard mode of Database Insights for Amazon RDS

To turn on the Standard mode of Database Insights for Amazon RDS, use the following procedures.

Turning on the Standard mode of Database Insights when creating a DB instance or Multi-AZ DB cluster

Turn on the Standard mode of Database Insights when creating a database for Amazon RDS.

Console

In the console, you can turn on the Standard mode of Database Insights when you create a DB instance or Multi-AZ DB cluster.

To turn on the Standard mode of Database Insights when creating a DB instance or Multi-AZDB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Databases**.

3. Choose **Create database**.
4. In the **Database Insights** section, select **Standard mode**. Then, choose from the following options to turn Performance Insights on or off:

   - To turn off Performance Insights, deselect **Enable Performance Insights**.
   - To turn on Performance Insights, select **Enable Performance Insights**. To configure Performance Insights, specify the following options:
     - **Retention** – The amount of time to retain Performance Insights data. The retention period must be at least 7 days.
     - **AWS KMS key** – Specify your KMS key. Performance Insights encrypts all potentially sensitive data using your KMS key. Data is encrypted in flight and at rest. For more information, see Encrypting Amazon RDS resources.
5. Choose **Create database**.

AWS CLI

To turn on the Standard mode of Database Insights when creating a DB instance or Multi-AZ DB cluster, call the create-db-instance or create-db-cluster AWS CLI command and supply the following values:

- `--database-insights-mode standard` to turn on the Standard mode of Database Insights.
- `--engine` – The database engine for the DB instance.
- `--db-instance-identifier` – The identifier for the DB instance or `--db-cluster-identifier` – The identifier for the Multi-AZ DB cluster.
- `--enable-performance-insights` or `--no-enable-performance-insights` to turn Performance Insights on or off. If you specify `--enable-performance-insights`, you must also specify the `--performance-insights-retention-period` – The retention period for data for your DB instance. The retention period must be at least 7 days.

The following example enables the Standard mode of Database Insights and Performance Insights when creating a DB instance.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
```

```
    --database-insights-mode standard \
    --engine postgresql \
    --db-instance-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

For Windows:

```
aws rds create-db-instance ^
    --database-insights-mode standard ^
    --engine postgresql ^
    --db-instance-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

The following example enables the Standard mode of Database Insights and disables Performance Insights when creating a DB instance.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --database-insights-mode standard \
    --engine postgresql \
    --db-instance-identifier sample-db-identifier \
    --no-enable-performance-insights
```

For Windows:

```
aws rds create-db-instance ^
    --database-insights-mode standard ^
    --engine postgresql ^
    --db-instance-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

To turn on the Standard mode of Database Insights when you create a DB instance or Multi-AZ DB cluster, specify the following parameters for your CreateDBInstance or CreateDBCluster Amazon RDS API operation.

- `DatabaseInsightsMode` to `standard`

- `EnablePerformanceInsights` to `True` or `False`. If you set `EnablePerformanceInsights` to `True`, you must set `PerformanceInsightsRetentionPeriod` to at least 7 days.

Turning on the Standard mode of Database Insights when modifying a DB instance or Multi-AZ DB cluster

Turn on Standard mode of Database Insights when modifying a database for Amazon RDS. Modifying a DB instance to enable the Standard mode of Database Insights doesn't cause downtime.

> **Note**
>
> To enable Database Insights, each DB instance in a Multi-AZ DB cluster must have the same Performance Insights and Enhanced Monitoring settings.

Console

In the console, you can turn on the Standard mode of Database Insights when you modify a DB instance or Multi-AZ DB cluster.

To turn on the Standard mode of Database Insights when modifying a DB instance or Multi-AZDB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Databases**.
3. Choose a DB instance or Multi-AZ DB cluster, and choose **Modify**.
4. In the **Database Insights** section, select **Standard mode**. Then, choose from the following options:
   - To turn off Performance Insights, deselect **Enable Performance Insights**.
   - To turn on Performance Insights, select **Enable Performance Insights**. To configure Performance Insights, specify the following options:
     - **Retention** – The amount of time to retain Performance Insights data. The retention period must be at least 7 days.

- **AWS KMS key** – Specify your KMS key. Performance Insights encrypts all potentially sensitive data using your KMS key. Data is encrypted in flight and at rest. For more information, see Encrypting Amazon RDS resources.

5. Choose **Continue**.
6. For **Scheduling of Modifications**, choose **Apply immediately**. If you choose **Apply during the next scheduled maintenance window**, your database ignores this setting and turns on the Standard mode of Database Insights immediately.
7. Choose **Modify instance**.

AWS CLI

To turn on the Standard mode of Database Insights when modifying a DB instance or Multi-AZ DB cluster, call the modify-db-instance or modify-db-cluster AWS CLI command and supply the following values:

- `--database-insights-mode standard` to turn on the Standard mode of Database Insights.
- `--db-instance-identifier` – The identifier for the DB instance or `--db-cluster-identifier` – The identifier for the Multi-AZ DB cluster.
- `--enable-performance-insights` or `--no-enable-performance-insights` to turn Performance Insights on or off. If you specify `--enable-performance-insights`, you must also specify the `--performance-insights-retention-period` – The retention period for data for your DB instance or Multi-AZ DB cluster. The retention period must be at least 7 days.

The following example enables the Standard mode of Database Insights and enables Performance Insights when modifying a DB instance.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --database-insights-mode standard \
    --db-instance-identifier sample-db-identifier \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

For Windows:

```
aws rds modify-db-instance ^
    --database-insights-mode standard ^
    --db-instance-identifier sample-db-identifier ^
    --enable-performance-insights ^
    --performance-insights-retention-period 7
```

The following example enables the Standard mode of Database Insights and disables Performance Insights when modifying a DB instance.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --database-insights-mode standard \
    --db-instance-identifier sample-db-identifier \
    --no-enable-performance-insights
```

For Windows:

```
aws rds modify-db-instance ^
    --database-insights-mode standard ^
    --db-instance-identifier sample-db-identifier ^
    --no-enable-performance-insights
```

RDS API

To turn on the Standard mode of Database Insights when you modify a DB instance or Multi-AZ DB cluster, specify the following parameters for your ModifyDBInstance or ModifyDBCluster Amazon RDS API operation.

- `DatabaseInsightsMode` to `standard`
- `EnablePerformanceInsights` to `True` or `False`. If you set `EnablePerformanceInsights` to `True`, you must set `PerformanceInsightsRetentionPeriod` to at least 7 days.

Configuring your database to monitor slow SQL queries with Database Insights for Amazon RDS

To monitor slow SQL queries for your database, you can use the **Slow SQL Queries** section in the Database Insights dashboard. Before configuring your database to monitor slow SQL queries, the **Slow SQL Queries** section is blank.

For more information about monitoring slow SQL queries in the Database Insights dashboard, see Viewing the Database Instance Dashboard for CloudWatch Database Insights in the *Amazon CloudWatch User Guide*.

To configure your database to monitor slow SQL queries with Database Insights, complete the following steps:

1. Enable log exports to CloudWatch Logs.
2. Create or modify the DB parameter group for your DB instance.

For information about configuring log exports, see Publishing database logs to Amazon CloudWatch Logs in the *Amazon RDS User Guide*.

To create or modify your DB parameter group, see the following topics.

- Creating a DB parameter group in Amazon RDS
- Modifying parameters in a DB parameter group in Amazon RDS

RDS for MariaDB

To configure your RDS for MariaDB DB instance to monitor slow SQL queries, you can use the following parameter combination as an example:

- `log_slow_query` – set to `1`
- `log_slow_query_time` – set to `1.0`
- `log_output` – set to `FILE`

This is one possible configuration. For a comprehensive guide to MariaDB slow query log parameters and additional configuration options, see the MariaDB documentation for the slow query log.

RDS for MySQL

To configure your RDS for MySQL DB instance to monitor slow SQL queries, you can use the following parameter combination as an example:

- `slow_query_log` – set to `1`
- `long_query_time` – set to `1.0`
- `log_output` – set to `FILE`

This is one possible configuration. For a comprehensive guide to MySQL slow query log parameters and additional configuration options, see the MySQL documentation for the slow query log.

RDS for PostgreSQL

To configure your RDS for PostgreSQL DB instance to monitor slow SQL queries, you can use the following parameter combination as an example. Note that setting these parameters might reduce the performance of your DB instance.

- `log_min_duration_statement` – set to `1000`
- `log_statement` – set to none
- `log_destination` – set to `stderr`

This is one possible configuration. For a comprehensive guide to PostgreSQL logging parameters and additional configuration options, see the PostgreSQL documentation for logging configuration.

Note

For RDS for MySQL, you can configure the parameter `long_query_time` with 1-microsecond granularity. For example, you can set this parameter to `0.000001`. Depending on the amount of queries on the DB instance, the value of the parameter `long_query_time` can reduce performance. Start with the value `1.0`, and adjust it based on your workload. When you set this parameter to `0`, Database Insights logs all queries.

For information about RDS for MariaDB, RDS for MySQL, and RDS for PostgreSQL logs, see the following.

- MariaDB database log files
- MySQL database log files
- RDS for PostgreSQL database log files

Considerations for Database Insights for Amazon RDS

Following are considerations for Database Insights for Amazon RDS.

- You can't manage Database Insights for a DB instance in a Multi-AZ DB cluster.
- To enable the Advanced mode of Database Insights, you must enable Performance Insights and set the Performance Insights retention period to at least 465 days (15 months). There is no additional cost to set the Performance Insights retention period to 15 months besides the cost of Database Insights. For information about pricing for Database Insights, see Amazon CloudWatch Pricing.
- To enable Database Insights, each DB instance in a Multi-AZ DB cluster must have the same Performance Insights and Enhanced Monitoring settings.
- Modifying a DB instance to enable either mode of Database Insights doesn't cause downtime.

Monitoring DB load with Performance Insights on Amazon RDS

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

Performance Insights expands on existing Amazon RDS monitoring features to illustrate and help you analyze your database performance. With the Performance Insights dashboard, you can visualize the database load on your Amazon RDS DB instance load and filter the load by waits, SQL statements, hosts, or users. For information about using Performance Insights with Amazon DocumentDB, see *Amazon DocumentDB Developer Guide*.

Topics

- Overview of Performance Insights on Amazon RDS
- Turning Performance Insights on and off for Amazon RDS
- Overview of the Performance Schema for Performance Insights on Amazon RDS for MariaDB or MySQL

- Configuring access policies for Performance Insights
- Analyzing metrics with the Performance Insights dashboard
- Viewing Performance Insights proactive recommendations
- Retrieving metrics with the Performance Insights API for Amazon RDS
- Logging Performance Insights calls using AWS CloudTrail
- Performance Insights API and interface VPC endpoints (AWS PrivateLink)

Overview of Performance Insights on Amazon RDS

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

By default, RDS enables Performance Insights in the console create wizard for all Amazon RDS engines. If you have more than one database on a DB instance, Performance Insights aggregates performance data.

You can find an overview of Performance Insights for Amazon RDS in the following video.

Using Performance Insights to Analyze Performance of Amazon Aurora PostgreSQL

Important

The following topics describe using Amazon RDS Performance Insights with non-Aurora DB engines. For information about using Amazon RDS Performance Insights with Amazon Aurora, see Using Amazon RDS Performance Insights in the *Amazon Aurora User Guide*.

Topics

- Database load
- Maximum CPU
- Amazon RDS DB engine, Region, and instance class support for Performance Insights
- Pricing and data retention for Performance Insights

Database load

Database load (DB load) measures the level of session activity in your database. `DBLoad` is the key metric in Performance Insights, and Performance Insights collects DB load every second.

Topics

- Active sessions
- Average active sessions
- Average active executions
- Dimensions

Active sessions

A *database session* represents an application's dialogue with a relational database. An *active session* is a connection that has submitted work to the DB engine and is waiting for a response.

A session is active when it's either running on CPU or waiting for a resource to become available so that it can proceed. For example, an active session might wait for a page (or block) to be read into memory, and then consume CPU while it reads data from the page.

Average active sessions

The *average active sessions (AAS)* is the unit for the DBLoad metric in Performance Insights. It measures how many sessions are concurrently active on the database.

Every second, Performance Insights samples the number of sessions concurrently running a query. For each active session, Performance Insights collects the following data:

- SQL statement
- Session state (running on CPU or waiting)
- Host
- User running the SQL

Performance Insights calculates the AAS by dividing the total number of sessions by the number of samples for a specific time period. For example, the following table shows 5 consecutive samples of a running query taken at 1-second intervals.

Sample	Number of sessions running query	AAS	Calculation
1	2	2	2 total sessions / 1 sample
2	0	1	2 total sessions / 2 samples
3	4	2	6 total sessions / 3 samples
4	0	1.5	6 total sessions / 4 samples
5	4	2	10 total sessions / 5 samples

In the preceding example, the DB load for the time interval was 2 AAS. This measurement means that, on average, 2 sessions were active at any given time during the interval when the 5 samples were taken.

Average active executions

The *average active executions (AAE)* per second is related to AAS. To calculate the AAE, Performance Insights divides the total execution time of a query by the time interval. The following table shows the AAE calculation for the same query in the preceding table.

Elapsed time (sec)	Total execution time (sec)	AAE	Calculation
60	120	2	120 execution seconds/60 elapsed seconds
120	120	1	120 execution seconds/120 elapsed seconds
180	380	2.11	380 execution seconds/180 elapsed seconds
240	380	1.58	380 execution seconds/240 elapsed seconds
300	600	2	600 execution seconds/300 elapsed seconds

In most cases, the AAS and AAE for a query are approximately the same. However, because the inputs to the calculations are different data sources, the calculations often vary slightly.

Dimensions

The `db.load` metric is different from the other time-series metrics because you can break it into subcomponents called *dimensions*. You can think of dimensions as "slice by" categories for the different characteristics of the `DBLoad` metric.

When you are diagnosing performance issues, the following dimensions are often the most useful:

Topics

- Wait events
- Top SQL
- Plans

For a complete list of dimensions for the Amazon RDS engines, see DB load sliced by dimensions.

Wait events

A *wait event* causes a SQL statement to wait for a specific event to happen before it can continue running. Wait events are an important dimension, or category, for DB load because they indicate where work is impeded.

Every active session is either running on the CPU or waiting. For example, sessions consume CPU when they search memory for a buffer, perform a calculation, or run procedural code. When sessions aren't consuming CPU, they might be waiting for a memory buffer to become free, a data file to be read, or a log to be written to. The more time that a session waits for resources, the less time it runs on the CPU.

When you tune a database, you often try to find out the resources that sessions are waiting for. For example, two or three wait events might account for 90 percent of DB load. This measure means that, on average, active sessions are spending most of their time waiting for a small number of resources. If you can find out the cause of these waits, you can attempt a solution.

Wait events vary by DB engine:

- For information about all MariaDB and MySQL wait events, see Wait Event Summary Tables in the MySQL documentation.
- For information about all PostgreSQL wait events, see The Statistics Collector > Wait Event tables in the PostgreSQL documentation.
- For information about all Oracle wait events, see Descriptions of Wait Events in the Oracle documentation.
- For information about all SQL Server wait events, see Types of Waits in the SQL Server documentation.

Note

For Oracle, background processes sometimes do work without an associated SQL statement. In these cases, Performance Insights reports the type of background process concatenated with a colon and the wait class associated with that background process. Types of background process include `LGWR`, `ARC0`, `PMON`, and so on.
For example, when the archiver is performing I/O, the Performance Insights report for it is similar to `ARC1:System I/O`. Occasionally, the background process type is also missing, and Performance Insights only reports the wait class, for example `:System I/O`.

Top SQL

Where wait events show bottlenecks, top SQL shows which queries are contributing the most to DB load. For example, many queries might be currently running on the database, but a single query might consume 99 percent of the DB load. In this case, the high load might indicate a problem with the query.

By default, the Performance Insights console displays top SQL queries that are contributing to the database load. The console also shows relevant statistics for each statement. To diagnose performance problems for a specific statement, you can examine its execution plan.

Plans

An *execution plan*, also called simply a *plan*, is a sequence of steps that access data. For example, a plan for joining tables `t1` and `t2` might loop through all rows in `t1` and compare each row to a row in `t2`. In a relational database, an *optimizer* is built-in code that determines the most efficient plan for a SQL query.

For DB instances, Performance Insights collects execution plans automatically. To diagnose SQL performance problems, examine the captured plans for high-resource SQL queries. The plans show how the database has parsed and run queries.

To learn how to analyze DB load using plans, see:

- Oracle: Analyzing Oracle execution plans using the Performance Insights dashboard for Amazon RDS
- SQL Server: Analyzing SQL Server execution plans using the Performance Insights dashboard for Amazon RDS

Plan capture

Every five minutes, Performance Insights identifies the most resource-intensive queries and captures their plans. Thus, you don't need to manually collect and manage a huge number of plans. Instead, you can use the **Top SQL** tab to focus on the plans for the most problematic queries.

> **Note**
>
> Performance Insights doesn't capture plans for queries whose text exceeds the maximum collectable query text limit. For more information, see Accessing more SQL text in the Performance Insights dashboard.

The retention period for execution plans is the same as for your Performance Insights data. The retention setting is **Default (7 days)**. To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see Pricing and data retention for Performance Insights.

Digest queries

The **Top SQL** tab shows digest queries by default. A digest query doesn't itself have a plan, but all queries that use literal values have plans. For example, a digest query might include the text `WHERE `email`=?`. The digest might contain two queries, one with the text `WHERE email=user1@example.com` and another with `WHERE email=user2@example.com`. Each of these literal queries might include multiple plans.

When you select a digest query, the console shows all plans for child statements of the selected digest. Thus, you don't need to look through all the child statements to find the plan. You might see plans that aren't in the displayed list of top 10 child statements. The console shows plans for all child queries for which plans have been collected, regardless of whether the queries are in the top 10.

Maximum CPU

In the dashboard, the **Database load** chart collects, aggregates, and displays session information. To see whether active sessions are exceeding the maximum CPU, look at their relationship to the **Max vCPU** line. Performance Insights determines the **Max vCPU** value by the number of vCPU (virtual CPU) cores for your DB instance.

One process can run on a vCPU at a time. If the number of processes exceeds the number of vCPUs, the processes start queuing. When queuing increases, database performance decreases. If the DB load is often above the **Max vCPU** line, and the primary wait state is CPU, the CPU is overloaded. In this case, you might want to throttle connections to the instance, tune any SQL queries with a high CPU load, or consider a larger instance class. High and consistent instances of any wait state indicate that there might be bottlenecks or resource contention issues to resolve. This can be true even if the DB load doesn't cross the **Max vCPU** line.

Amazon RDS DB engine, Region, and instance class support for Performance Insights

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

The following table provides Amazon RDS DB engines that support Performance Insights.

Note

For Amazon Aurora, see Amazon Aurora DB engine support for Performance Insights in *Amazon Aurora User Guide*.

Amazon RDS DB engine	Supported engine versions and Regions	Instance class restrictions
Amazon RDS for MariaDB	For more information on version and Region availability of Performance Insights with RDS for MariaDB, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	Performance Insights isn't supported for the following instance classes: • db.t2.micro • db.t2.small • db.t3.micro • db.t3.small • db.t4g.micro • db.t4g.small
RDS for MySQL	For more information on version and Region availability of Performance Insights with RDS for MySQL, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	Performance Insights isn't supported for the following instance classes: • db.t2.micro • db.t2.small • db.t3.micro • db.t3.small •

Amazon RDS DB engine	Supported engine versions and Regions	Instance class restrictions
		• db.t4g.micro • db.t4g.small
Amazon RDS for Microsoft SQL Server	For more information on version and Region availability of Performance Insights with RDS for SQL Server, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	N/A
Amazon RDS for PostgreSQL	For more information on version and Region availability of Performance Insights with RDS for PostgreSQL, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	N/A
Amazon RDS for Oracle	For more information on version and Region availability of Performance Insights with RDS for Oracle, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	N/A

Amazon RDS DB engine, Region, and instance class support for Performance Insights features

The following table provides Amazon RDS DB engines that support Performance Insights features.

Feature	Pricing tier	Supported regions	Supported DB engines	Supported instance classes
SQL statistics for Performance Insights	All	All	All	All
Analyzing Oracle execution plans using the Performance Insights dashboard for Amazon RDS	All	All	RDS for Oracle	All
Analyzing database performance for a period of time	Paid tier only	All	RDS for PostgreSQL	All
Viewing Performance Insights proactive recommendations	Paid tier only	• US East (Ohio) • US East (N. Virginia) • US West (N. California) • US West (Oregon) • Asia Pacific (Mumbai) • Asia Pacific (Seoul) • Asia Pacific (Singapore) • Asia Pacific (Sydney)	All	All

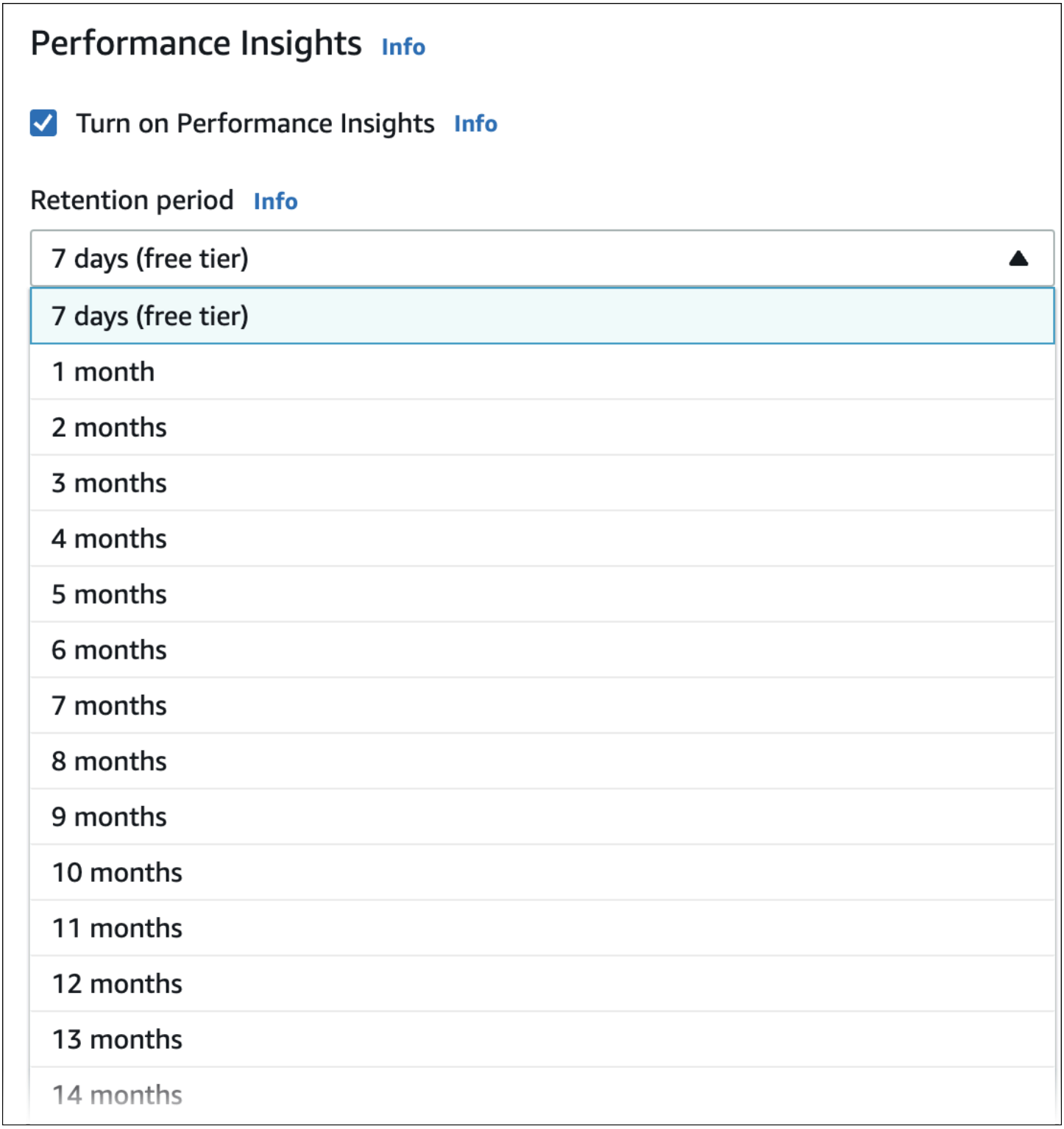
Feature	Pricing tier	Supported regions	Supported DB engines	Supported instance classes
		• Asia Pacific (Tokyo) • Canada (Central) • Europe (Frankfurt) • Europe (Ireland) • Europe (London) • Europe (Paris) • Europe (Stockholm) • South America (São Paulo)		

Pricing and data retention for Performance Insights

By default, Performance Insights includes 7 days of performance data history and 1 million API requests per month. You can also purchase longer retention periods. For complete pricing information, see Performance Insights Pricing.

In the RDS console, you can choose any of the following retention periods for your Performance Insights data:

- **Default (7 days)**
- ***n* months**, where ***n*** is a number from 1–24



To learn how to set a retention period using the AWS CLI, see Turning Performance Insights on and off for Amazon RDS.

Note

Stopping a DB instance or Multi-AZ DB cluster with Performance Insights enabled doesn't affect data retention. While a DB instance or Multi-AZ DB cluster is stopped, Performance Insights won't collect any data.

Turning Performance Insights on and off for Amazon RDS

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

You can turn on Performance Insights for your DB instance or Multi-AZ DB cluster when you create it. If needed, you can turn it off later by modifying your DB instance from the console. Turning Performance Insights on and off doesn't cause downtime, a reboot, or a failover.

> **Note**
>
> Performance Schema is an optional performance tool used by Amazon RDS for MariaDB or MySQL. If you turn Performance Schema on or off, you need to reboot. If you turn Performance Insights on or off, however, you don't need to reboot. For more information, see Overview of the Performance Schema for Performance Insights on Amazon RDS for MariaDB or MySQL.

The Performance Insights agent consumes limited CPU and memory on the DB host. When the DB load is high, the agent limits the performance impact by collecting data less frequently.

Console

In the console, you can turn Performance Insights on or off when you create or modify a DB instance or Multi-AZ DB cluster.

Turning Performance Insights on or off when creating a DB instance or Multi-AZ DB cluster

After creating a new DB instance or Multi-AZ DB cluster, Amazon RDS enables Performance Insights by default. To turn off Performance Insights, choose the option **Database Insights – Standard** and deselect the option **Enable Performance Insights**.

For more information, see the following topics.

- To create a DB instance, follow the instructions for your DB engine in Creating an Amazon RDS DB instance.
- To create a Multi-AZ DB cluster, follow the instructions for your DB engine in Creating a Multi-AZ DB cluster for Amazon RDS.

The following screenshot shows the **Performance Insights** section.



If you choose **Enable Performance Insights**, you have the following options:

- **Retention** (for the Standard mode of Database Insights only) – The amount of time to retain Performance Insights data. The retention setting is **Default (7 days)**. To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see Pricing and data retention for Performance Insights.
- **AWS KMS key** – Specify your AWS KMS key. Performance Insights encrypts all potentially sensitive data using your KMS key. Data is encrypted in flight and at rest. For more information, see Changing an AWS KMS policy for Performance Insights.

Turning Performance Insights on or off when modifying a DB instance or Multi-AZ DB cluster

In the console, you can modify a DB instance or Multi-AZ DB cluster to manage Performance Insights.

To manage Performance Insights for a DB instance or Multi-AZ DB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Databases**.
3. Choose a DB instance or Multi-AZ DB cluster, and choose **Modify**.

4. To turn on Performance Insights, select **Enable Performance Insights**. To turn off Performance Insights, choose the option **Database Insights – Standard** and deselect the option **Enable Performance Insights**.

   If you choose **Enable Performance Insights**, you have the following options:

   - **Retention** (for the Standard mode of Database Insights only) – The amount of time to retain Performance Insights data. The retention setting is **Default (7 days)**. To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see Pricing and data retention for Performance Insights.
   - **AWS KMS key** – Specify your KMS key. Performance Insights encrypts all potentially sensitive data using your KMS key. Data is encrypted in flight and at rest. For more information, see Encrypting Amazon RDS resources.
5. Choose **Continue**.
6. For **Scheduling of Modifications**, choose Apply immediately. If you choose Apply during the next scheduled maintenance window, your instance ignores this setting and turns on Performance Insights immediately.
7. Choose **Modify instance**.

AWS CLI

When you use the create-db-instance AWS CLI command, turn on Performance Insights by specifying `--enable-performance-insights` and set `--database-insights-mode` to either `advanced` or `standard`. To turn off Performance Insights, specify `--no-enable-performance-insights` and set `database-insights-mode` to `standard`.

You can also specify these values using the following AWS CLI commands:

- create-db-cluster
- modify-db-cluster
- create-db-instance-read-replica
- modify-db-instance
- restore-db-instance-from-s3

When you turn on Performance Insights in the CLI, you can optionally specify the number of days to retain Performance Insights data with the `--performance-insights-retention-`

`period` option. You can specify 7, *`month`* * 31 (where *`month`* is a number from 1–23), or 731. For example, if you want to retain your performance data for 3 months, specify 93, which is 3 * 31. The default is 7 days. For more information about retention periods, see Pricing and data retention for Performance Insights.

The following example turns on Performance Insights for `sample-db-cluster` and specifies that Performance Insights data is retained for 93 days (3 months).

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
 --database-insights-mode standard \
    --db-cluster-identifier sample-db-instance \
    --enable-performance-insights \
    --performance-insights-retention-period 93
```

For Windows:

```
aws rds modify-db-cluster ^
 --database-insights-mode standard ^
    --db-cluster-identifier sample-db-instance ^
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

If you specify a retention period such as 94 days, which isn't a valid value, RDS issues an error.

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance
 operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93,
 124, 155, 186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

Note

You can only toggle Performance Insights for an instance in a DB cluster where Performance Insights is not managed at the cluster level.

RDS API

When you create a new DB instance using the CreateDBInstance operation Amazon RDS API operation, turn on Performance Insights by setting `EnablePerformanceInsights` to `True`. To turn off Performance Insights, set `EnablePerformanceInsights` to `False` and set `DatabaseInsightsMode` to `standard`.

You can also specify the `EnablePerformanceInsights` value using the following API operations:

- CreateDBCluster (Multi-AZ DB cluster)
- ModifyDBCluster (Multi-AZ DB cluster)
- ModifyDBInstance
- CreateDBInstanceReadReplica
- RestoreDBInstanceFromS3

When you turn on Performance Insights, you can optionally specify the amount of time, in days, to retain Performance Insights data with the `PerformanceInsightsRetentionPeriod` parameter. You can specify 7, *`month`* * 31 (where *`month`* is a number from 1–23), or 731. For example, if you want to retain your performance data for 3 months, specify 93, which is 3 * 31. The default is 7 days. For more information about retention periods, see Pricing and data retention for Performance Insights.

Overview of the Performance Schema for Performance Insights on Amazon RDS for MariaDB or MySQL

The Performance Schema is an optional feature for monitoring Amazon RDS for MariaDB or MySQL runtime performance at a low level of detail. The Performance Schema is designed to have minimal impact on database performance. Performance Insights is a separate feature that you can use with or without the Performance Schema.

Topics

- Overview of the Performance Schema
- Performance Insights and the Performance Schema
- Automatic management of the Performance Schema by Performance Insights

- Effect of a reboot on the Performance Schema
- Determining whether Performance Insights is managing the Performance Schema
- Turn on the Performance Schema for Amazon RDS for MariaDB or MySQL

Overview of the Performance Schema

The Performance Schema monitors events in MariaDB and MySQL databases. An *event* is a database server action that consumes time and has been instrumented so that timing information can be collected. Examples of events include the following:

- Function calls
- Waits for the operating system
- Stages of SQL execution
- Groups of SQL statements

The `PERFORMANCE_SCHEMA` storage engine is a mechanism for implementing the Performance Schema feature. This engine collects event data using instrumentation in the database source code. The engine stores events in memory-only tables in the `performance_schema` database. You can query `performance_schema` just as you can query any other tables. For more information, see MySQL Performance Schema in the *MySQL Reference Manual*.

Performance Insights and the Performance Schema

Performance Insights and the Performance Schema are separate features, but they are connected. The behavior of Performance Insights for Amazon RDS for MariaDB or MySQL depends on whether the Performance Schema is turned on, and if so, whether Performance Insights manages the Performance Schema automatically. The following table describes the behavior.

Performance Schema turned on	Performance Insights management mode	Performance Insights behavior
Yes	Automatic	• Collects detailed, low-level monitoring information •

Performance Schema turned on	Performance Insights management mode	Performance Insights behavior
		• Collects active session metrics every second • Displays DB load categorized by detailed wait events, which you can use to identify bottlenecks
Yes	Manual	• Collects wait events and per-SQL metrics • Collects active session metrics every second • Reports user states such as inserting and sending, which don't help you identify bottlenecks
No	N/A	• Doesn't collect wait events, per-SQL metrics, or other detailed, low-level monitoring information • Collects active session metrics every five seconds instead of every second • Reports user states such as inserting and sending, which don't help you identify bottlenecks

Automatic management of the Performance Schema by Performance Insights

When you create an Amazon RDS for MariaDB or MySQL DB instance with Performance Insights turned on, the Performance Schema is also turned on. In this case, Performance Insights automatically manages your Performance Schema parameters. This is the recommended configuration.

When Performance Insights manages the Performance Schema automatically, the **Source** of `performance_schema` is `System default`.

> **Note**
>
> Automatic management of the Performance Schema isn't supported for the t4g.medium instance class.

If you change the `performance_schema` parameter value manually, and then later want to change to automatic management, see Turn on the Performance Schema for Amazon RDS for MariaDB or MySQL.

> **Important**
>
> When Performance Insights turns on the Performance Schema, it doesn't change the parameter group values. However, the values are changed on the DB instances that are running. The only way to see the changed values is to run the `SHOW GLOBAL VARIABLES` command.

Effect of a reboot on the Performance Schema

Performance Insights and the Performance Schema differ in their requirements for DB instance reboots:

Performance Schema

To turn this feature on or off, you must reboot the DB instance.

Performance Insights

To turn this feature on or off, you don't need to reboot the DB instance.

If the Performance Schema isn't currently turned on, and you turn on Performance Insights without rebooting the DB instance, the Performance Schema won't be turned on.

Determining whether Performance Insights is managing the Performance Schema

To find out whether Performance Insights is currently managing the Performance Schema for all supported major engine versions, review the following table.

Setting of performan ce_schema parameter	Setting of the Source column	Performance Insights is managing the Performance Schema?
`0`	`System default`	Yes
`0` or `1`	`Modified`	No

In the following procedure, you determine whether Performance Insights is managing the Performance Schema automatically.

To determine whether Performance Insights is managing the Performance Schema automatically

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Parameter groups**.
3. Select the name of the parameter group for your DB instance.
4. Enter **`performance_schema`** in the search bar.
5. Check whether **Source** is the system default and **Value** is **0**. If so, Performance Insights is managing the Performance Schema automatically.

   In the example shown here, Performance Insights isn't managing the Performance Schema automatically.



Turn on the Performance Schema for Amazon RDS for MariaDB or MySQL

Assume that Performance Insights is turned on for your DB instance or Multi-AZ DB cluster but isn't currently managing the Performance Schema. If you want to allow Performance Insights to manage the Performance Schema automatically, complete the following steps.

To configure the Performance Schema for automatic management

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Parameter groups**.
3. Select the name of the parameter group for your DB instance or Multi-AZ DB cluster.
4. Choose **Edit**.
5. Enter **`performance_schema`** in the search bar.
6. Select the `performance_schema` parameter.
7. Choose **Set to default value**.
8. Confirm by choosing **Set values to default**.
9. Choose **Save Changes**.
10. Reboot the DB instance or Multi-AZ DB cluster.

> **Important**
>
> Whenever you turn the Performance Schema on or off, make sure to reboot the DB instance or Multi-AZ DB cluster.

For more information about modifying instance parameters, see Modifying parameters in a DB parameter group in Amazon RDS. For more information about the dashboard, see Analyzing metrics with the Performance Insights dashboard. For more information about the MySQL performance schema, see MySQL Performance Schema (for 8.0) and MySQL Performance Schema (for 8.4) in the MySQL documentation.

Configuring access policies for Performance Insights

To access Performance Insights, a principal must have the appropriate permissions from AWS Identity and Access Management (IAM).

> **Note**
>
> To use Performance Insights with a customer-managed key, grant users the `kms:Decrypt` and `kms:GenerateDataKey` permissions for your AWS AWS KMS key.

Access Performance Insights using these methods:

- Attach the `AmazonRDSPerformanceInsightsReadOnly` managed policy for read-only access
- Attach the `AmazonRDSPerformanceInsightsFullAccess` managed policy for access to all operations of the Performance Insights API
- Create a custom IAM policy with specific permissions
- Configure AWS KMS permissions for encrypted Performance Insights data
- Set up fine-grained access using resource-level permissions
- Use tag-based access control to manage permissions through resource tags

Attaching the AmazonRDSPerformanceInsightsReadOnly policy to an IAM principal

`AmazonRDSPerformanceInsightsReadOnly` is an AWS managed policy that grants access to all read-only operations of the Amazon RDS Performance Insights API.

If you attach `AmazonRDSPerformanceInsightsReadOnly` to a permission set or role, you must also attach the following CloudWatch permissions:

- `GetMetricStatistics`
- `ListMetrics`
- `GetMetricData`

With these permissions, the recipient can use Performance Insights with other console features.

For more information about CloudWatch permissions, see Amazon CloudWatch permissions reference.

For more information about `AmazonRDSPerformanceInsightsReadOnly`, see AWS managed policy: AmazonRDSPerformanceInsightsReadOnly.

Attaching the AmazonRDSPerformanceInsightsFullAccess policy to an IAM principal

`AmazonRDSPerformanceInsightsFullAccess` is an AWS managed policy that grants access to all operations of the Amazon RDS Performance Insights API.

If you attach `AmazonRDSPerformanceInsightsFullAccess` to a permission set or role, you must also attach the following CloudWatch permissions:

- `GetMetricStatistics`
- `ListMetrics`
- `GetMetricData`

With these permissions, the recipient can use Performance Insights with other console features.

For more information about CloudWatch permissions, see Amazon CloudWatch permissions reference.

For more information about `AmazonRDSPerformanceInsightsFullAccess`, see AWS managed policy: AmazonRDSPerformanceInsightsFullAccess.

Creating a custom IAM policy for Performance Insights

For users who don't have either the `AmazonRDSPerformanceInsightsReadOnly` or `AmazonRDSPerformanceInsightsFullAccess` policy, you can grant access to Performance Insights by creating or modifying a user-managed IAM policy. When you attach the policy to an IAM permission set or role, the recipient can use Performance Insights.

To create a custom policy

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Create Policy** page, choose the **JSON** option.
5. Copy and paste the text provided in the *JSON policy document* section in the *AWS Managed Policy Reference Guide* for AmazonRDSPerformanceInsightsReadOnly or AmazonRDSPerformanceInsightsFullAccess policy.
6. Choose **Review policy**.
7. Provide a name for the policy and optionally a description, and then choose **Create policy**.

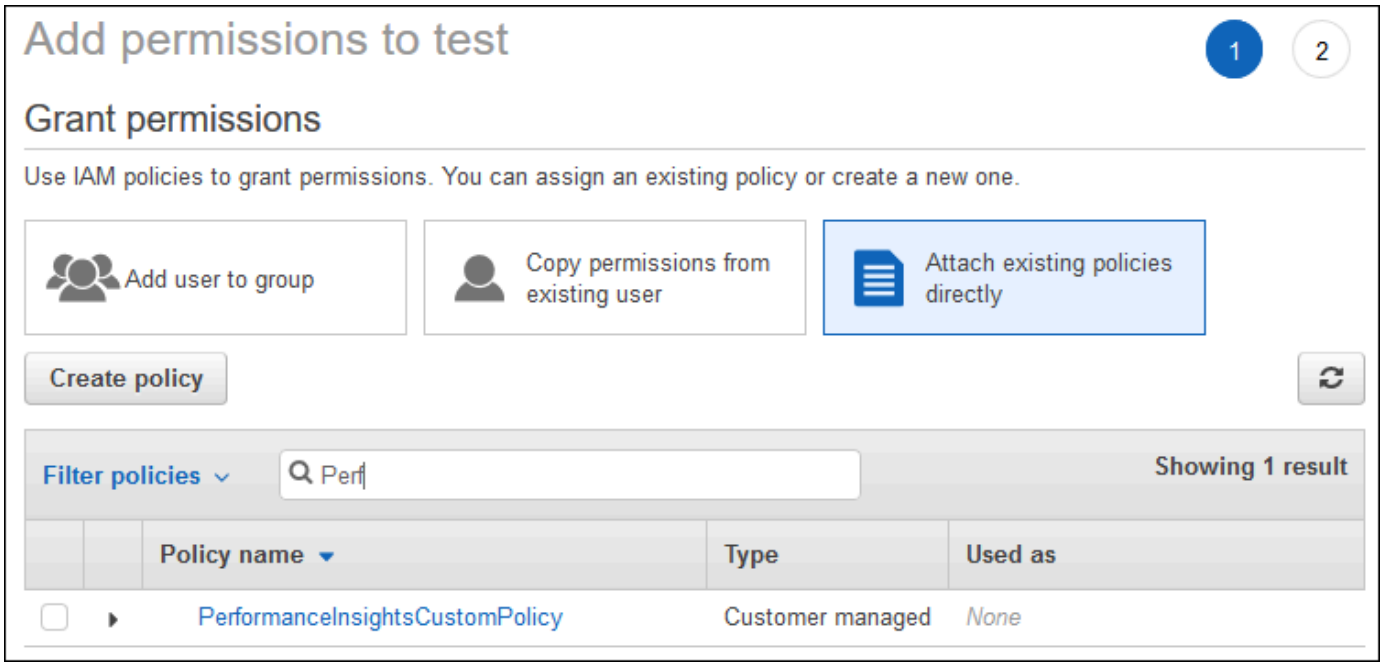
You can now attach the policy to a permission set or role. The following procedure assumes that you already have a user available for this purpose.

To attach the policy to a user

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose an existing user from the list.

> **Important**
>
> To use Performance Insights, make sure that you have access to Amazon RDS in addition to the custom policy. For example, the `AmazonRDSPerformanceInsightsReadOnly` predefined policy provides read-only access to Amazon RDS. For more information, see Managing access using policies.

4. On the **Summary** page, choose **Add permissions**.
5. Choose **Attach existing policies directly**. For **Search**, type the first few characters of your policy name, as shown in the following image.



6. Choose your policy, and then choose **Next: Review**.
7. Choose **Add permissions**.

Changing an AWS KMS policy for Performance Insights

Performance Insights uses an AWS KMS key to encrypt sensitive data. When you enable Performance Insights through the API or the console, you can do either of the following:

- Choose the default AWS managed key.

  Amazon RDS uses the AWS managed key for your new DB instance. Amazon RDS creates an AWS managed key for your AWS account. Your AWS account has a different AWS managed key for Amazon RDS for each AWS Region.
- Choose a customer managed key.

  If you specify a customer managed key, users in your account that call the Performance Insights API need the `kms:Decrypt` and `kms:GenerateDataKey` permissions on the KMS key. You can configure these permissions through IAM policies. However, we recommend that you manage these permissions through your KMS key policy. For more information, see Key policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Example

The following example shows how to add statements to your KMS key policy. These statements allow access to Performance Insights. Depending on how you use the KMS key, you might want to change some restrictions. Before adding statements to your policy, remove all comments.

JSON

```
{
    "Version":"2012-10-17",
    "Id" : "your-policy",
    "Statement" : [
        {
            "Sid" : "AllowViewingRDSPerformanceInsights",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:role/Role1"
                ]
                },
             "Action": [
                "kms:Decrypt",
```

```
                "kms:GenerateDataKey"
                ],
            "Resource": "*",
            "Condition" : {
            "StringEquals" : {
                "kms:ViaService" : "rds.us-east-1.amazonaws.com"
                },
            "ForAnyValue:StringEquals": {
                "kms:EncryptionContext:aws:pi:service": "rds",
                "kms:EncryptionContext:service": "pi",
                "kms:EncryptionContext:aws:rds:db-id": "db-
AAAAABBBBBCCCCDDDDDEEEEE"
                }
            }
        }
    ]
}
```

How Performance Insights uses AWS KMS customer managed key

Performance Insights uses customer managed keys to encrypt sensitive data. When you turn on Performance Insights, you can provide an AWS KMS key through the API. Performance Insights creates AWS KMS permissions on this key. It uses the key and performs the necessary operations to process sensitive data. Sensitive data includes fields such as user, database, application, and SQL query text. Performance Insights ensures that the data remains encrypted both at rest and in-flight.

How Performance Insights IAM works with AWS KMS

IAM gives permissions to specific APIs. Performance Insights has the following public APIs, which you can restrict using IAM policies:

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetadata`
- `GetResourceMetrics`
- `ListAvailableResourceDimensions`
- `ListAvailableResourceMetrics`

You can use the following API requests to get sensitive data.

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetrics`

When you use the API to get sensitive data, Performance Insights leverages the caller's credentials. This check ensures that access to sensitive data is limited to those with access to the KMS key.

When calling these APIs, you need permissions to call the API through the IAM policy and permissions to invoke the `kms:decrypt` action through the AWS KMS key policy.

The `GetResourceMetrics` API can return both sensitive and non-sensitive data. The request parameters determine whether the response should include sensitive data. The API returns sensitive data when the request includes a sensitive dimension in either the filter or group-by parameters.

For more information about the dimensions that you can use with the `GetResourceMetrics` API, see DimensionGroup.

Examples

The following example requests the sensitive data for the `db.user` group:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
  "ServiceType": "RDS",
  "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
  "MetricQueries": [
    {
```

```
      "Metric": "db.load.avg",
      "GroupBy": {
        "Group": "db.user",
        "Limit": 2
      }
    }
  ],
  "StartTime": 1693872000,
  "EndTime": 1694044800,
  "PeriodInSeconds": 86400
}
```

Example

The following example requests the non-sensitive data for the `db.load.avg` metric:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "MetricQueries": [
        {
            "Metric": "db.load.avg"
        }
    ],
    "StartTime": 1693872000,
    "EndTime": 1694044800,
    "PeriodInSeconds": 86400
}
```

Granting fine-grained access for Performance Insights

Fine-grained access control offers additional ways of controlling access to Performance Insights. This access control can allow or deny access to individual dimensions for `GetResourceMetrics`, `DescribeDimensionKeys`, and `GetDimensionKeyDetails` Performance Insights actions. To use fine-grained access, specify dimensions in the IAM policy by using condition keys. The evaluation of the access follows the IAM policy evaluation logic. For more information, see Policy evaluation logic in the *IAM User Guide*. If the IAM policy statement doesn't specify any dimension, then the statement controls access to all the dimensions for the specified action. For the list of available dimensions, see DimensionGroup.

To find out the dimensions that your credentials are authorized to access, use the `AuthorizedActions` parameter in `ListAvailableResourceDimensions` and specify the action. The allowed values for `AuthorizedActions` are as follows:

- `GetResourceMetrics`
- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`

For example, if you specify `GetResourceMetrics` to the `AuthorizedActions` parameter, `ListAvailableResourceDimensions` returns the list of dimensions that the `GetResourceMetrics` action is authorized to access. If you specify multiple actions in the `AuthorizedActions` parameter, then `ListAvailableResourceDimensions` returns an intersection of dimensions that those actions are authorized to access.

Example

The following example provides access to the specified dimensions for `GetResourceMetrics` and `DescribeDimensionKeys` actions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
```

```
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },
        {
            "Sid": "SingleAllow",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.id",
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }


    ]
}
```

The following is the response for the requested dimension:

```
 // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
```

```
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                  //  { "Identifier": "db.sql_tokenized.db_id" }, // not included
 because not allows in the IAM Policy
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            }

        ] }
    ]
}
```

The following example specifies one allow and two deny access for the dimensions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
          {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
          },

          {
            "Sid": "O01AllowAllWithoutSpecifyingDimensions",
```

```
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },

        {
            "Sid": "O01DenyAppDimensionForAll",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.application.name"
                    ]
                }
            }
        },

        {
            "Sid": "O01DenySQLForGetResourceMetrics",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
```

```
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }
    ]
}
```

The following are the responses for the requested dimensions:

```
   // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["GetResourceMetrics"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.sql_tokenized.statement" }
```

```
                ]
            },
            ...
        ] }
    ]
}
```

```
// ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [
                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // allowed for DescribeDimensionKeys because our IAM Policy
                  // denies it only for GetResourceMetrics
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
        ] }
    ]
```

```
}
```

Using tag-based access control for Performance Insights

You can control access to Performance Insights metrics using tags inherited from the parent DB instance. To control access to Performance Insights operations, use IAM policies. These policies can check the tags on your DB instance to determine permissions.

How tags work with Performance Insights

Performance Insights automatically applies your DB instance tags to authorize Performance Insights metrics. When you add tags to your DB instance, you can immediately use those tags to control access to Performance Insights data.

- To add or update tags for Performance Insights metrics, modify the tags on your DB instance.
- To view tags for Performance Insights metrics, call `ListTagsForResource` on the Performance Insights metric resource. It will return the tags from the DB instance associated with the metric.

> **Note**
>
> The `TagResource` and `UntagResource` operations return an error if you try to use them directly on Performance Insights metrics.

Creating tag-based IAM policies

To control access to Performance Insights operations, use the `aws:ResourceTag` condition key in your IAM policies. These policies check the tags on yourDB instance.

Example

This policy prevents access to Performance Insights metrics for production databases. The policy denies the `pi:GetResourceMetrics` operation in Performance Insights for any database resource tagged with `env:prod`.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Deny",
```

```
            "Action": "pi:GetResourceMetrics",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/env": "prod"
                }
            }
        }
    ]
}
```

Analyzing metrics with the Performance Insights dashboard

Important

AWS has announced the end-of-life date for Performance Insights: June 30, 2026. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. The Performance Insights API will continue to exist with no pricing changes. Costs for the Performance Insights API will appear in your AWS bill with the cost of CloudWatch Database Insights.
We recommend that you upgrade any DB instances using the paid tier of Performance Insights to the Advanced mode of Database Insights before June 30, 2026. For information about upgrading to the Advanced mode of Database Insights, see Turning on the Advanced mode of Database Insights for Amazon RDS.
If you take no action, DB instances using Performance Insights will default to using the Standard mode of Database Insights. With Standard mode of Database Insights, you might lose access to performance data history beyond 7 days and might not be able to use execution plans and on-demand analysis features in the Amazon RDS console. After June 30, 2026 only the Advanced mode of Database Insights will support execution plans and on-demand analysis.
With CloudWatch Database Insights, you can monitor database load for your fleet of databases and analyze and troubleshoot performance at scale. For more information about Database Insights, see Monitoring Amazon RDS databases with CloudWatch Database Insights. For pricing information, see Amazon CloudWatch Pricing.

The Performance Insights dashboard contains database performance information to help you analyze and troubleshoot performance issues. On the main dashboard page, you can view

information about the database load. You can "slice" DB load by dimensions such as wait events or SQL.

Performance Insights dashboard

- Overview of the Performance Insights dashboard
- Accessing the Performance Insights dashboard
- Analyzing DB load by wait events
- Analyzing database performance for a period of time
- Analyzing queries with the Top SQL tab in Performance Insights
- Analyzing top Oracle PDB load
- Analyzing execution plans using the Performance Insights dashboard for Amazon RDS

Overview of the Performance Insights dashboard

The dashboard is the easiest way to interact with Performance Insights. The following example shows the dashboard for a PostgreSQL DB instance.

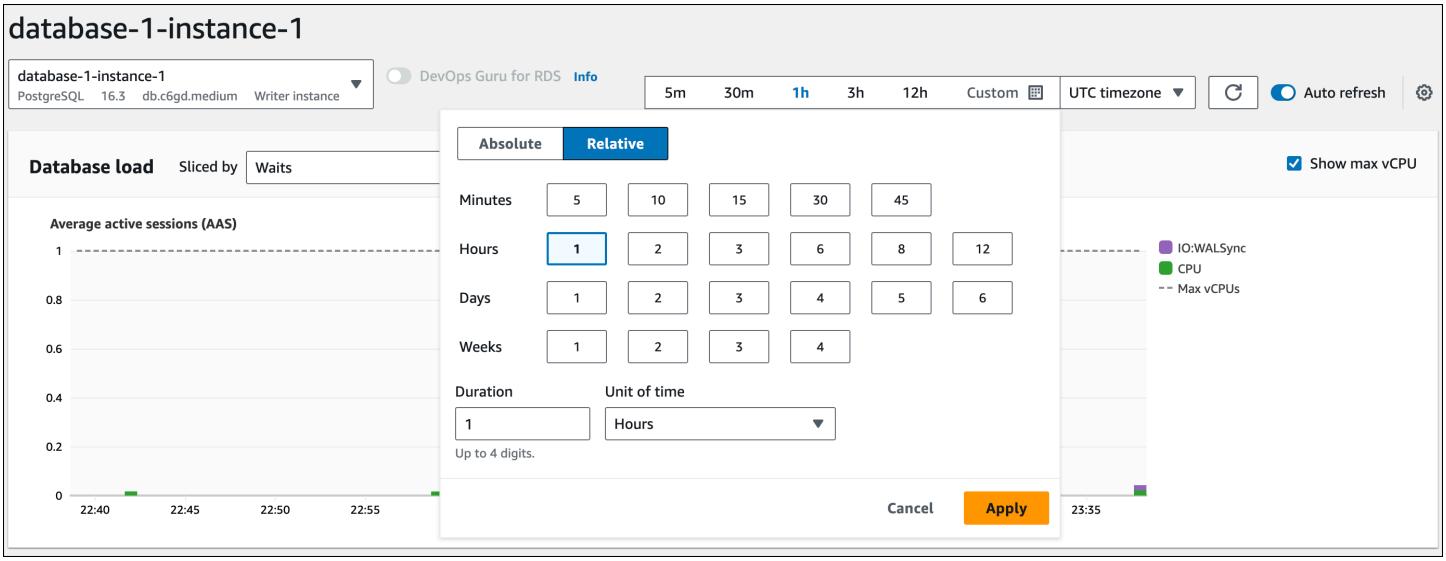


Topics

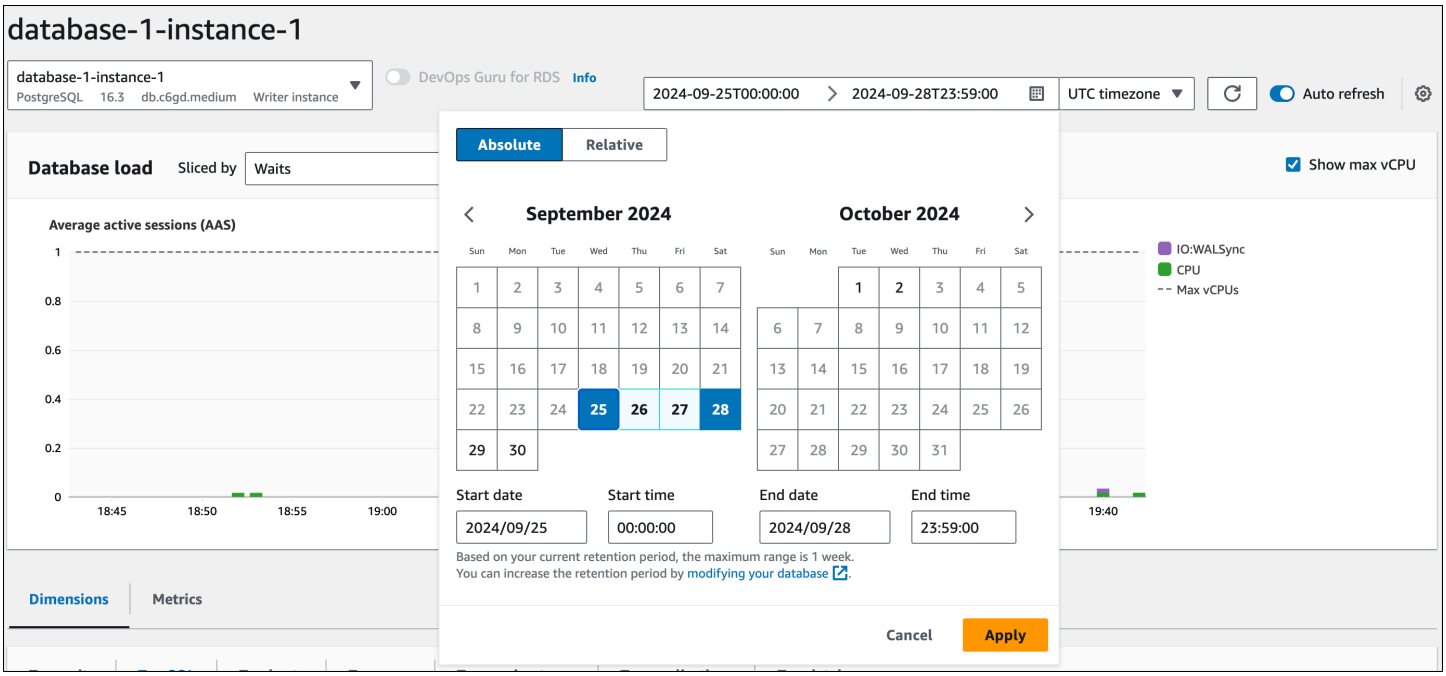
- Time range filter
- Counter metrics chart
- Database load chart
- Top dimensions table

Time range filter

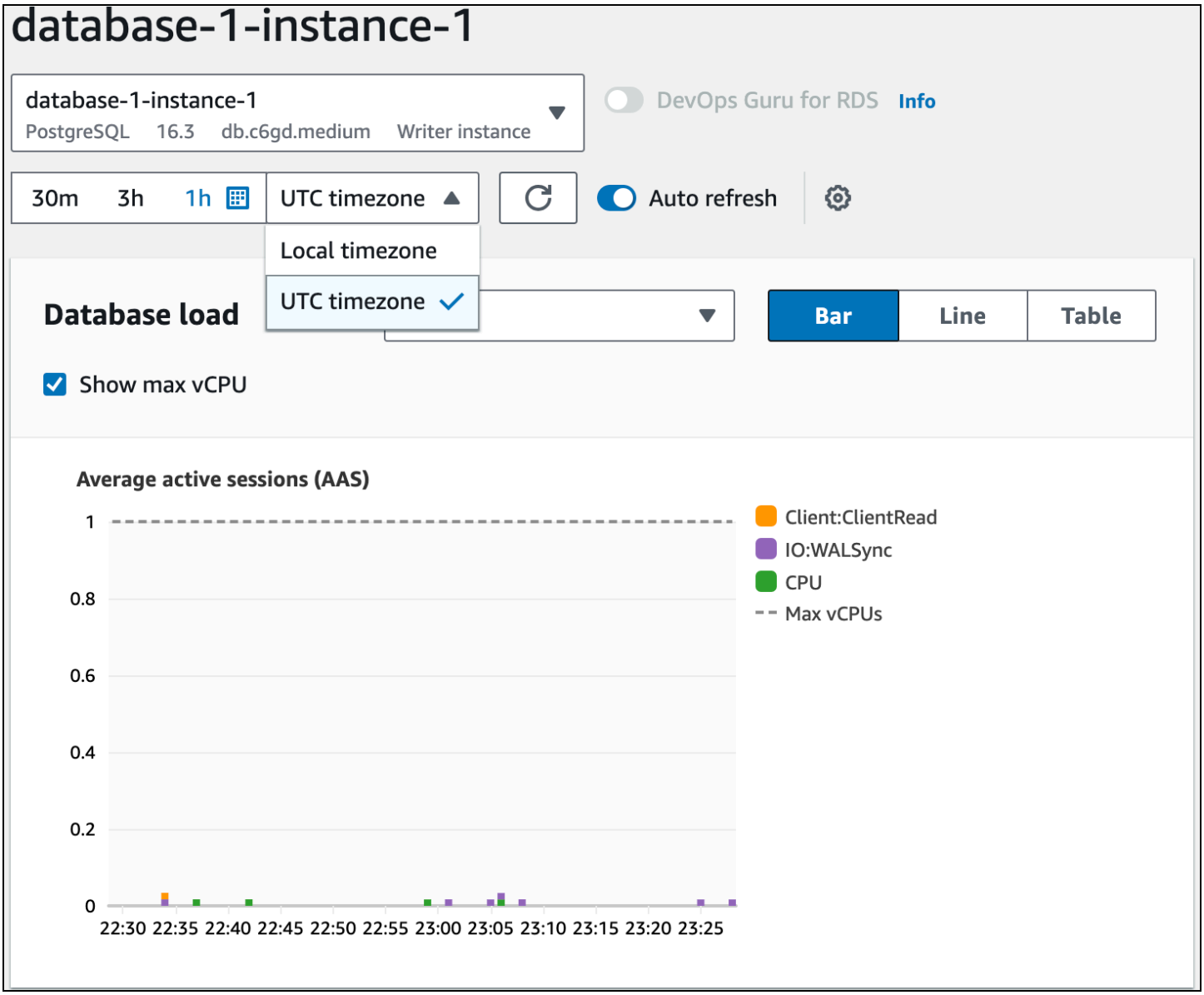
By default, the Performance Insights dashboard shows DB load for the last hour. You can adjust this range to be as short as 5 minutes or as long as 2 years. You can also select a custom relative range.



You can select an absolute range with a beginning and ending date and time. The following example shows the time range beginning at midnight on 9/25/24 and ending at 11:59 PM on 9/28/24.



By default, the time zone for the Performance Insights dashboard is Coordinated Universal Time (UTC). You can also choose the local time zone.

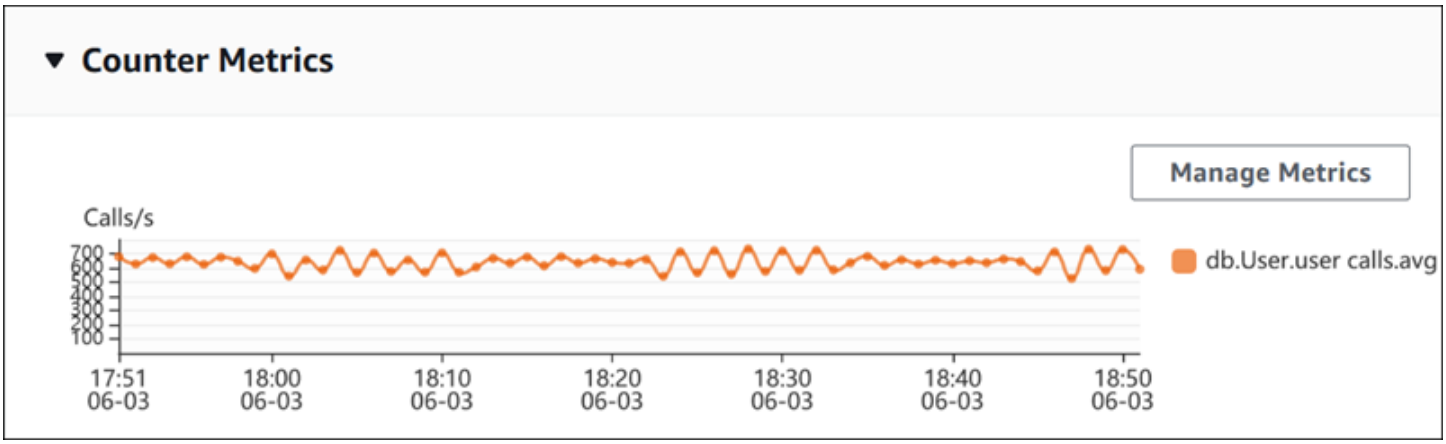


Counter metrics chart

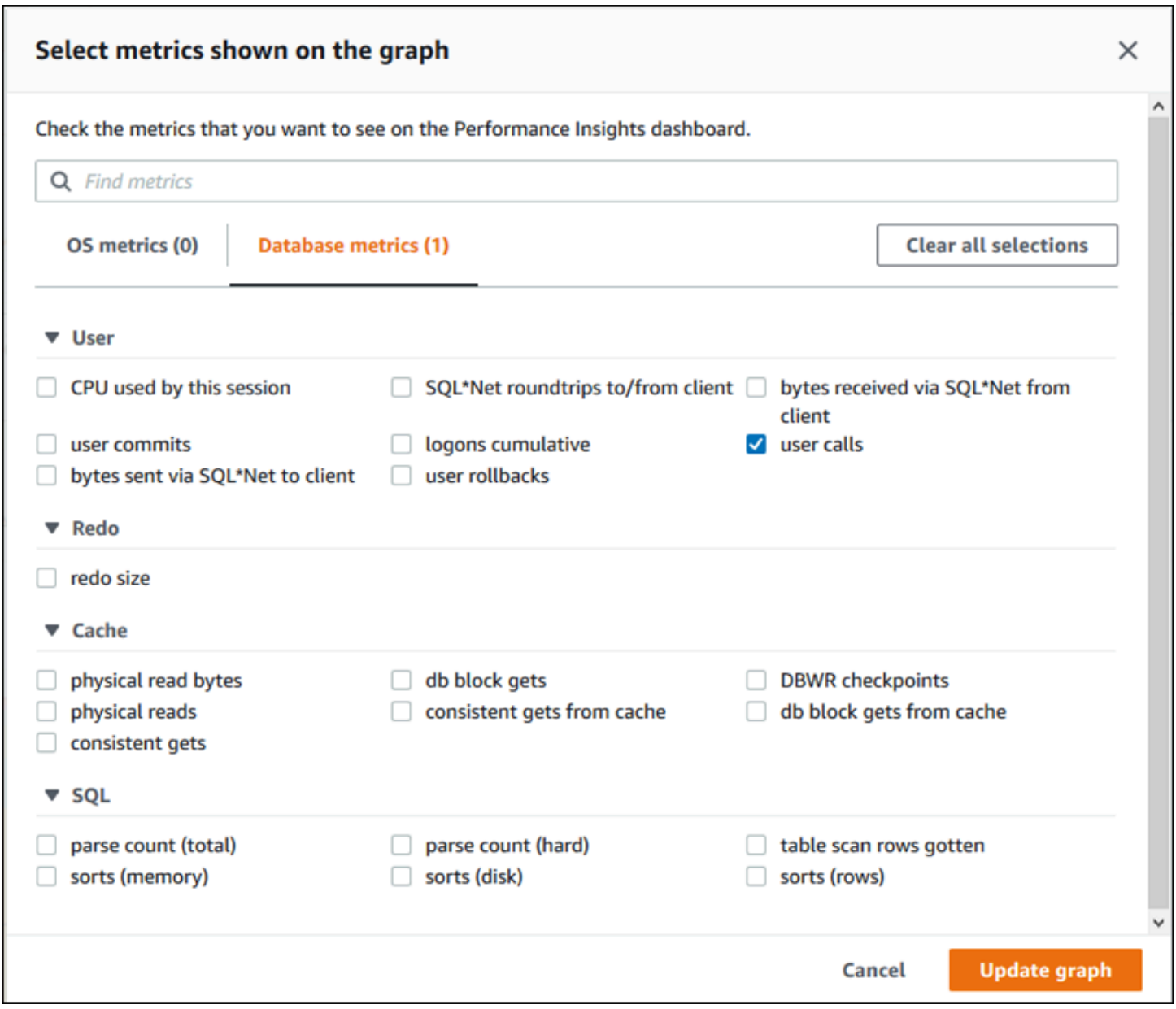
With counter metrics, you can customize the Performance Insights dashboard to include up to 10 additional graphs. These graphs show a selection of dozens of operating system and database performance metrics. You can correlate this information with DB load to help identify and analyze performance problems.

The **Counter metrics** chart displays data for performance counters. The default metrics depend on the DB engine:

- MySQL and MariaDB – `db.SQL.Innodb_rows_read.avg`
- Oracle – `db.User.user calls.avg`
- Microsoft SQL Server – `db.Databases.Active Transactions(_Total).avg`
- PostgreSQL – `db.Transactions.xact_commit.avg`



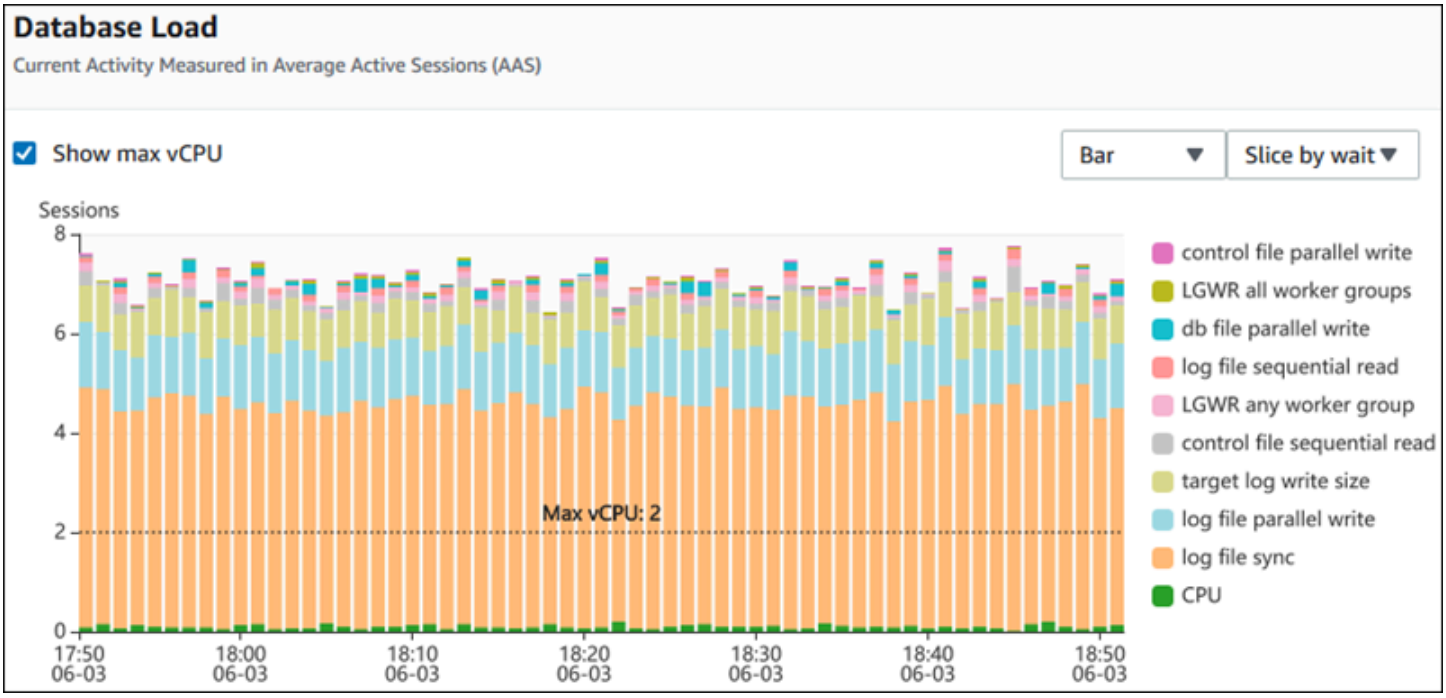
To change the performance counters, choose **Manage Metrics**. You can select multiple **OS metrics** or **Database metrics**, as shown in the following screenshot. To see details for any metric, hover over the metric name.



For descriptions of the counter metrics that you can add for each DB engine, see Performance Insights counter metrics.

Database load chart

The **Database load** chart shows how the database activity compares to DB instance capacity as represented by the **Max vCPU** line. By default, the stacked line chart represents DB load as average active sessions per unit of time. The DB load is sliced (grouped) by wait states.

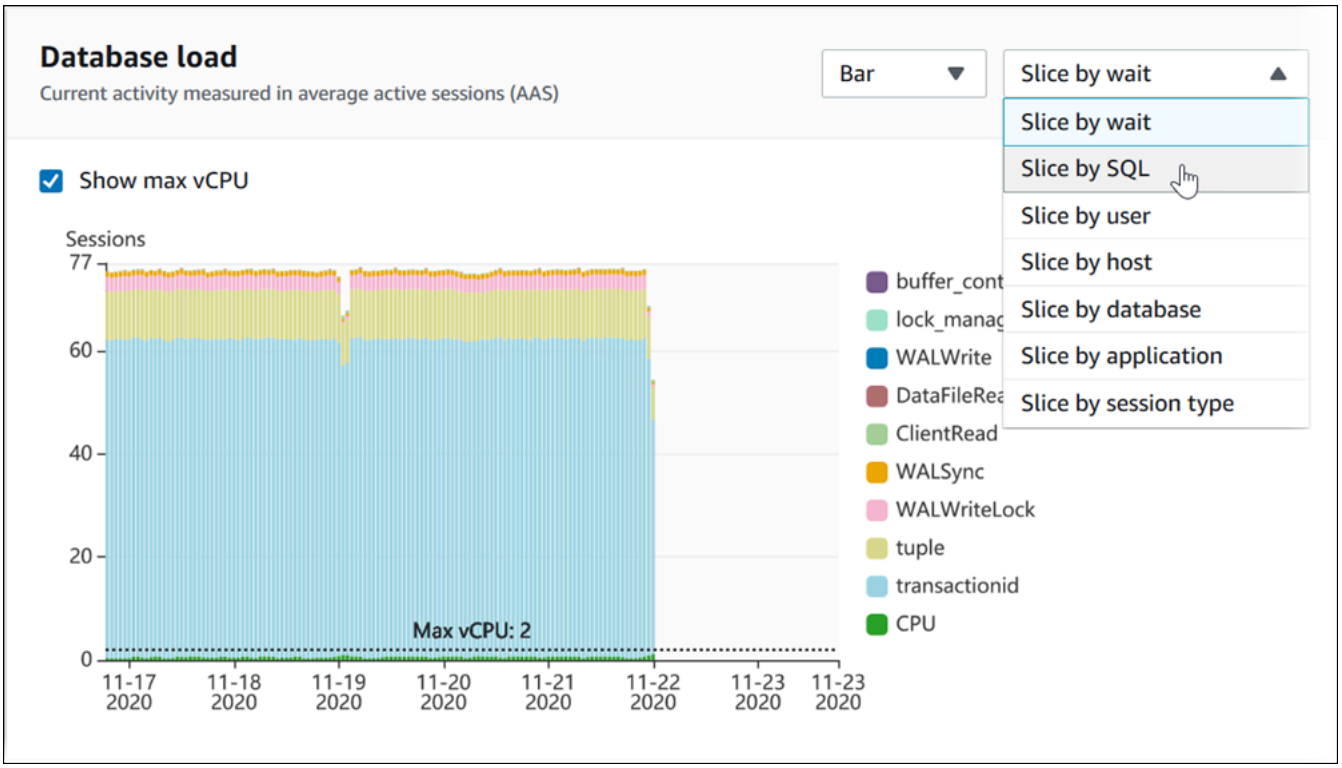


DB load sliced by dimensions

You can choose to display load as active sessions grouped by any supported dimensions. The following table shows which dimensions are supported for the different engines.

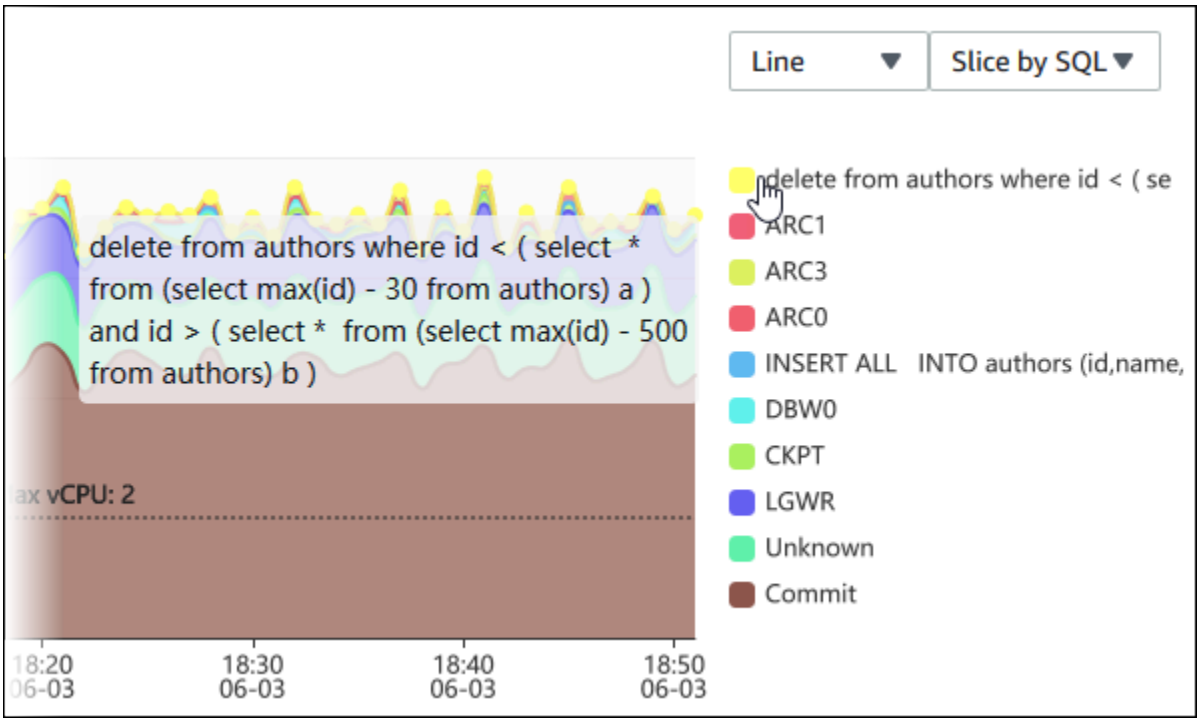
Dimension	Oracle	SQL Server	PostgreSQL	MySQL
Host	Yes	Yes	Yes	Yes
SQL	Yes	Yes	Yes	Yes
User	Yes	Yes	Yes	Yes
Waits	Yes	Yes	Yes	Yes
Plans	Yes	No	No	No
Application	No	No	Yes	No
Database	No	No	Yes	Yes
Session type	No	No	Yes	No

The following image shows the dimensions for a PostgreSQL DB instance.

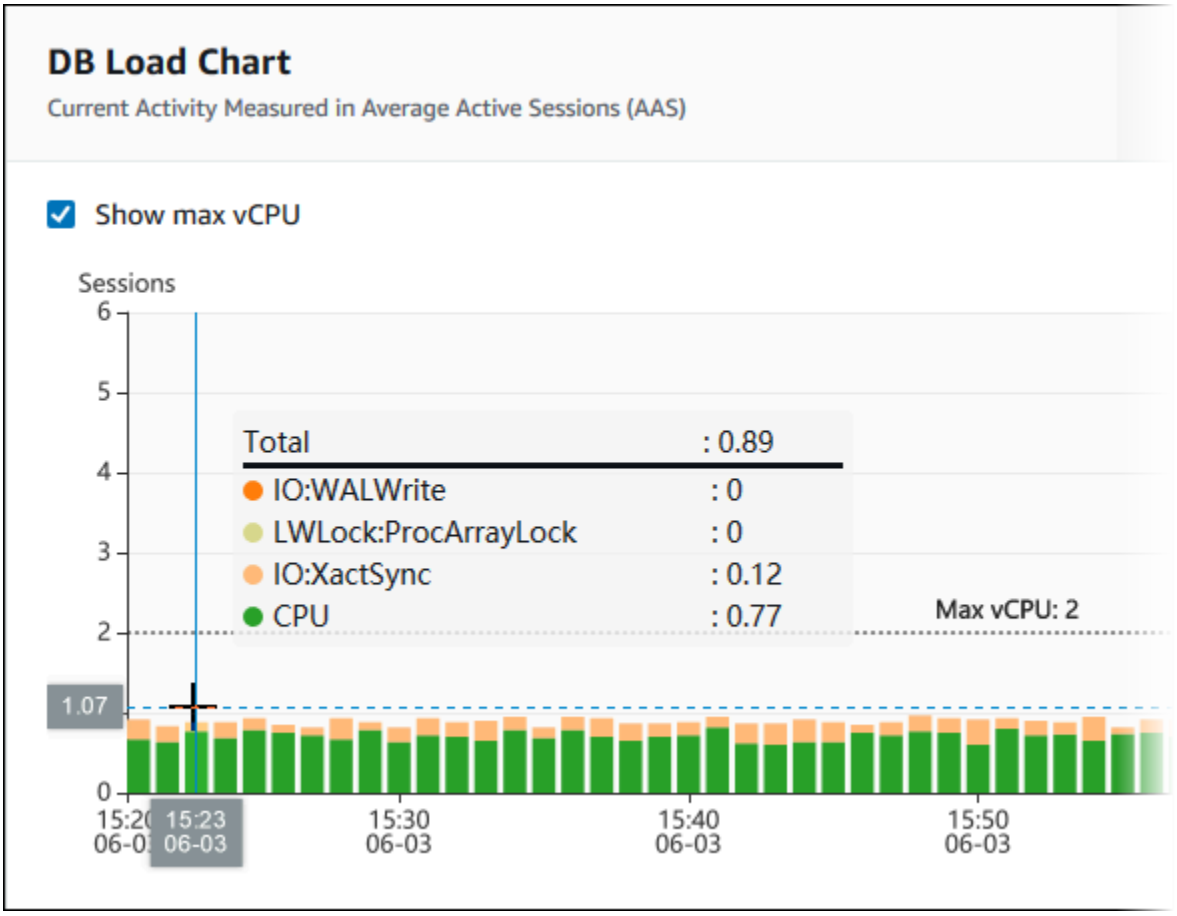


DB load details for a dimension item

To see details about a DB load item within a dimension, hover over the item name. The following image shows details for a SQL statement.

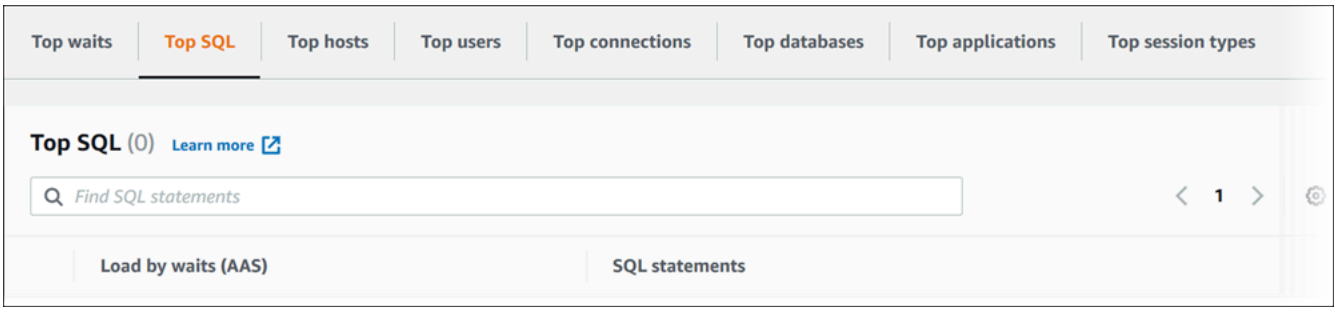


To see details for any item for the selected time period in the legend, hover over that item.



Top dimensions table

The Top dimensions table slices DB load by different dimensions. A dimension is a category or "slice by" for different characteristics of DB load. If the dimension is SQL, **Top SQL** shows the SQL statements that contribute the most to DB load.



Choose any of the following dimension tabs.

Tab	Description	Supported engines
Top SQL	The SQL statements that are currently running	All
Top waits	The event for which the database backend is waiting	All
Top hosts	The host name of the connected client	All
Top users	The user logged in to the database	All
Top databases	The name of the database to which the client is connected	PostgreSQL, MySQL, MariaDB, and SQL Server only
Top applications	The name of the application that is connected to the database	PostgreSQL and SQL Server only
Top session types	The type of the current session	PostgreSQL only

To learn how to analyze queries by using the **Top SQL** tab, see Overview of the Top SQL tab.

Accessing the Performance Insights dashboard

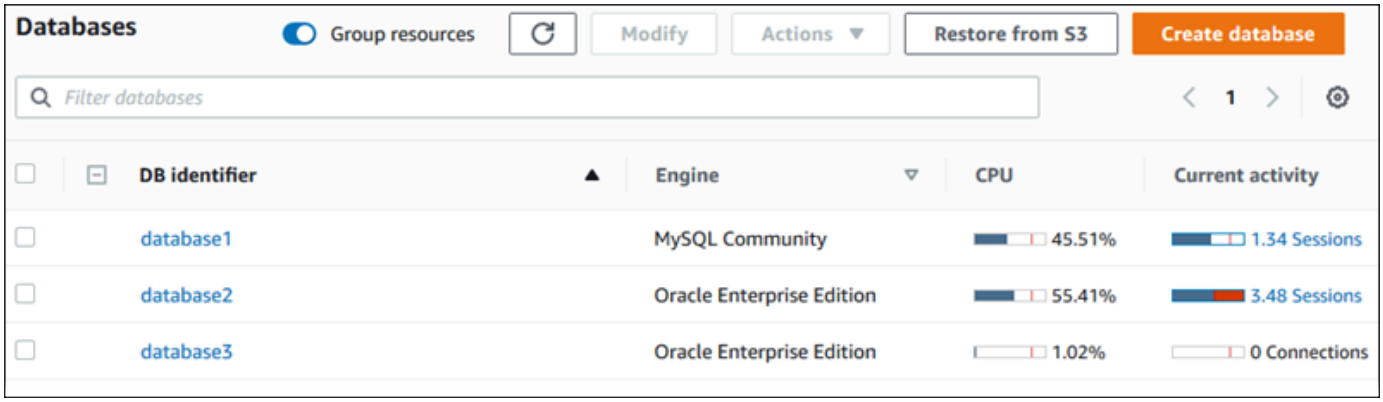
Amazon RDS provides a consolidated view of Performance Insights and CloudWatch metrics in the Performance Insights dashboard.

To access the Performance Insights dashboard, use the following procedure.

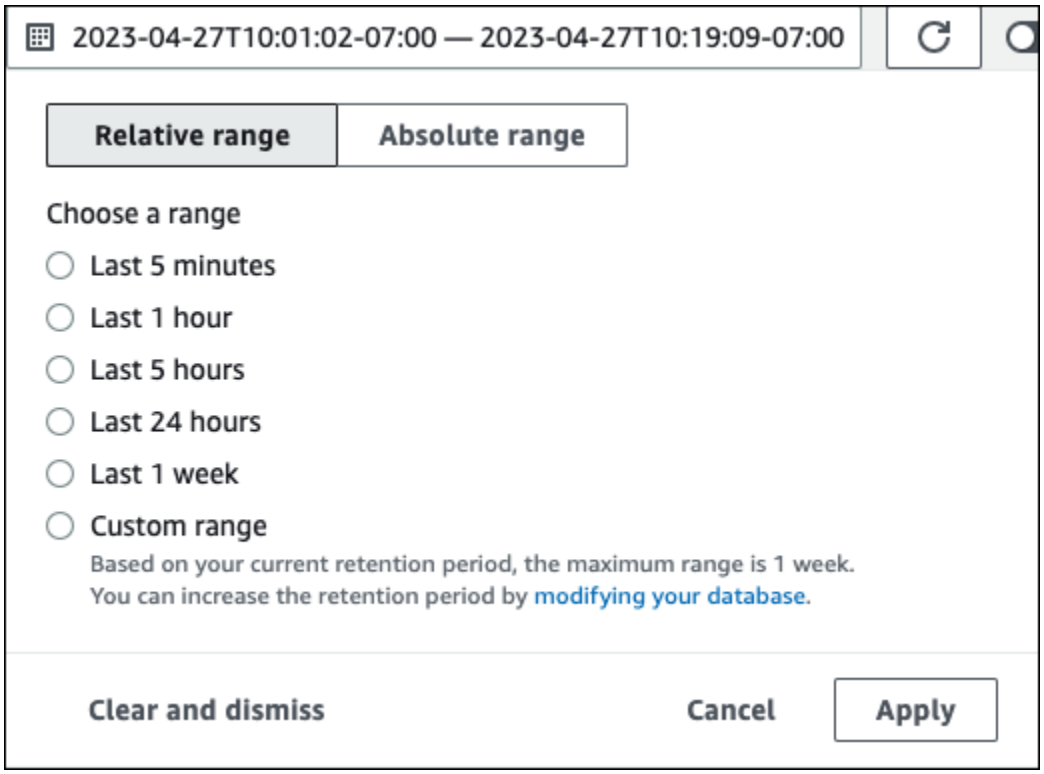
To view the Performance Insights dashboard in the AWS Management Console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.

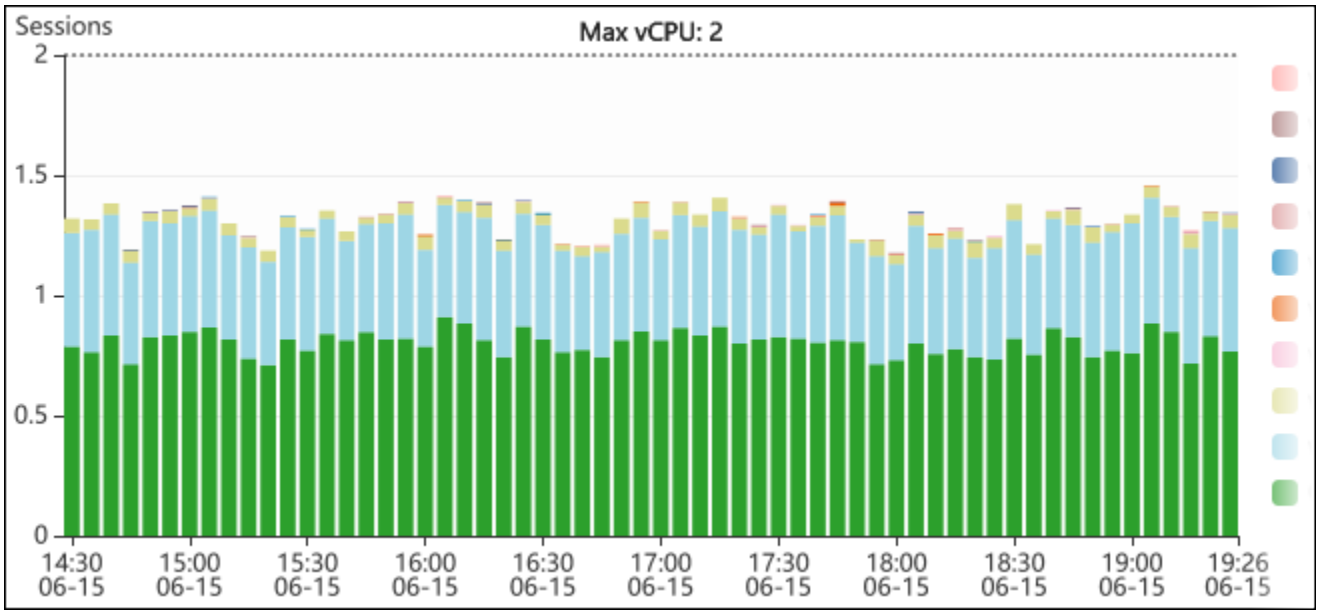
For DB instances with Performance Insights turned on, you can also access the Performance Insights dashboard by choosing the **Sessions** item in the list of DB instances. Under **Current activity**, the **Sessions** item shows the database load in average active sessions over the last five minutes. The bar graphically shows the load. When the bar is empty, the DB instance is idle. As the load increases, the bar fills with blue. When the load passes the number of virtual CPUs (vCPUs) on the DB instance class, the bar turns red, indicating a potential bottleneck.



4. (Optional) Choose the date or time range in the upper right and specify a different relative or absolute time interval. You can now specify a time period, and generate a database performance analysis report. The report provides the identified insights and recommendations. For more information, see Creating a performance analysis report in Performance Insights.

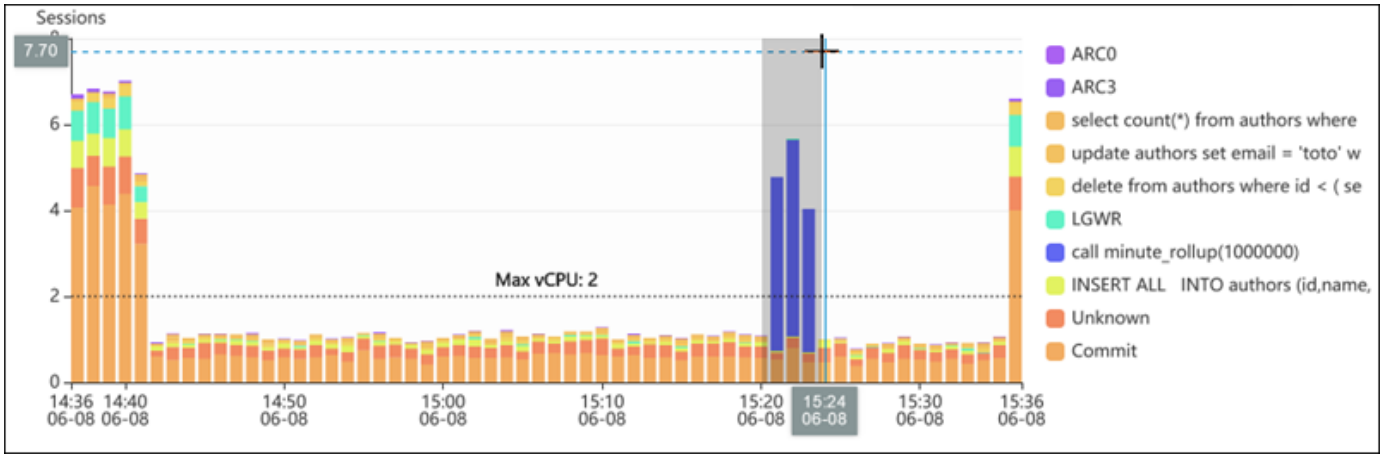


In the following screenshot, the DB load interval is 5 hours.

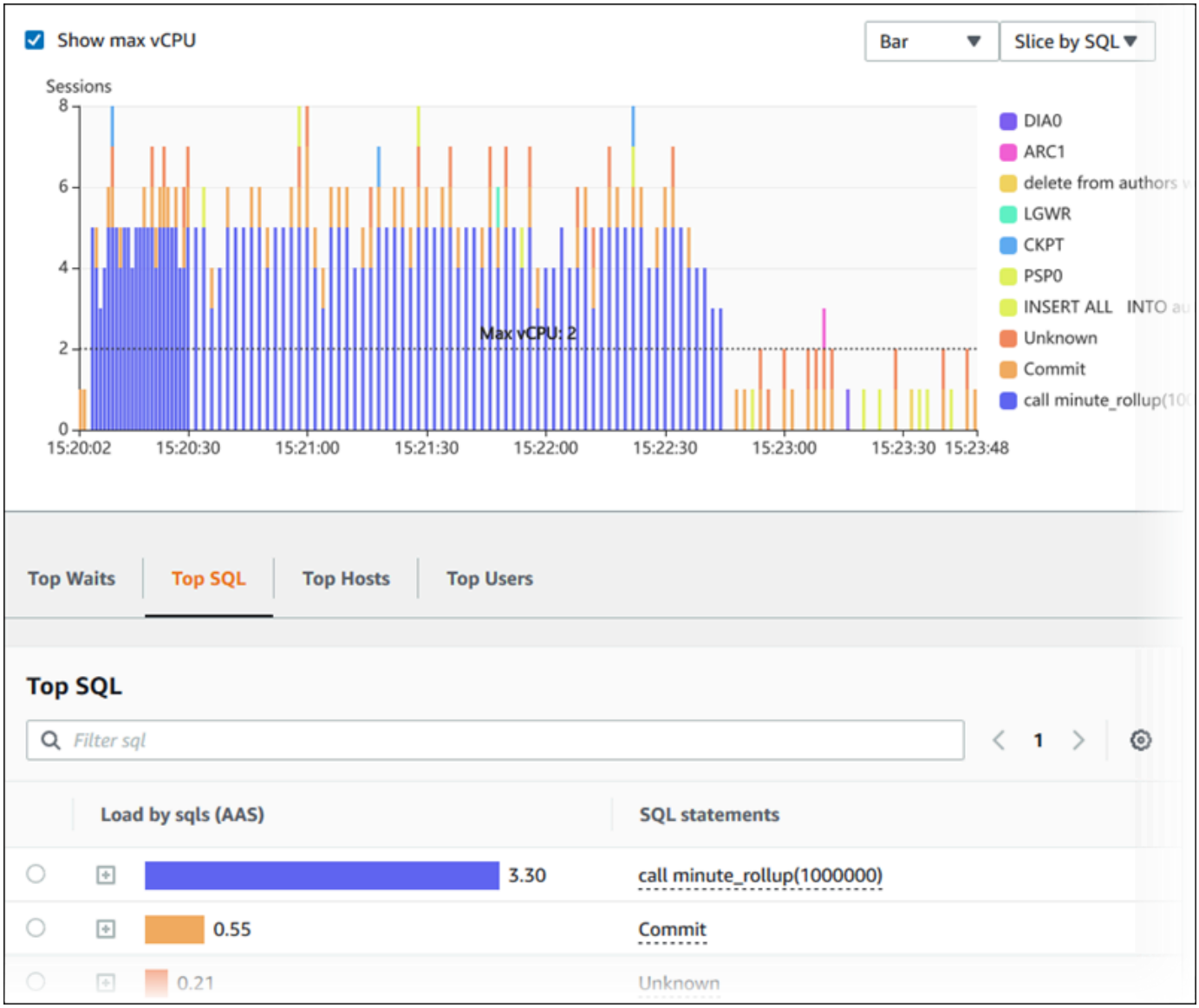


5. (Optional) To zoom in on a portion of the DB load chart, choose the start time and drag to the end of the time period you want.

   The selected area is highlighted in the DB load chart.



   When you release the mouse, the DB load chart zooms in on the selected AWS Region, and the **Top *dimensions*** table is recalculated.



6. (Optional) To refresh your data automatically, select **Auto refresh**.



The Performance Insights dashboard automatically refreshes with new data. The refresh rate depends on the amount of data displayed:

- 5 minutes refreshes every 10 seconds.
- 1 hour refreshes every 5 minutes.
- 5 hours refreshes every 5 minutes.
- 24 hours refreshes every 30 minutes.
- 1 week refreshes every day.

- 1 month refreshes every day.

Analyzing DB load by wait events

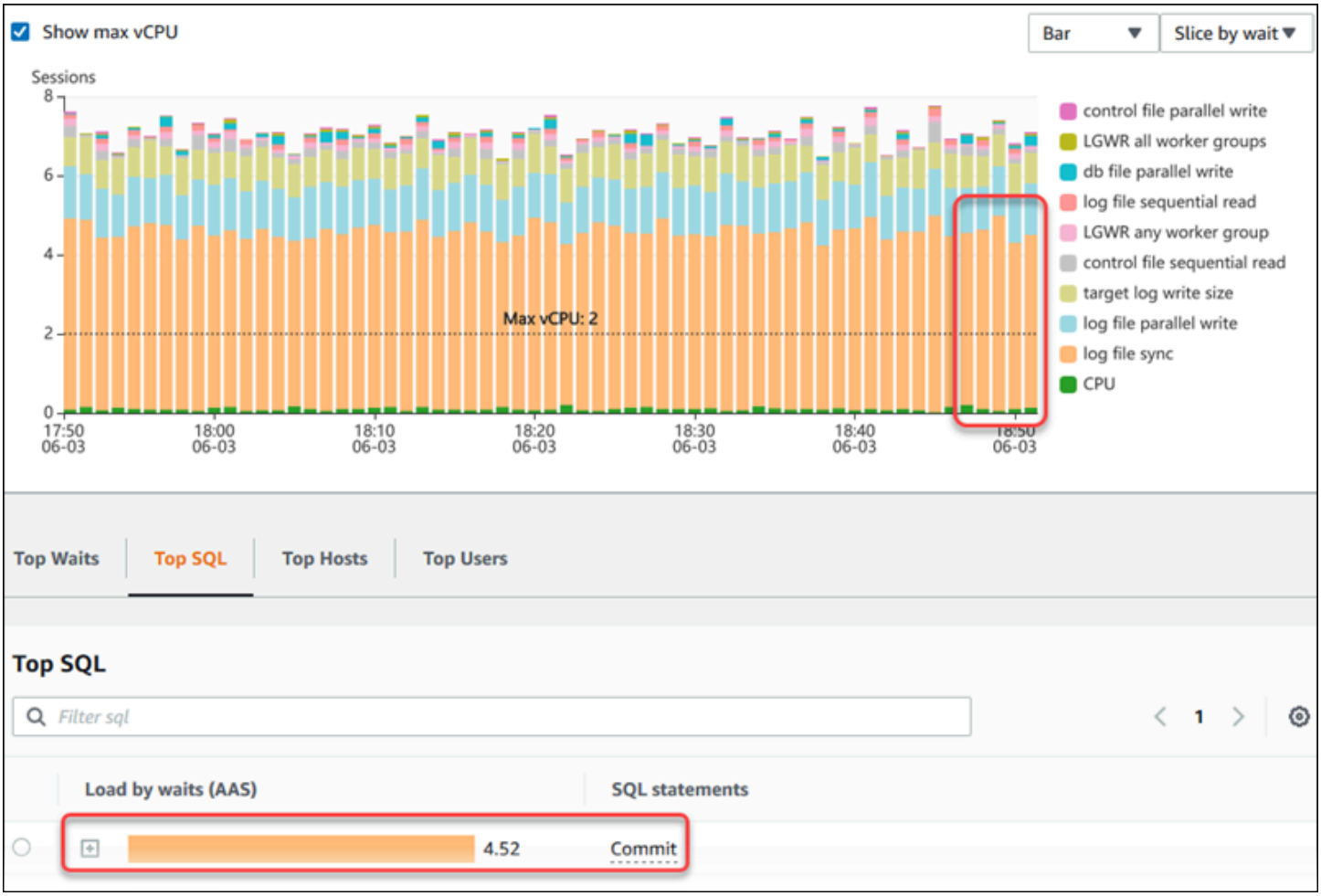
If the **Database load** chart shows a bottleneck, you can find out where the load is coming from. To do so, look at the top load items table below the **Database load** chart. Choose a particular item, like a SQL query or a user, to drill down into that item and see details about it.

DB load grouped by waits and top SQL queries is the default Performance Insights dashboard view. This combination typically provides the most insight into performance issues. DB load grouped by waits shows if there are any resource or concurrency bottlenecks in the database. In this case, the **SQL** tab of the top load items table shows which queries are driving that load.

Your typical workflow for diagnosing performance issues is as follows:

1. Review the **Database load** chart and see if there are any incidents of database load exceeding the **Max CPU** line.
2. If there is, look at the **Database load** chart and identify which wait state or states are primarily responsible.
3. Identify the digest queries causing the load by seeing which of the queries the **SQL** tab on the top load items table are contributing most to those wait states. You can identify these by the **DB Load by Wait** column.
4. Choose one of these digest queries in the **SQL** tab to expand it and see the child queries that it is composed of.

For example, in the dashboard following, **log file sync** waits account for most of the DB load. The **LGWR all worker groups** wait is also high. The **Top SQL** chart shows what is causing the **log file sync** waits: frequent `COMMIT` statements. In this case, committing less frequently will reduce DB load.



Analyzing database performance for a period of time

Analyze database performance with on-demand analysis by creating a performance analysis report for a period of time. View performance analysis reports to find performance issues, such as resource bottlenecks or changes in a query in your DB instance. The Performance Insights dashboard allows you to select a time period and create a performance analysis report. You can also add one or more tags to the report.

To use this feature, you must be using the paid tier retention period. For more information, see Pricing and data retention for Performance Insights

The report is available in the **Performance analysis reports - new** tab to select and view. The report contains the insights, related metrics, and recommendations to resolve the performance issue. The report is available to view for the duration of Performance Insights retention period.

The report is deleted if the start time of the report analysis period is outside of the retention period. You can also delete the report before the retention period ends.

To detect the performance issues and generate the analysis report for your DB instance, you must turn on Performance Insights. For more information about turning on Performance Insights, see Turning Performance Insights on and off for Amazon RDS.

For the region, DB engine, and instance class support information for this feature, see Amazon RDS DB engine, Region, and instance class support for Performance Insights features

In the following sections, you can create, view, add tags, and delete a performance analysis report.

Topics

- Creating a performance analysis report in Performance Insights
- Viewing a performance analysis report in Performance Insights
- Adding tags to a performance analysis report in Performance Insights
- Deleting a performance analysis report in Performance Insights

Creating a performance analysis report in Performance Insights

You can create a performance analysis report for a specific period in the Performance Insights dashboard. You can select a time period and add one or more tags to the analysis report.

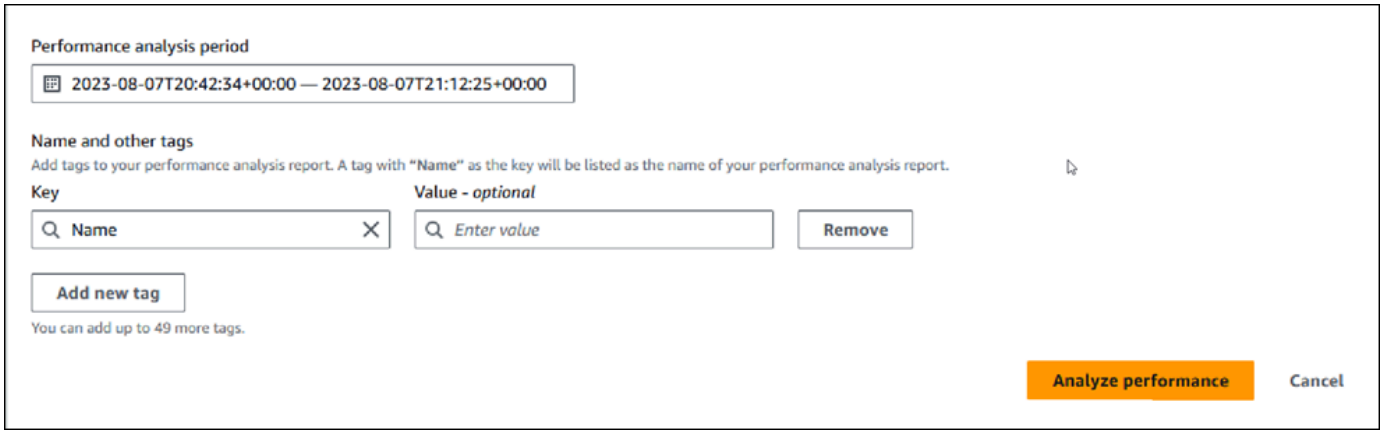
The analysis period can range from 5 minutes to 6 days. There must be at least 24 hours of performance data before the analysis start time.

For the region, DB engine, and instance class support information for this feature, see Amazon RDS DB engine, Region, and instance class support for Performance Insights features

To create a performance analysis report for a time period

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.
4. Choose **Analyze performance** in **Database load** section on the Performance Insights dashboard.

   The fields to set the time period and add one or more tags to the performance analysis report are displayed.

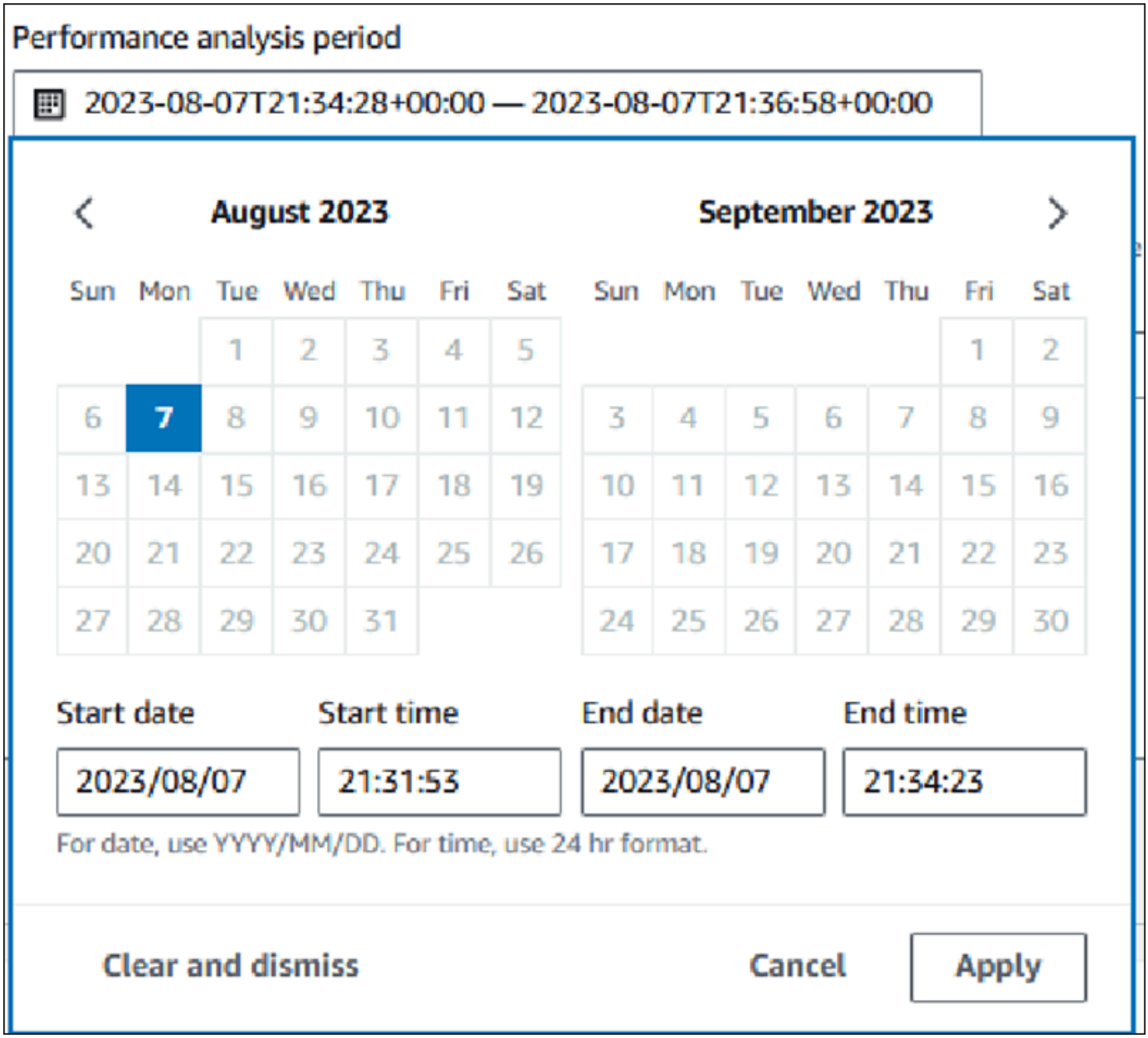


5. Choose the time period. If you set a time period in the **Relative range** or **Absolute range** in the upper right, you can only enter or select the analysis report date and time within this time period. If you select the analysis period outside of this time period, an error message displays.

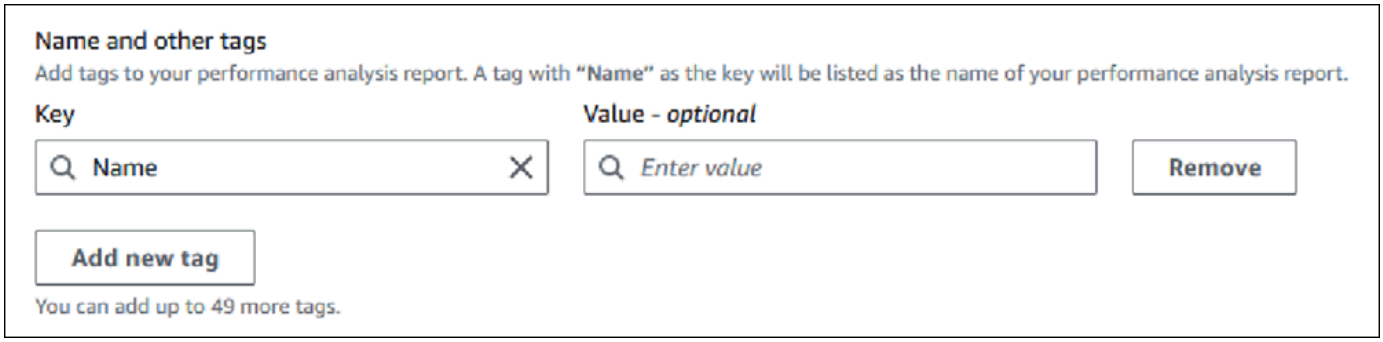
   To set the time period, you can do any of the following:

   - Press and drag any of the sliders on the DB load chart.

     The **Performance analysis period** box displays the selected time period and DB load chart highlights the selected time period.
   - Choose the **Start date**, **Start time**, **End date**, and **End time** in the **Performance analysis period** box.



6. (Optional) Enter **Key** and **Value-*optional*** to add a tag for the report.



7. Choose **Analyze performance**.

   A banner displays a message whether the report generation is successful or failed. The message also provides the link to view the report.

   The following example shows the banner with the report creation successful message.



   The report is available to view in **Performance analysis reports - new** tab.

You can create a performance analysis report using the AWS CLI. For an example on how to create a report using AWS CLI, see Creating a performance analysis report for a time period.

Viewing a performance analysis report in Performance Insights

The **Performance analysis reports - new** tab lists all the reports that are created for the DB instance. The following are displayed for each report:

- **ID**: Unique identifier of the report.
- **Name**: Tag key added to the report.
- **Report creation time**: Time you created the report.
- **Analysis start time**: Start time of the analysis in the report.
- **Analysis end time**: End time of the analysis in the report.

To view a performance analysis report

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance for which you want to view the analysis report.
4. Scroll down and choose **Performance analysis reports - new** tab in the Performance Insights dashboard.

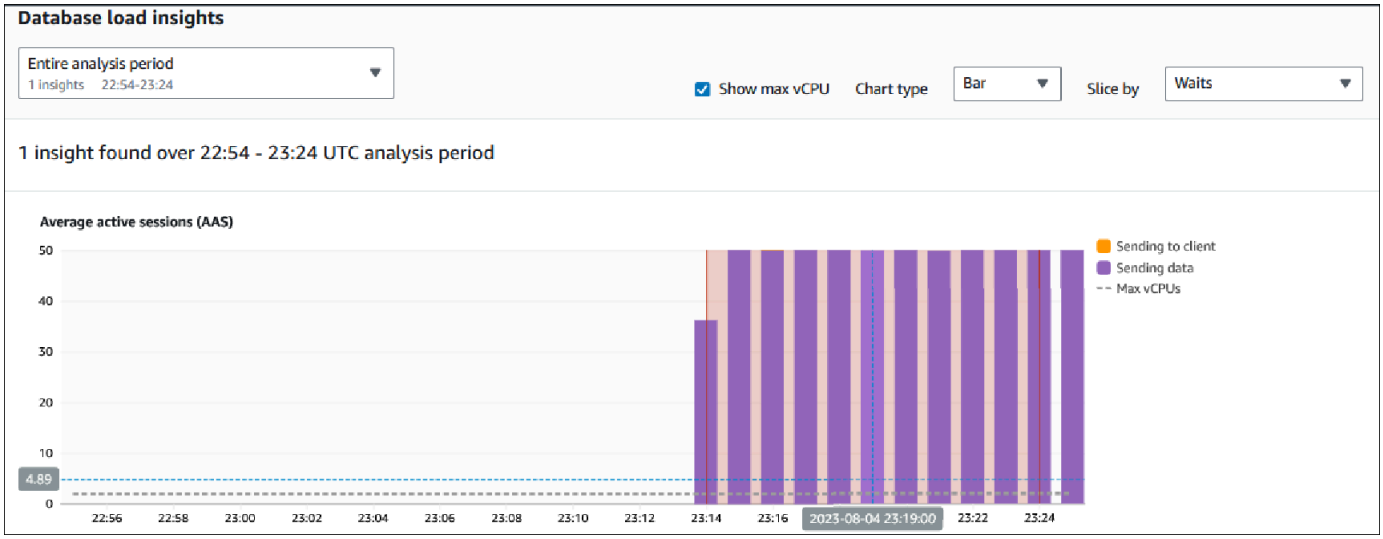
   All the analysis reports for the different time periods are displayed.

5. Choose **ID** of the report you want to view.

The DB load chart displays the entire analysis period by default if more than one insight is identified. If the report has identified one insight then the DB load chart displays the insight by default.

The dashboard also lists the tags for the report in the **Tags** section.

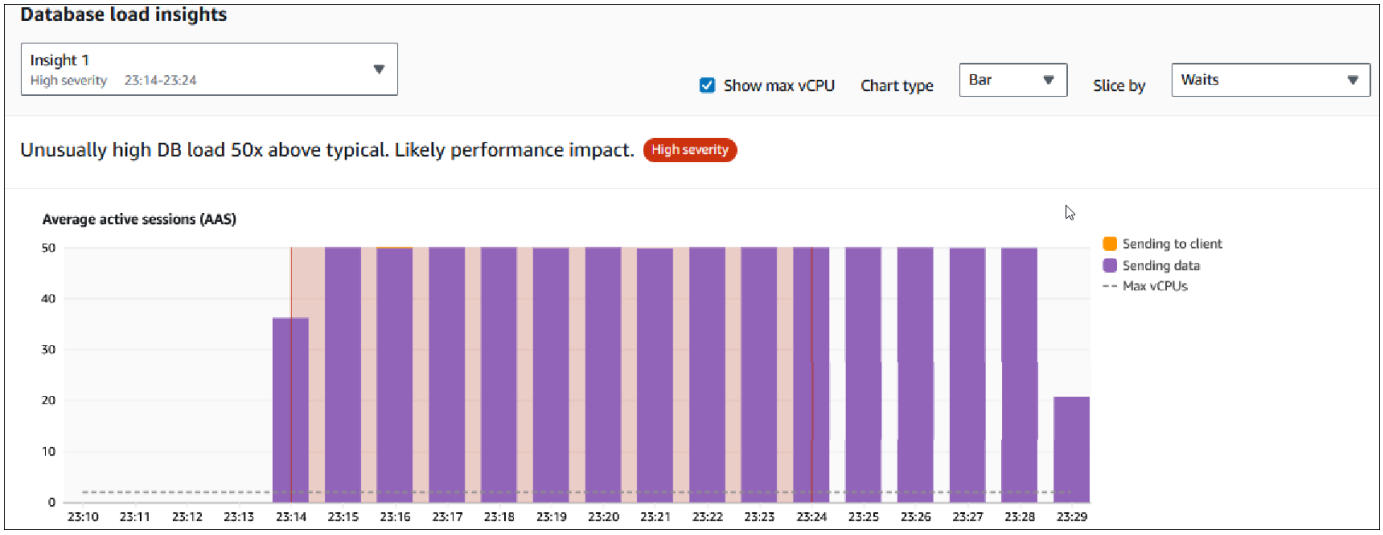
The following example shows the entire analysis period for the report.

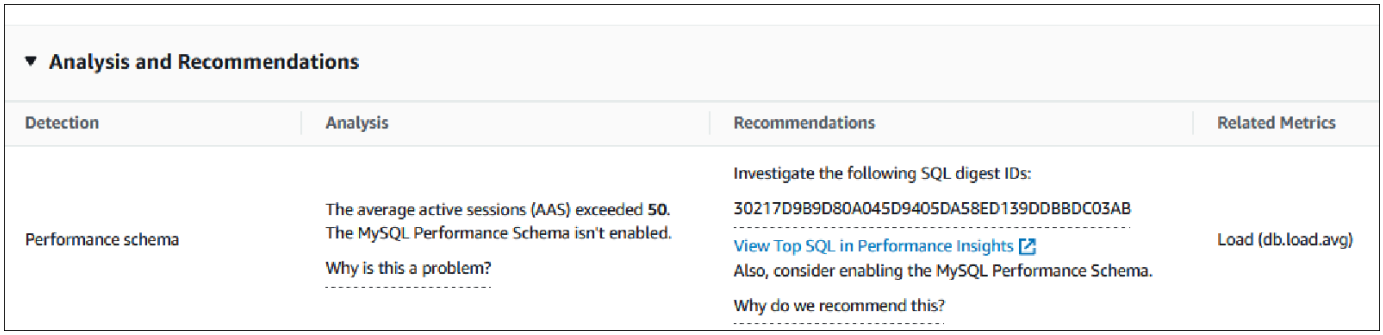


6. Choose the insight in the **Database load insights** list you want to view if more than one insight is identified in the report.

   The dashboard displays the insight message, DB load chart highlighting the time period of the insight, analysis and recommendations, and the list of report tags.

   The following example shows the DB load insight in the report.





Adding tags to a performance analysis report in Performance Insights

You can add a tag when you create or view a report. You can add up to 50 tags for a report.

You need permissions to add the tags. For more information about the access policies for Performance Insights, see Configuring access policies for Performance Insights

To add one or more tags while creating a report, see step 6 in the procedure Creating a performance analysis report in Performance Insights.

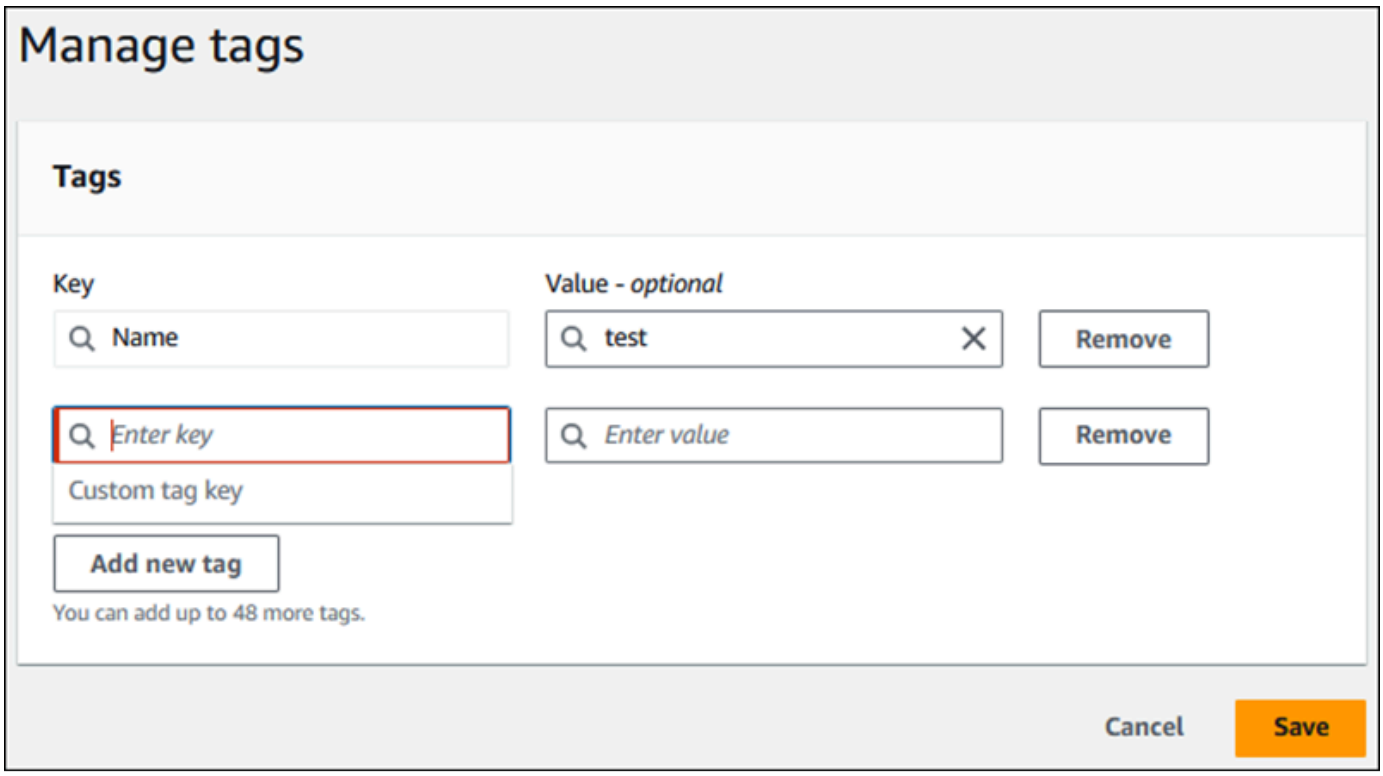
To add one or more tags when viewing a report

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.

   The Performance Insights dashboard appears for the DB instance.
4. Scroll down and choose **Performance analysis reports - new** tab.
5. Choose the report for which you want to add the tags.

   The dashboard displays the report.
6. Scroll down to **Tags** and choose **Manage tags**.
7. Choose **Add new tag**.
8. Enter the **Key** and **Value - *optional***, and choose **Add new tag**.

   The following example provides the option to add a new tag for the selected report.



A new tag is created for the report.

The list of tags for the report is displayed in the **Tags** section on the dashboard. If you want to remove a tag from the report, choose **Remove** next to the tag.

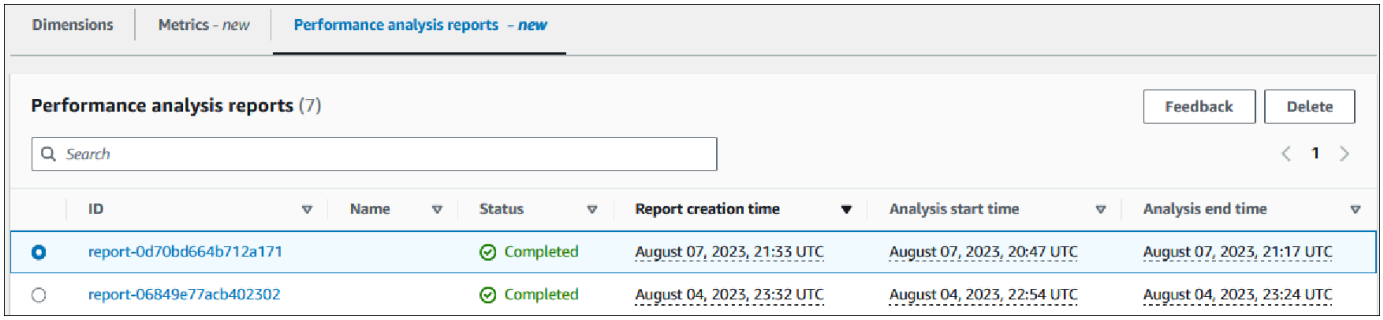
Deleting a performance analysis report in Performance Insights

You can delete a report from the list of reports displayed in the **Performance analysis reports** tab or while viewing a report.

To delete a report

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. Choose a DB instance.

   The Performance Insights dashboard appears for the DB instance.
4. Scroll down and choose **Performance analysis reports - new** tab.
5. Select the report you want to delete and choose **Delete** in the upper right.



A confirmation window is displayed. The report is deleted after you choose confirm.

6. (Optional) Choose **ID** of the report you want to delete.

   In the report page, choose **Delete** in the upper right.

   A confirmation window is displayed. The report is deleted after you choose confirm.

Analyzing queries with the Top SQL tab in Performance Insights

In the Amazon RDS Performance Insights dashboard, you can find information about running and recent queries in the **Top SQL** tab in the **Top dimensions** table. You can use this information to tune your queries.

Topics

- Overview of the Top SQL tab
- Accessing more SQL text in the Performance Insights dashboard
- Viewing SQL statistics in the Performance Insights dashboard

Overview of the Top SQL tab

By default, the **Top SQL** tab shows the 25 queries that are contributing the most to DB load. To help tune your queries, you can analyze information such as the query text and SQL statistics. You can also choose the statistics that you want to appear in the **Top SQL** tab.

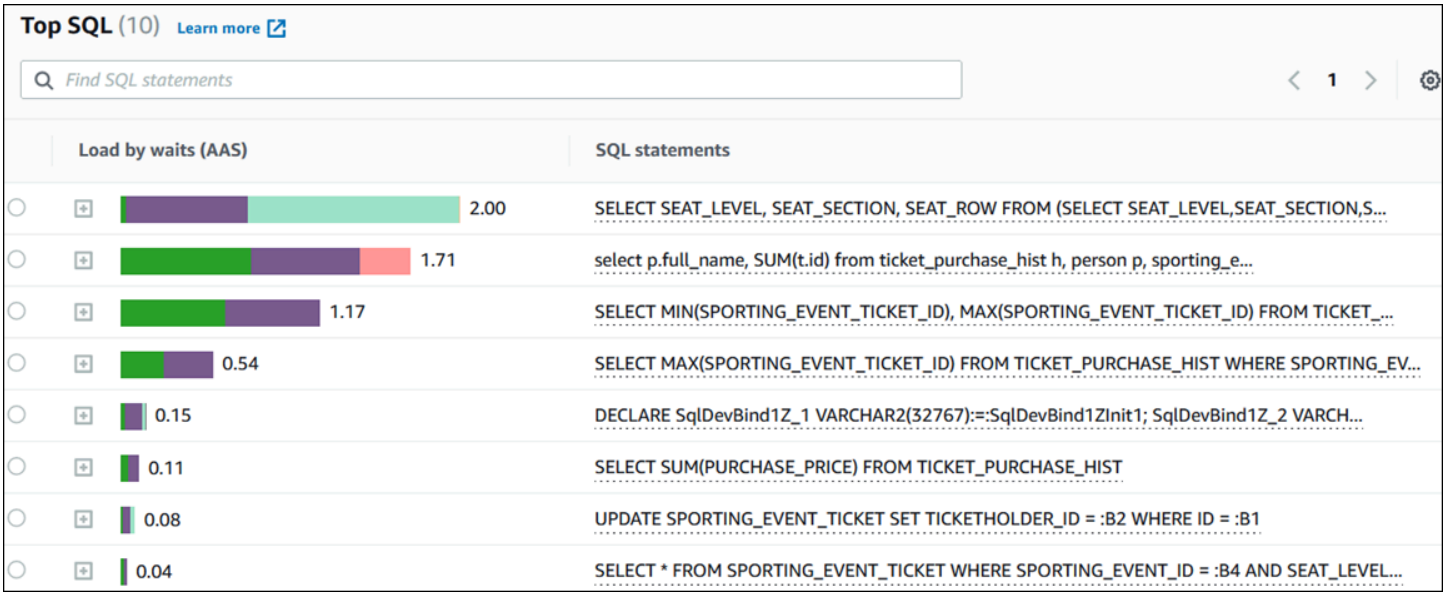
Topics

- SQL text
- SQL statistics
- Load by waits (AAS)

- View SQL information
- Choose statistics preferences

SQL text

By default, each row in the **Top SQL** table shows 500 bytes of text for each statement.

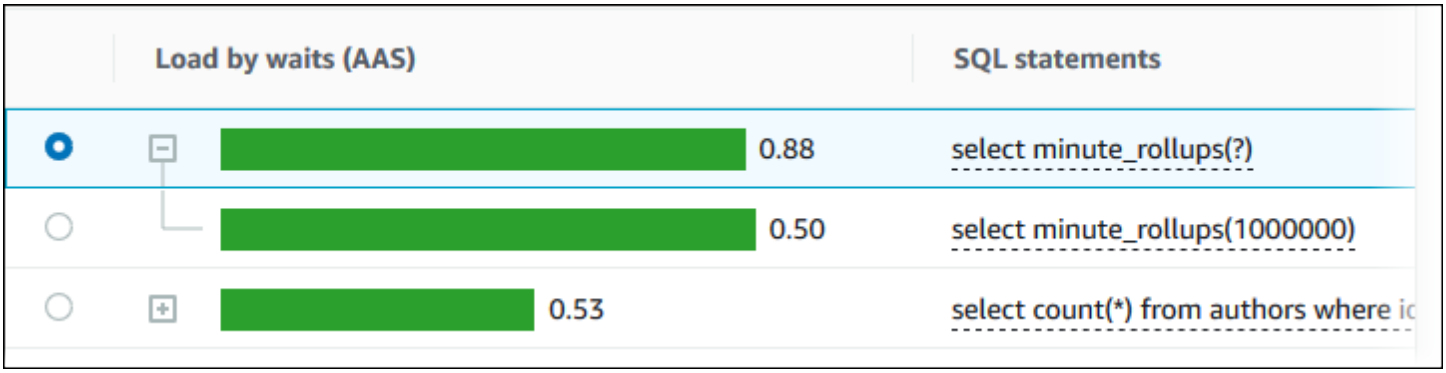


To learn how to see more than the default 500 bytes of SQL text, see Accessing more SQL text in the Performance Insights dashboard.

A *SQL digest* is a composite of multiple actual queries that are structurally similar but might have different literal values. The digest replaces hardcoded values with a question mark. For example, a digest might be `SELECT * FROM emp WHERE lname= ?`. This digest might include the following child queries:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

To see the literal SQL statements in a digest, select the query, and then choose the plus symbol (+). In the following example, the selected query is a digest.



> **Note**
>
> A SQL digest groups similar SQL statements, but doesn't redact sensitive information.

Performance Insights can show Oracle SQL text as **Unknown**. The text has this status in the following situations:

- An Oracle database user other than SYS is active but not currently executing SQL. For example, when a parallel query completes, the query coordinator waits for helper processes to send their session statistics. For the duration of the wait, the query text shows **Unknown**.
- For an RDS for Oracle instance on Standard Edition 2, Oracle Resource Manager limits the number of parallel threads. The background process doing this work causes the query text to show as **Unknown**.

SQL statistics

SQL statistics are performance-related metrics about SQL queries. For example, Performance Insights might show executions per second or rows processed per second. Performance Insights collects statistics for only the most common queries. Typically, these match the top queries by load shown in the Performance Insights dashboard.

Every line in the **Top SQL** table shows relevant statistics for the SQL statement or digest, as shown in the following example.



Performance Insights can report `0.00` and `-` (unknown) for SQL statistics. This situation occurs under the following conditions:

- Only one sample exists. For example, Performance Insights calculates rates of change for RDS PostgreSQL queries based on multiple samples from the `pg_stat_statements` view. When a workload runs for a short time, Performance Insights might collect only one sample, which means that it can't calculate a rate of change. The unknown value is represented with a dash (`-`).
- Two samples have the same values. Performance Insights can't calculate a rate of change because no change has occurred, so it reports the rate as `0.00`.
- An RDS PostgreSQL statement lacks a valid identifier. PostgreSQL creates a identifier for a statement only after parsing and analysis. Thus, a statement can exist in the PostgreSQL internal in-memory structures with no identifier. Because Performance Insights samples internal in-memory structures once per second, low-latency queries might appear for only a single sample. If the query identifier isn't available for this sample, Performance Insights can't associate this statement with its statistics. The unknown value is represented with a dash (`-`).

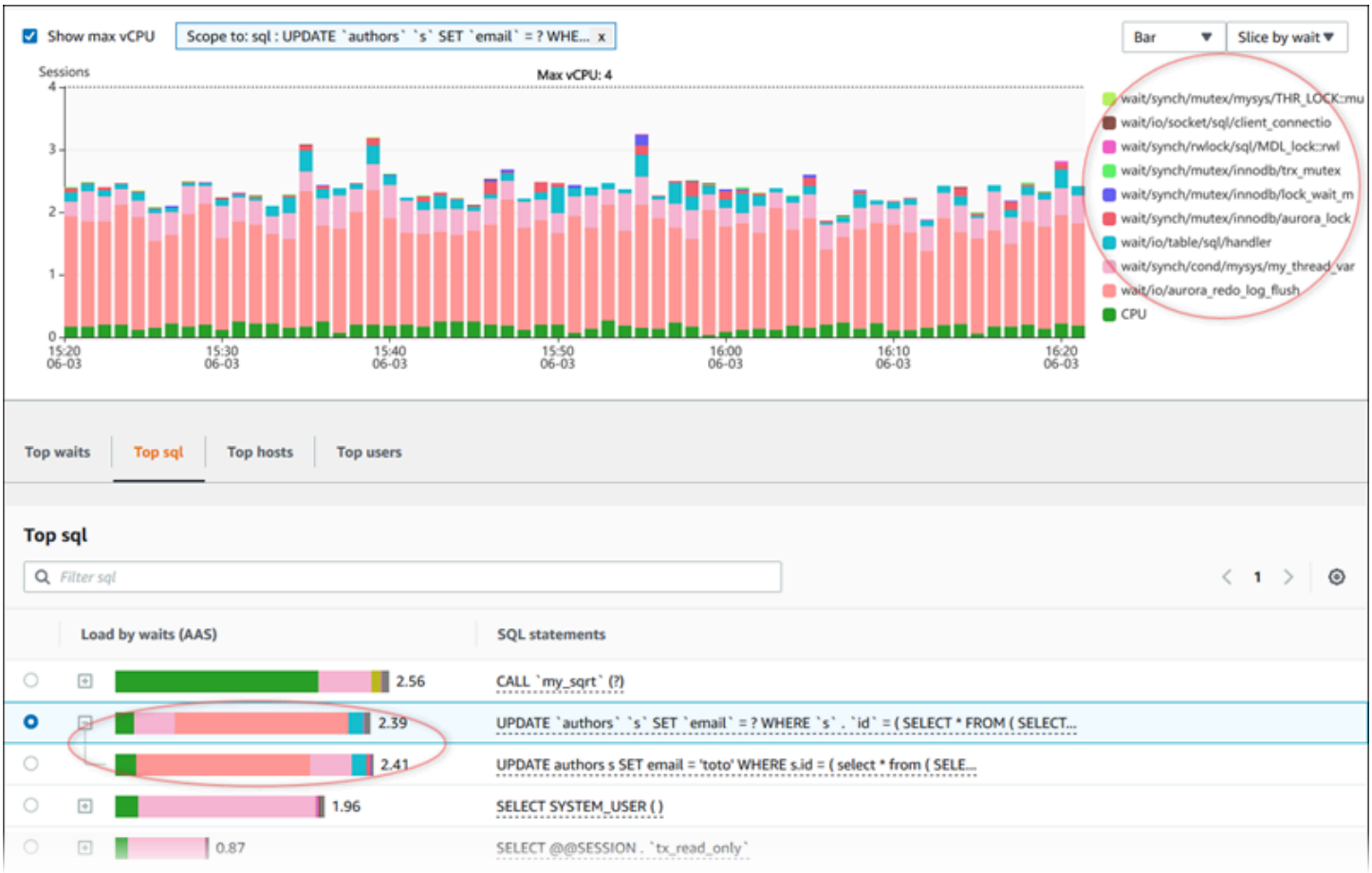
For a description of the SQL statistics for the Amazon RDS engines, see SQL statistics for Performance Insights.

Load by waits (AAS)

In **Top SQL**, the **Load by waits (AAS)** column illustrates the percentage of the database load associated with each top load item. This column reflects the load for that item by whatever grouping is currently selected in the **DB Load Chart**. For more information about Average active sessions (AAS), see Average active sessions.

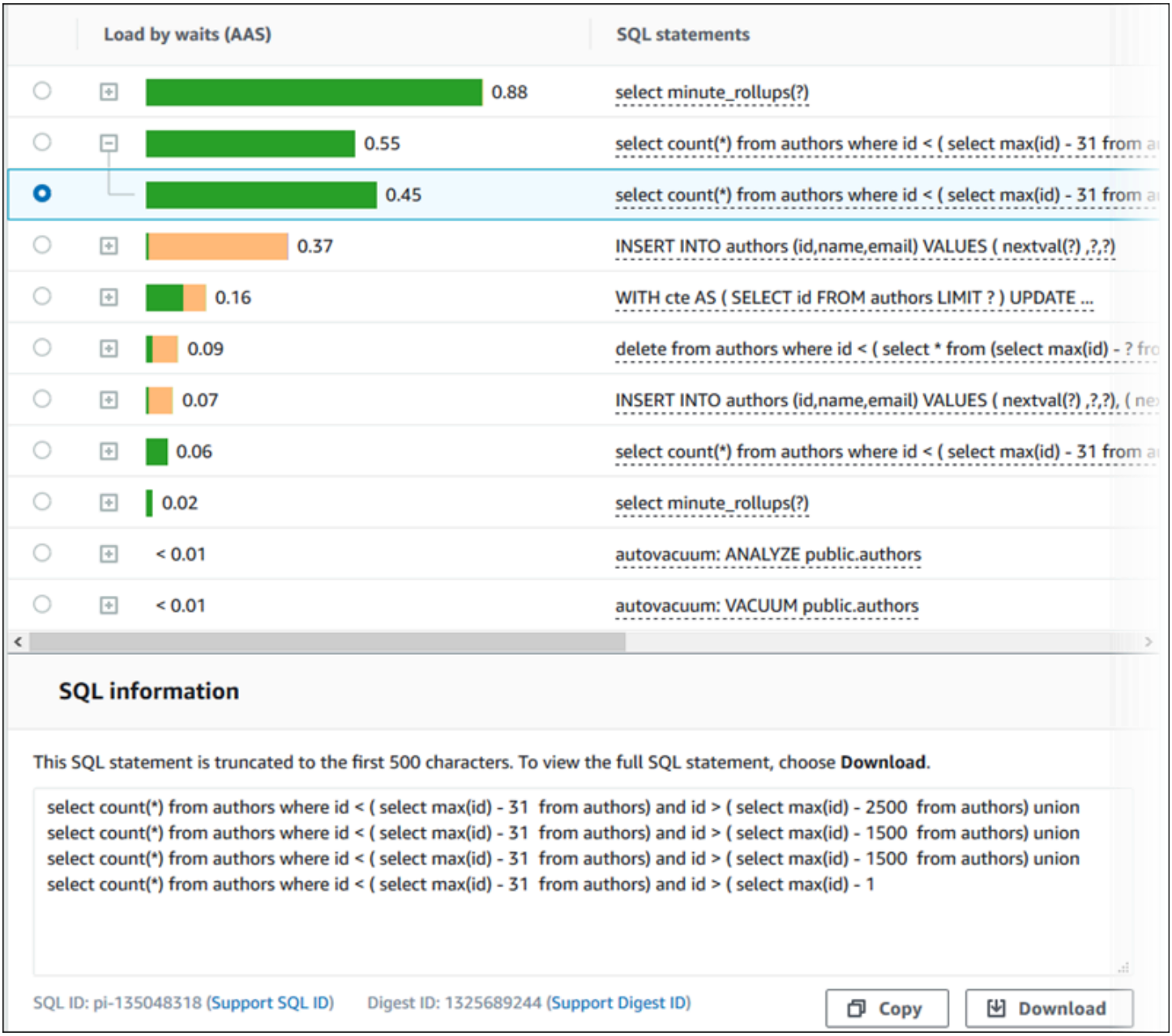
For example, you might group the **DB load** chart by wait states. You examine SQL queries in the top load items table. In this case, the **DB Load by Waits** bar is sized, segmented, and color-coded to

show how much of a given wait state that query is contributing to. It also shows which wait states are affecting the selected query.



View SQL information

In the **Top SQL** table, you can open a statement to view its information. The information appears in the bottom pane.

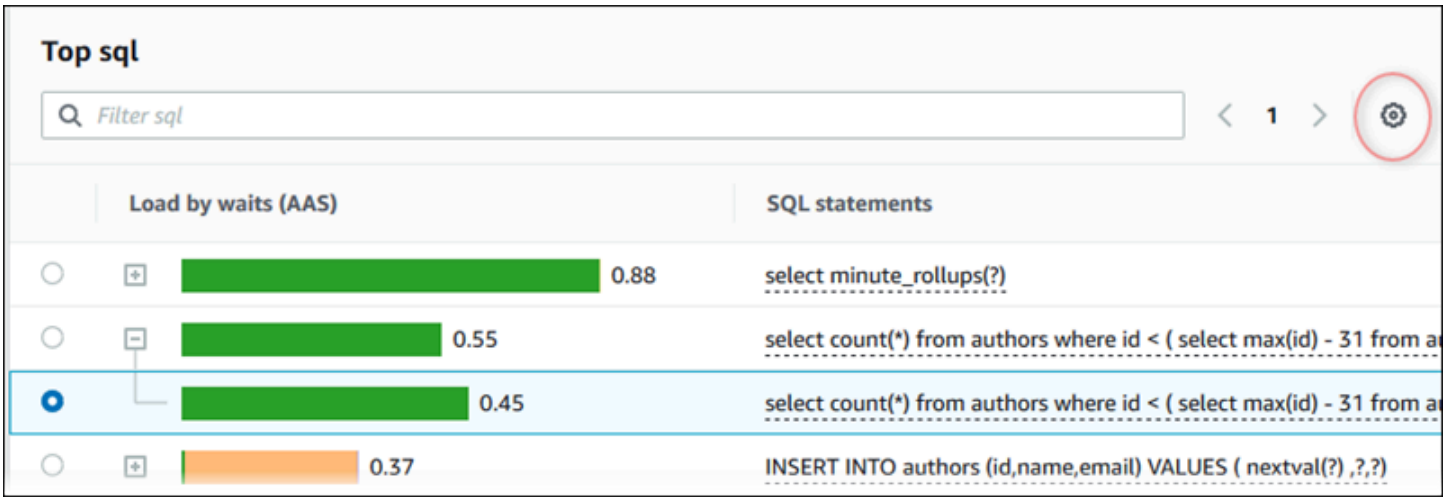


The following types of identifiers (IDs) that are associated with SQL statements:

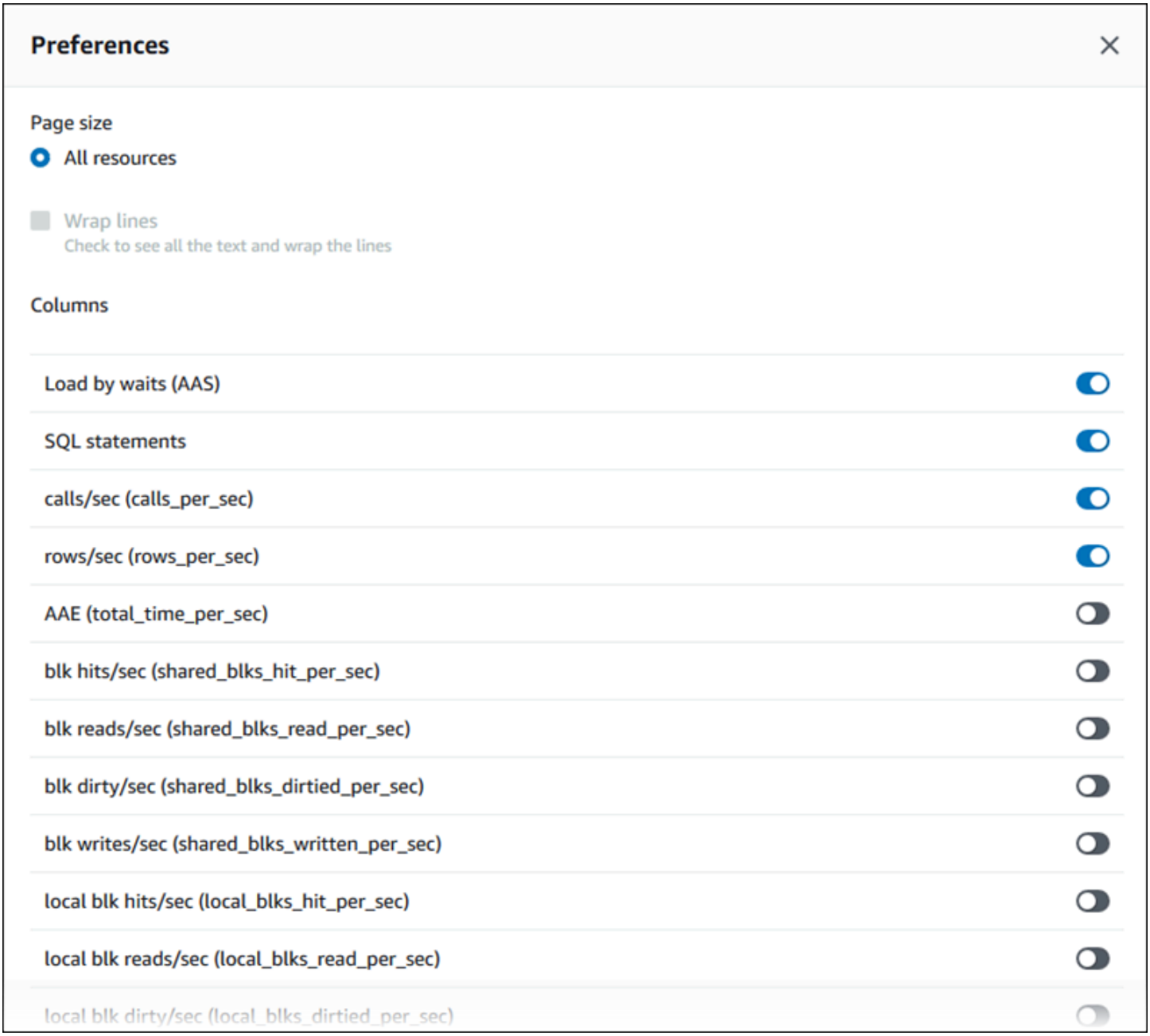
- **Support SQL ID** – A hash value of the SQL ID. This value is only for referencing a SQL ID when you are working with AWS Support. AWS Support doesn't have access to your actual SQL IDs and SQL text.
- **Support Digest ID** – A hash value of the digest ID. This value is only for referencing a digest ID when you are working with AWS Support. AWS Support doesn't have access to your actual digest IDs and SQL text.

Choose statistics preferences

You can control the statistics displayed in the **Top SQL** tab by choosing the **Preferences** icon.



When you choose the **Preferences** icon, the **Preferences** window opens. The following screenshot is an example of the **Preferences** window.



To enable the statistics that you want to appear in the **Top SQL** tab, use your mouse to scroll to the bottom of the window, and then choose **Continue**.

For more information about per-second or per-call statistics for the Amazon RDS engines, see the engine specific SQL statistics section in SQL statistics for Performance Insights

Accessing more SQL text in the Performance Insights dashboard

By default, each row in the **Top SQL** table shows 500 bytes of SQL text for each SQL statement.



When a SQL statement exceeds 500 bytes, you can view more text in the **SQL text** section below the **Top SQL** table. In this case, the maximum length for the text displayed in **SQL text** is 4 KB. This limit is introduced by the console and is subject to the limits set by the database engine. To save the text shown in **SQL text**, choose **Download**.

Topics

- Text size limits for Amazon RDS engines
- Setting the SQL text limit for Amazon RDS for PostgreSQL DB instances
- Viewing and downloading SQL text in the Performance Insights dashboard

Text size limits for Amazon RDS engines

When you download SQL text, the database engine determines its maximum length. You can download SQL text up to the following per-engine limits.

DB engine	Maximum length of downloaded text
Amazon RDS for MySQL and MariaDB	The length is fixed at 4,096 bytes when the Performance Schema is enabled. If the Performance Schema isn't enabled, the length is fixed at 65,535 bytes.
Amazon RDS for Microsoft SQL Server	4,096 characters
Amazon RDS for Oracle	1,000 bytes

The **SQL text** section of the Performance Insights console displays up to the maximum that the engine returns. For example, if MySQL returns at most 1 KB to Performance Insights, it can only collect and show 1 KB, even if the original query is larger. Thus, when you view the query in **SQL text** or download it, Performance Insights returns the same number of bytes.

If you use the AWS CLI or API, Performance Insights doesn't have the 4 KB limit enforced by the console. `DescribeDimensionKeys` and `GetResourceMetrics` return at most 500 bytes.

> **Note**
>
> `GetDimensionKeyDetails` returns the full query, but the size is subject to the engine limit.

Setting the SQL text limit for Amazon RDS for PostgreSQL DB instances

Amazon RDS for PostgreSQL handles text differently. You can set the text size limit with the DB instance parameter `track_activity_query_size`. This parameter has the following characteristics:

Default text size

On Amazon RDS for PostgreSQL version 9.6, the default setting for the `track_activity_query_size` parameter is 1,024 bytes. On Amazon RDS for PostgreSQL version 10 or higher, the default is 4,096 bytes.

Maximum text size

The limit for `track_activity_query_size` is 102,400 bytes for Amazon RDS for PostgreSQL version 12 and lower. The maximum is 1 MB for version 13 and higher.

If the engine returns 1 MB to Performance Insights, the console displays only the first 4 KB. If you download the query, you get the full 1 MB. In this case, viewing and downloading return different numbers of bytes. For more information about the `track_activity_query_size` DB instance parameter, see Run-time Statistics in the PostgreSQL documentation.

To increase the SQL text size, increase the `track_activity_query_size` limit. To modify the parameter, change the parameter setting in the parameter group that is associated with the Amazon RDS for PostgreSQL DB instance.

To change the setting when the instance uses the default parameter group

1. Create a new DB instance parameter group for the appropriate DB engine and DB engine version.
2. Set the parameter in the new parameter group.
3. Associate the new parameter group with the DB instance.

For information about setting a DB instance parameter, see Modifying parameters in a DB parameter group in Amazon RDS.

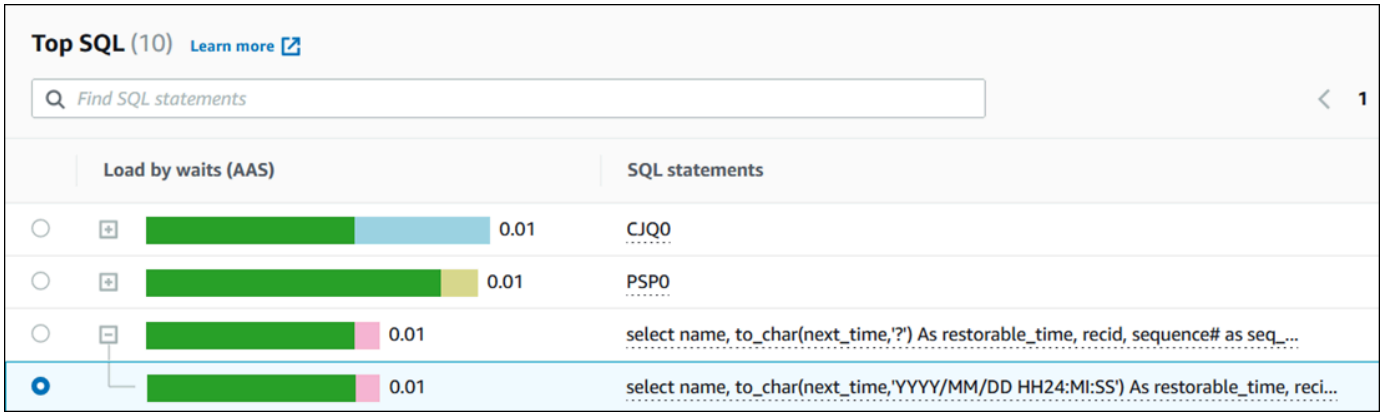
Viewing and downloading SQL text in the Performance Insights dashboard

In the Performance Insights dashboard, you can view or download SQL text.

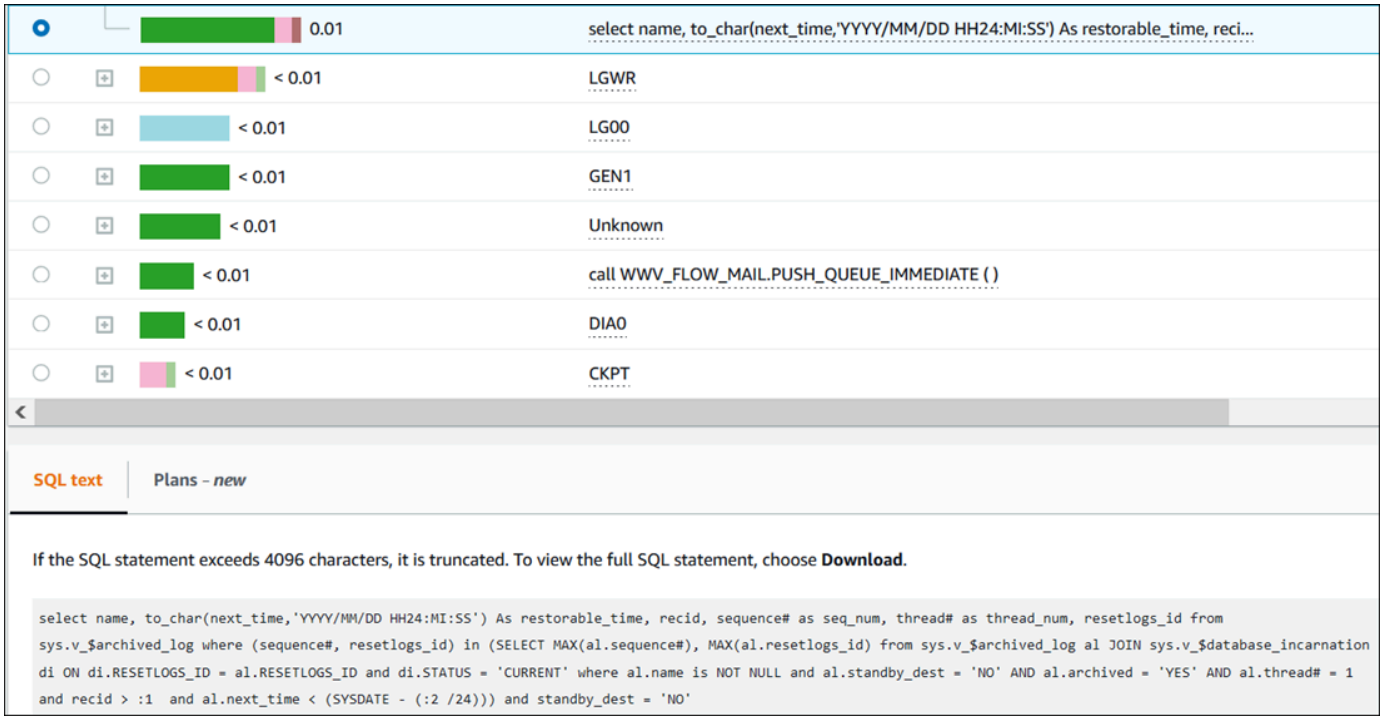
To view more SQL text in the Performance Insights dashboard

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance.
4. Scroll down to the **Top SQL** tab in the Performance Insights dashboard.
5. Choose the plus sign to expand a SQL digest and choose one of the digest's child queries.

   SQL statements with text larger than 500 bytes look similar to the following image.



6. Scroll down to the **SQL text** tab.



The Performance Insights dashboard can display up to 4,096 bytes for each SQL statement.

7. (Optional) Choose **Copy** to copy the displayed SQL statement, or choose **Download** to download the SQL statement to view the SQL text up to the DB engine limit.

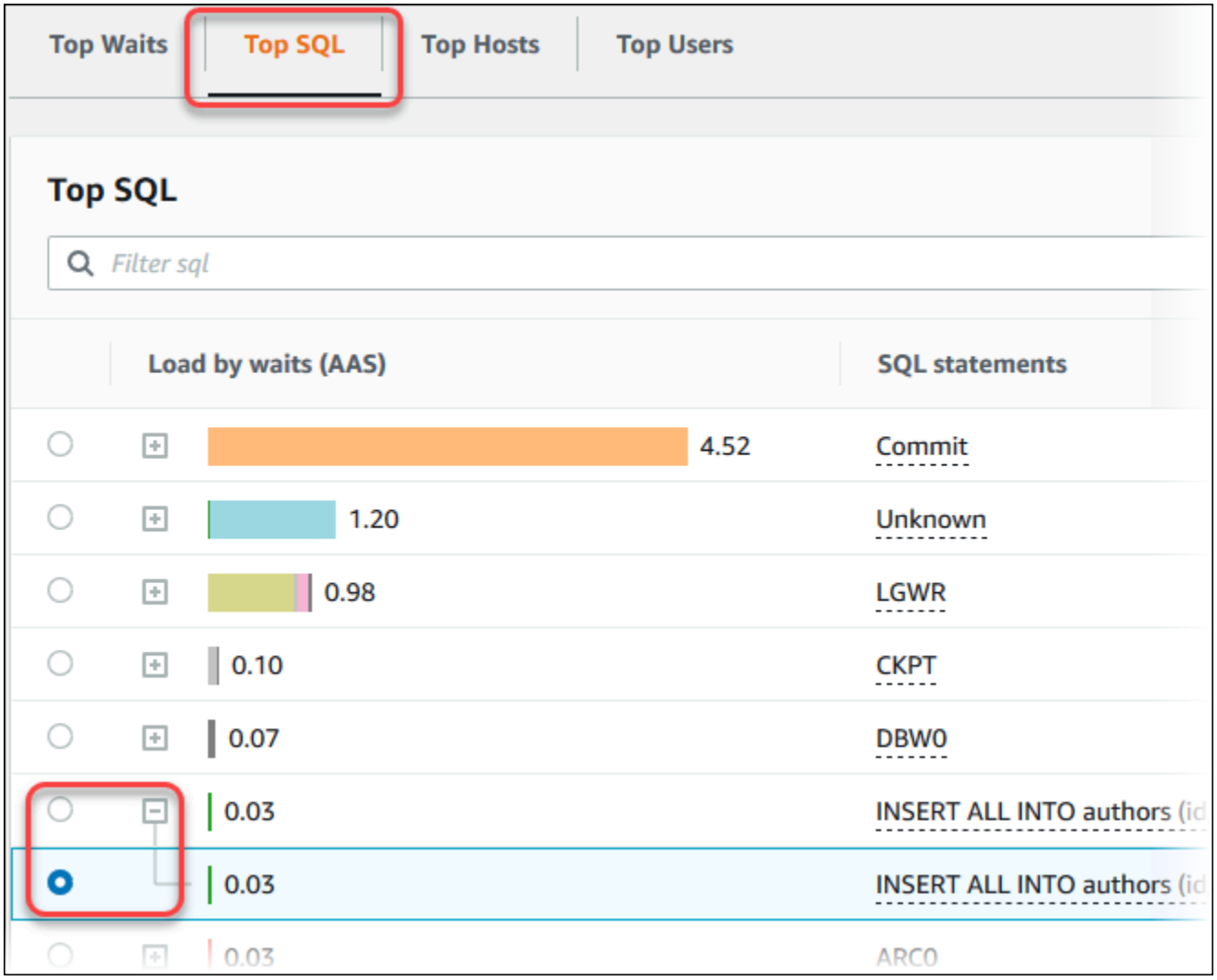
> **Note**
>
> To copy or download the SQL statement, disable pop-up blockers.

Viewing SQL statistics in the Performance Insights dashboard

In the Performance Insights dashboard, SQL statistics are available in the **Top SQL** tab of the **Database load** chart.

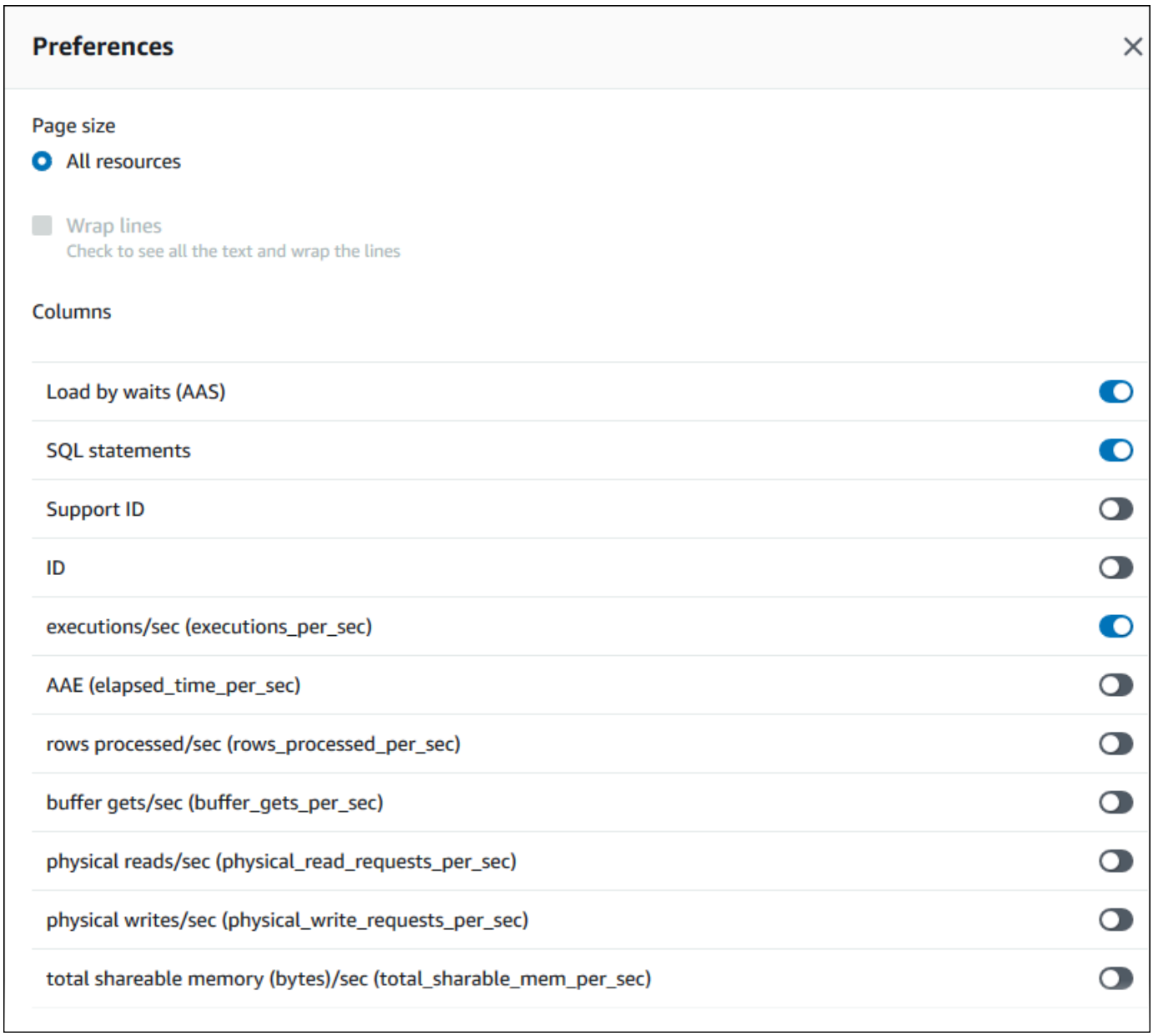
To view SQL statistics

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Performance Insights**.
3. At the top of the page, choose the database whose SQL statistics you want to see.
4. Scroll to the bottom of the page and choose the **Top SQL** tab.
5. Choose an individual statement or digest query.

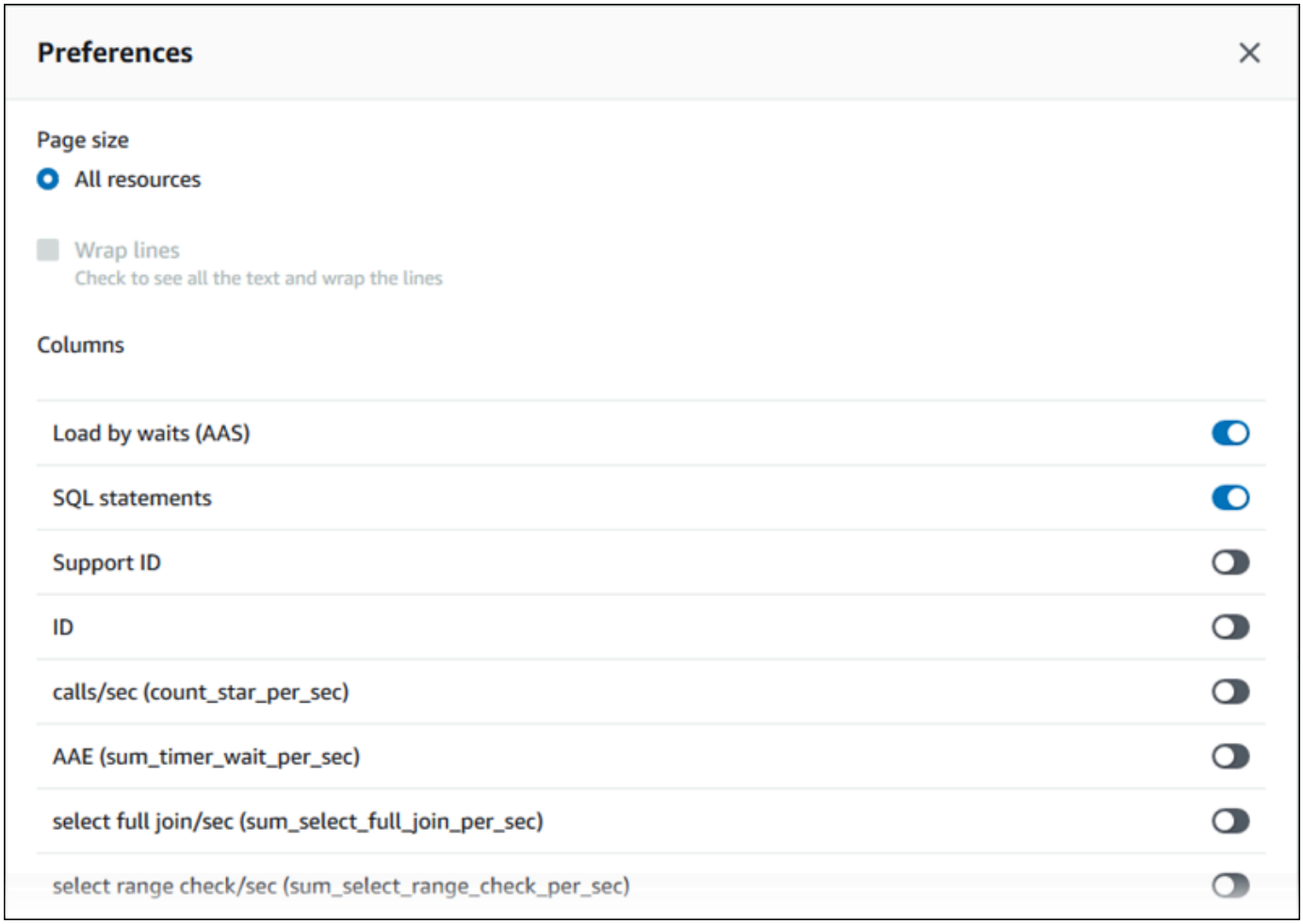


6. Choose which statistics to display by choosing the gear icon in the upper-right corner of the chart. For descriptions of the SQL statistics for the Amazon RDS engines, see SQL statistics for Performance Insights.

   The following example shows the statistics preferences for Oracle DB instances.



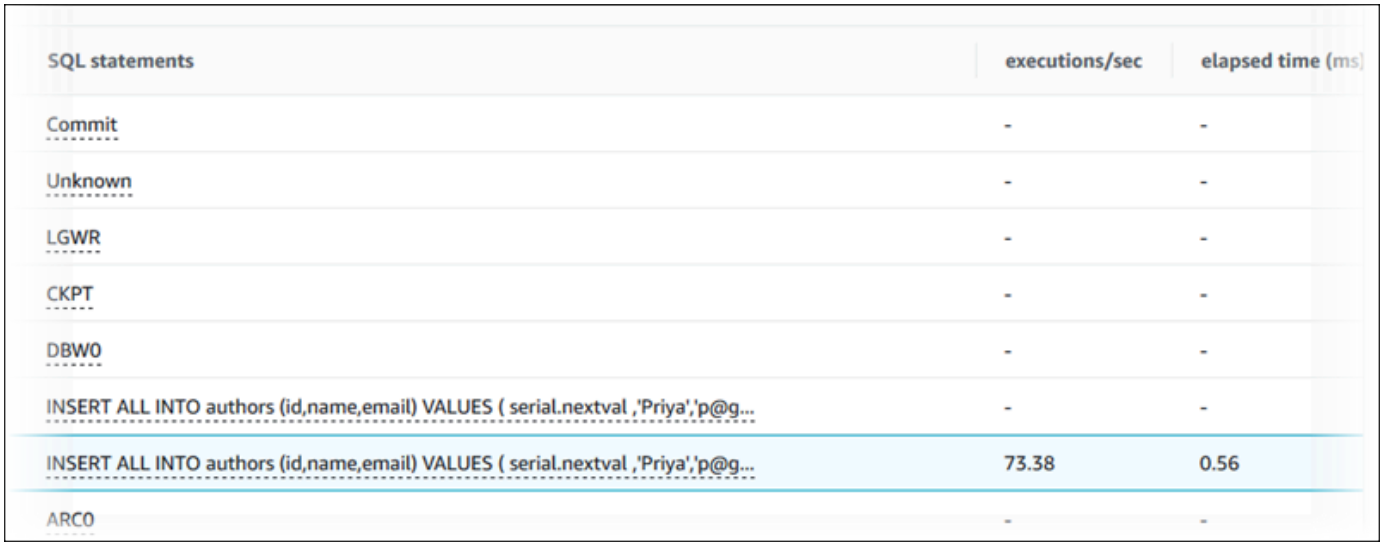
The following example shows the preferences for MariaDB and MySQL DB instances.



7. Choose Save to save your preferences.

   The **Top SQL** table refreshes.

   The following example shows statistics for an Oracle SQL query.



SQL statements	executions/sec	elapsed time (ms)
Commit	-	-
Unknown	-	-
LGWR	-	-
CKPT	-	-
DBW0	-	-
INSERT ALL INTO authors (id,name,email) VALUES (serial.nextval ,'Priya','p@g...	-	-
INSERT ALL INTO authors (id,name,email) VALUES (serial.nextval ,'Priya','p@g...	73.38	0.56
ARC0	-	-

Analyzing top Oracle PDB load

When analyzing the load on an Oracle container DB (CDB), you might want to identify which pluggable databases (PDBs) contribute the most to DB load. You might also want to compare the performance of individual PDBs that are running similar queries to fine tune performance. For more information about Oracle CDBs, see RDS for Oracle database architecture.

In the Amazon RDS Performance Insights dashboard, you can find information about pluggable databases (PDBs) under **Top PDB** tab in the **Dimensions** tab.

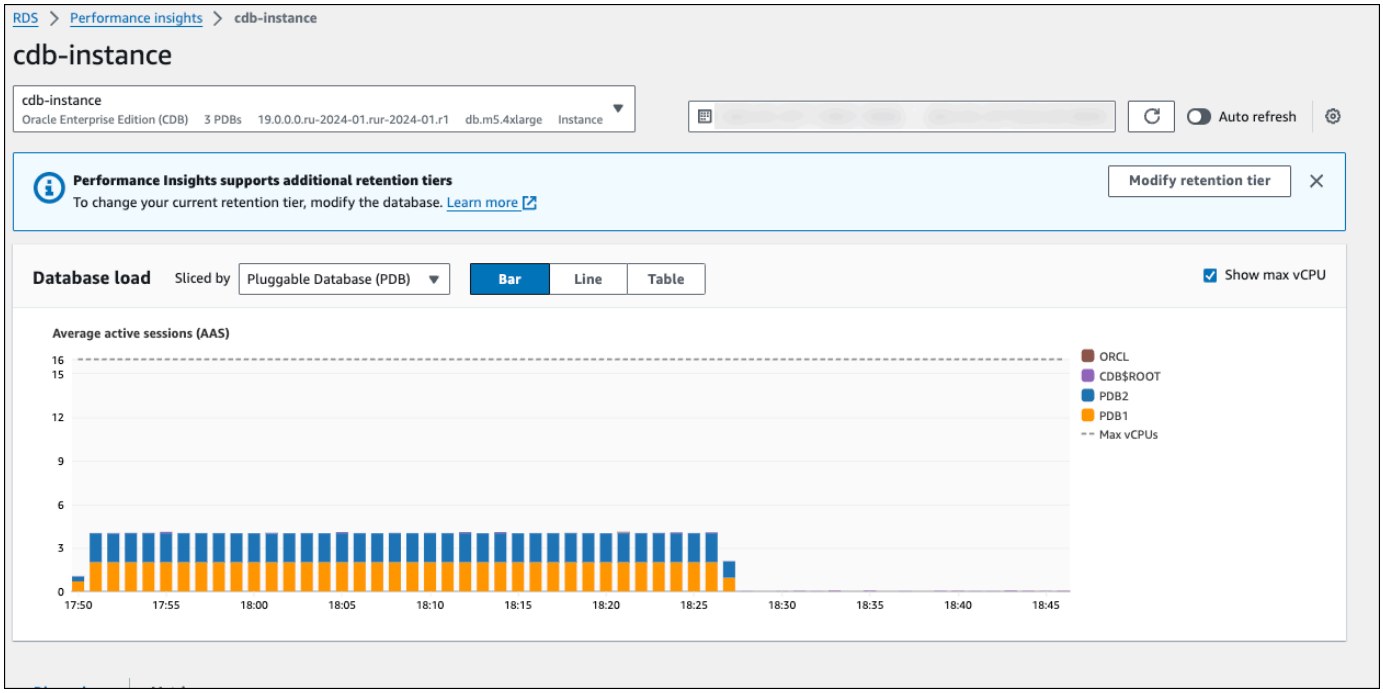
For the region, DB engine, and instance class support information for this feature, see Amazon RDS DB engine, Region, and instance class support for Performance Insights features.

To analyze Top PDB load in an Oracle CDB

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, select **Performance Insights**.
3. Choose an Oracle CDB instance.

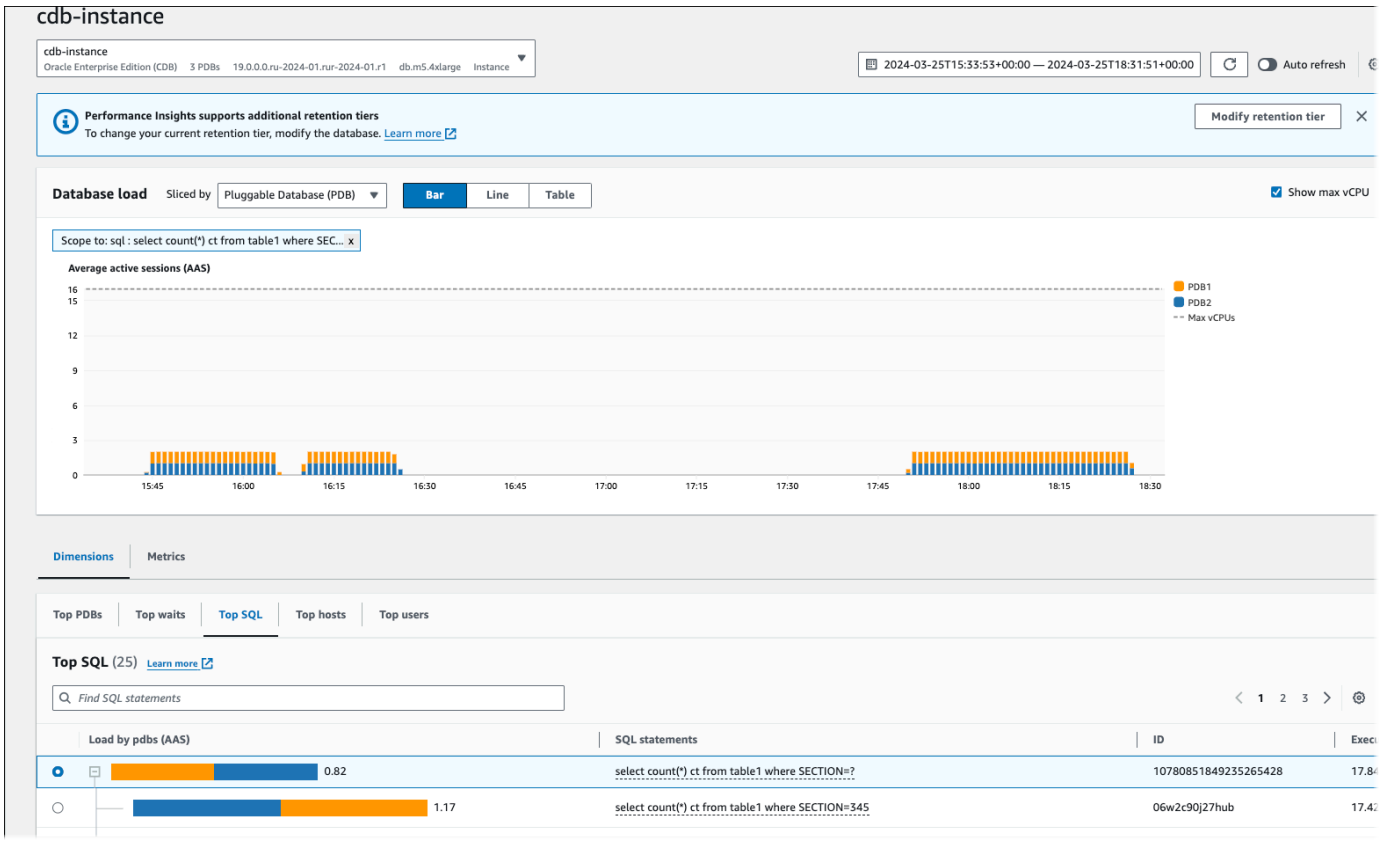
   The Performance Insights dashboard appears for the DB instance.
4. In the **Database load (DB load)** section, choose **Pluggable database (PDB)** next to Slice by.

   The Average active sessions chart shows the PDB with the highest load. The PDB identifiers appear to the right of the color-coded squares. Each identifier uniquely identifies a PDB.

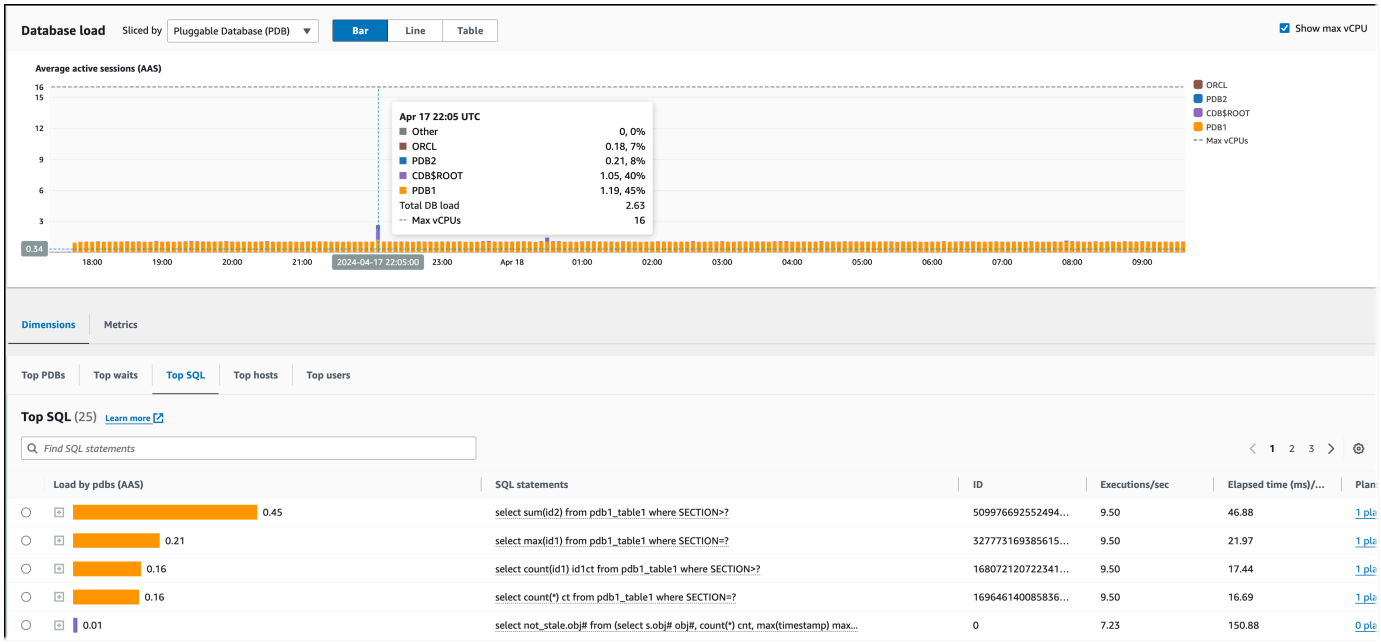


5. Scroll down to the **Top SQL** tab.

   In the following example, you can see the same SQL query and the load it drives to multiple PDBs.



In the following example, a single PDB is handling higher load than other PDBs in the CDB.



For more information about Oracle CDBs, see CDBs and PDBs.

Analyzing execution plans using the Performance Insights dashboard for Amazon RDS

In the Amazon RDS Performance Insights dashboard, you can find information about execution plans for Oracle and SQL Server DB instances. You can use this information to know which plans contribute the most to DB load.

To analyze Oracle or SQL Server execution plans, see the following topics.

Analyzing execution plans

- Analyzing Oracle execution plans using the Performance Insights dashboard for Amazon RDS
- Analyzing SQL Server execution plans using the Performance Insights dashboard for Amazon RDS

Overview of analyzing execution plans for Amazon RDS

You can use the Amazon RDS Performance Insights dashboard to know which plans contribute the most to DB load for Oracle and SQL Server DB instances.

For example, the top SQL statements at a given time might be using the plans shown in the following table.

Top SQL	Plan
SELECT SUM(amount_sold) FROM sales WHERE prod_id = 10	Plan A
SELECT SUM(amount_sold) FROM sales WHERE prod_id = 521	Plan B
SELECT SUM(s_total) FROM sales WHERE region = 10	Plan A
SELECT * FROM emp WHERE emp_id = 1000	Plan C
SELECT SUM(amount_sold) FROM sales WHERE prod_id = 72	Plan A

With the plan feature of Performance Insights, you can do the following:

- Find out which plans are used by the top SQL queries.

  For example, you might find out that most of the DB load is generated by queries using plan A and plan B, with only a small percentage using plan C.
- Compare different plans for the same query.

  In the preceding example, three queries are identical except for the product ID. Two queries use plan A, but one query uses plan B. To see the difference in the two plans, you can use Performance Insights.
- Find out when a query switched to a new plan.

  You might see that a query used plan A and then switched to plan B at a certain time. Was there a change in the database at this point? For example, if a table is empty, the optimizer might choose a full table scan. If the table is loaded with a million rows, the optimizer might switch to an index range scan.
- Drill down to the specific steps of a plan with the highest cost.

For example, the for a long-running query might show a missing a join condition in an equi-join. This missing condition forces a Cartesian join, which joins all rows of two tables.

You can perform the preceding tasks by using the plan capture feature of Performance Insights. Just as you can slice queries by wait events and top SQL, you can slice them by the plan dimension.

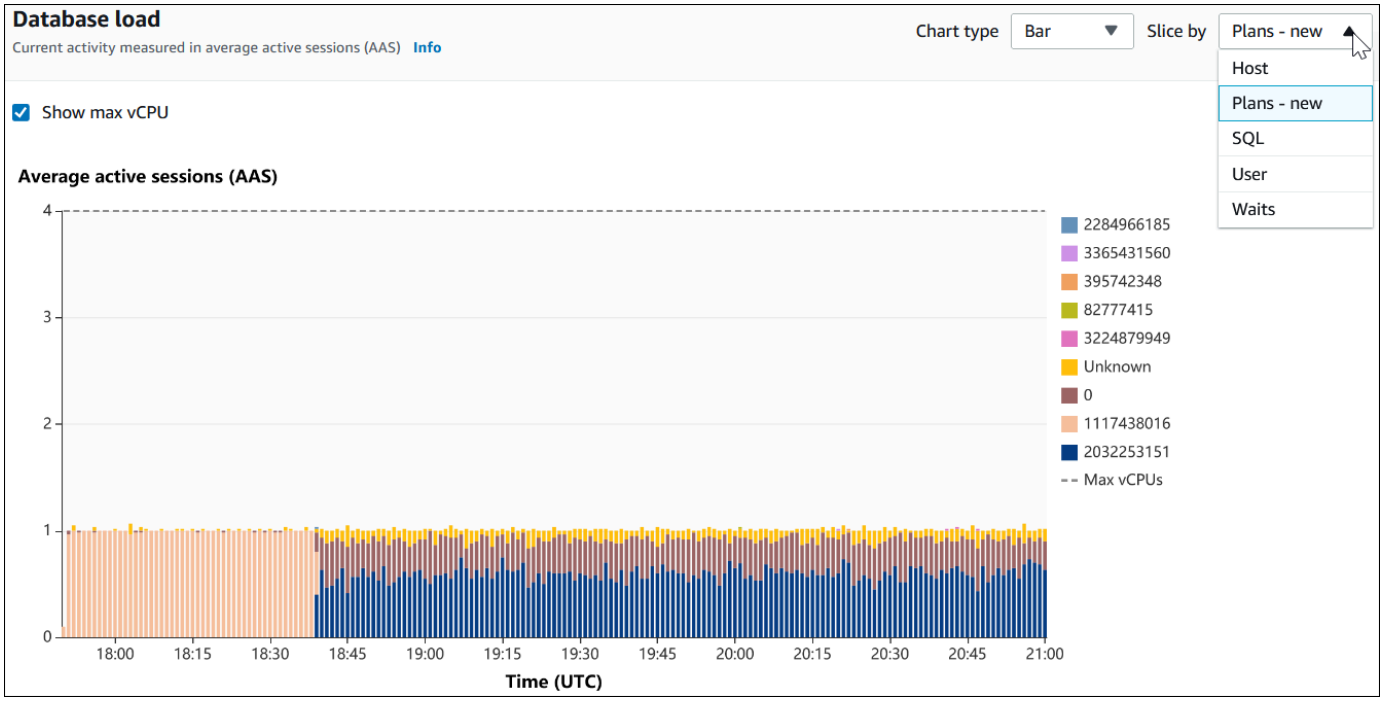
Analyzing Oracle execution plans using the Performance Insights dashboard for Amazon RDS

When analyzing DB load on an Oracle Database, you might want to know which plans are contributing the most to DB load. You can determine which plans are contributing the most to DB load by using the plan capture feature of Performance Insights.

To analyze Oracle execution plans using the console

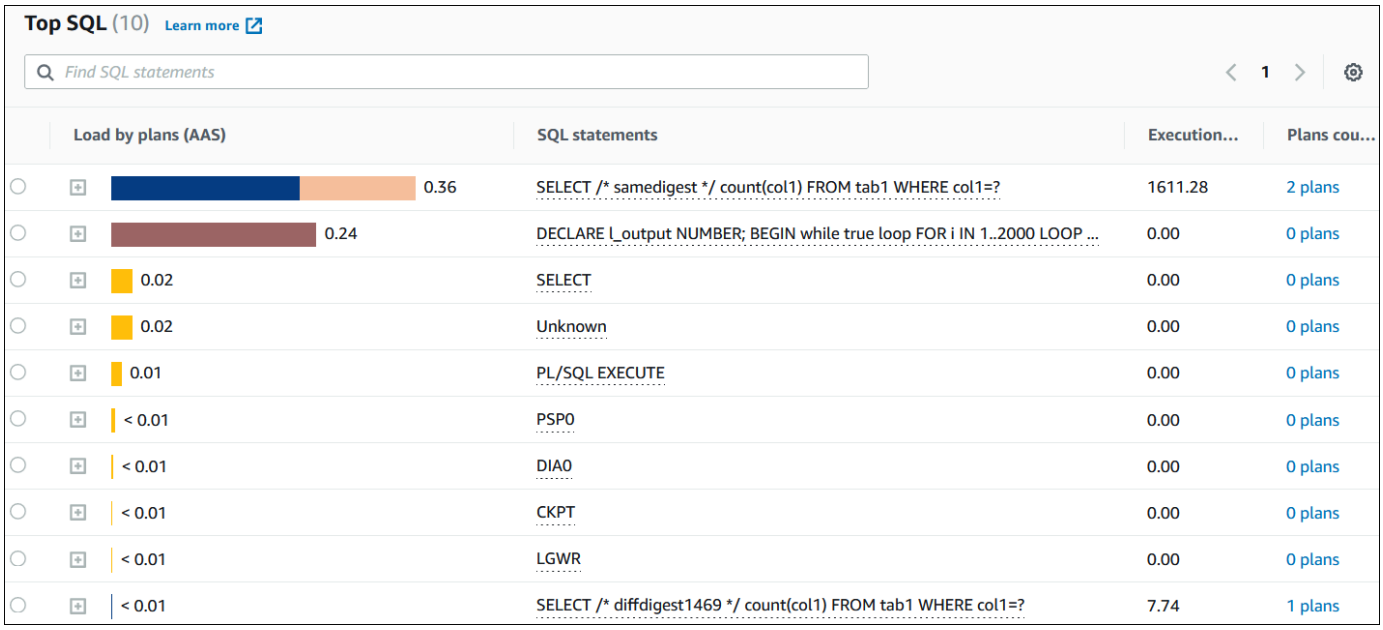
1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Performance Insights**.
3. Choose an Oracle DB instance. The Performance Insights dashboard is displayed for that DB instance.
4. In the **Database load (DB load)** section, choose **Plans** next to **Slice by**.

   The Average active sessions chart shows the plans used by your top SQL statements. The plan hash values appear to the right of the color-coded squares. Each hash value uniquely identifies a plan.



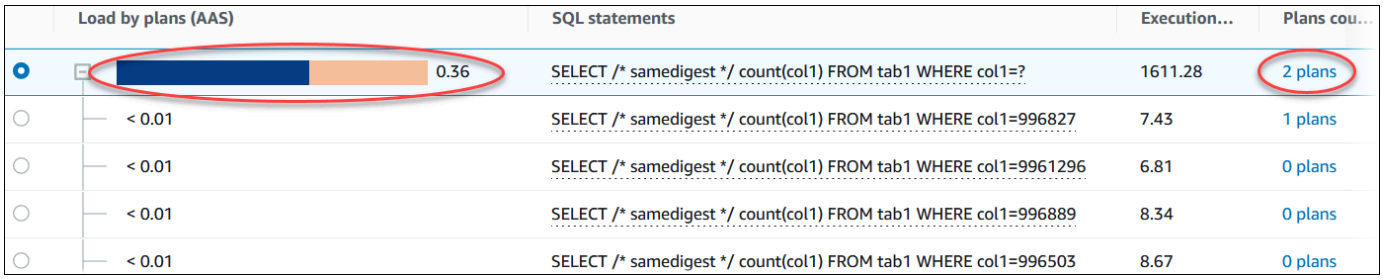
5. Scroll down to the **Top SQL** tab.

   In the following example, the top SQL digest has two plans. You can tell that it's a digest by the question mark in the statement.



6. Choose the digest to expand it into its component statements.

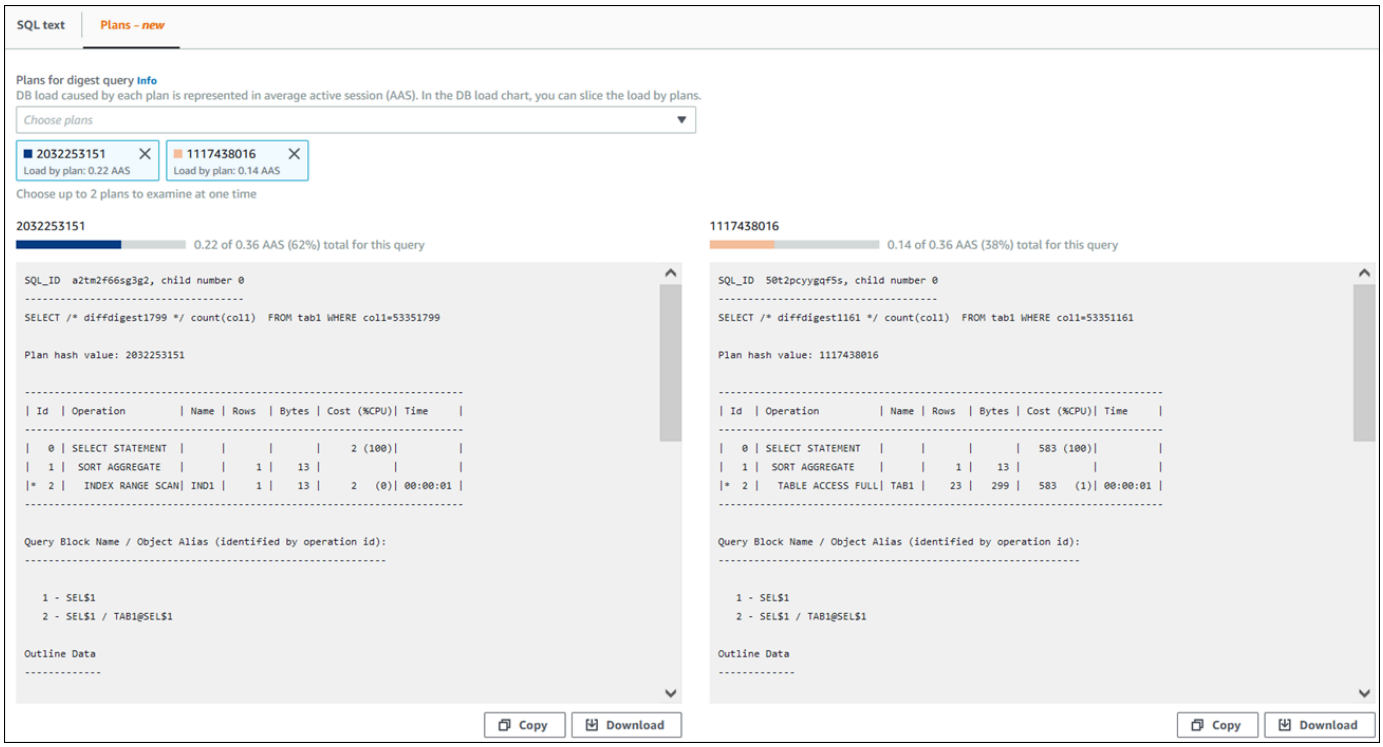
In the following example, the SELECT statement is a digest query. The component queries in the digest use two different plans. The colors of the plans correspond to the database load chart. The total number of plans in the digest is shown in the second column.



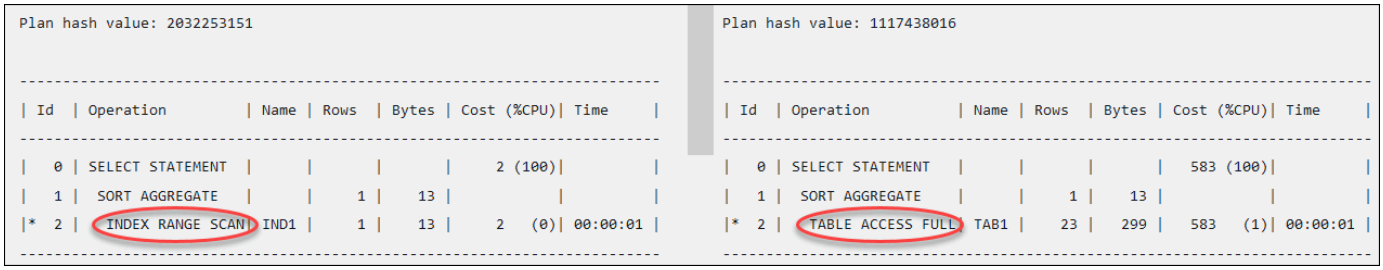
Load by plans (AAS)	SQL statements	Execution...	Plans cou...
0.36	SELECT /* samedigest */ count(col1) FROM tab1 WHERE col1=?	1611.28	2 plans
< 0.01	SELECT /* samedigest */ count(col1) FROM tab1 WHERE col1=996827	7.43	1 plans
< 0.01	SELECT /* samedigest */ count(col1) FROM tab1 WHERE col1=9961296	6.81	0 plans
< 0.01	SELECT /* samedigest */ count(col1) FROM tab1 WHERE col1=996889	8.34	0 plans
< 0.01	SELECT /* samedigest */ count(col1) FROM tab1 WHERE col1=996503	8.67	0 plans

7. Scroll down and choose two **Plans** to compare from **Plans for digest query** list.

You can view either one or two plans for a query at a time. The following screenshot compares the two plans in the digest, with hash 2032253151 and hash 1117438016. In the following example, 62% of the average active sessions running this digest query are using the plan on the left, whereas 38% are using the plan on the right.



In this example, the plans differ in an important way. Step 2 in plan 2032253151 uses an index scan, whereas plan 1117438016 uses a full table scan. For a table with a large number of rows, a query of a single row is almost always faster with an index scan.



```
Plan hash value: 2032253151

-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |       |       |     2 (100)|          |
|   1 |  SORT AGGREGATE   |      |     1 |    13 |            |          |
|*  2 |   INDEX RANGE SCAN| IND1 |     1 |    13 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Plan hash value: 1117438016

-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |       |       |   583 (100)|          |
|   1 |  SORT AGGREGATE   |      |     1 |    13 |            |          |
|*  2 |   TABLE ACCESS FULL| TAB1 |    23 |   299 |   583   (1)| 00:00:01 |
-------------------------------------------------------------------------------
```

8. (Optional) Choose **Copy** to copy the plan to the clipboard, or **Download** to save the plan to your hard drive.

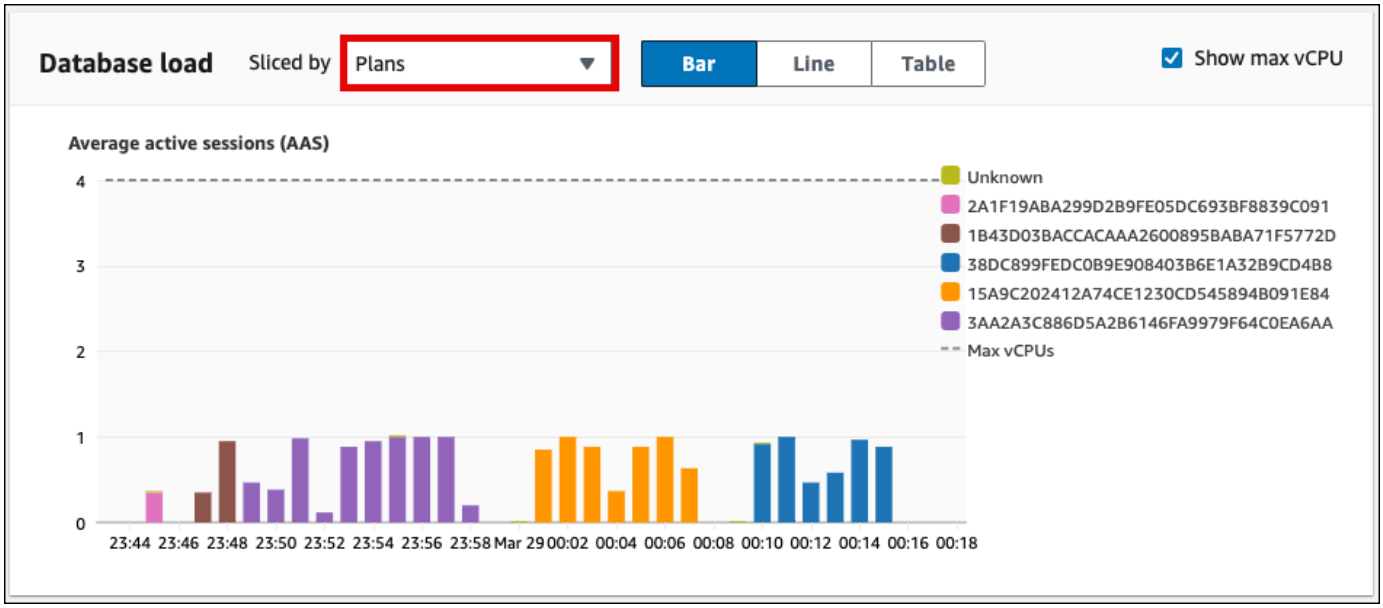
Analyzing SQL Server execution plans using the Performance Insights dashboard for Amazon RDS

When analyzing DB load on a SQL Server Database, you might want to know which plans are contributing the most to DB load. You can determine which plans are contributing the most to DB load by using the plan capture feature of Performance Insights.

To analyze SQL Server execution plans using the console

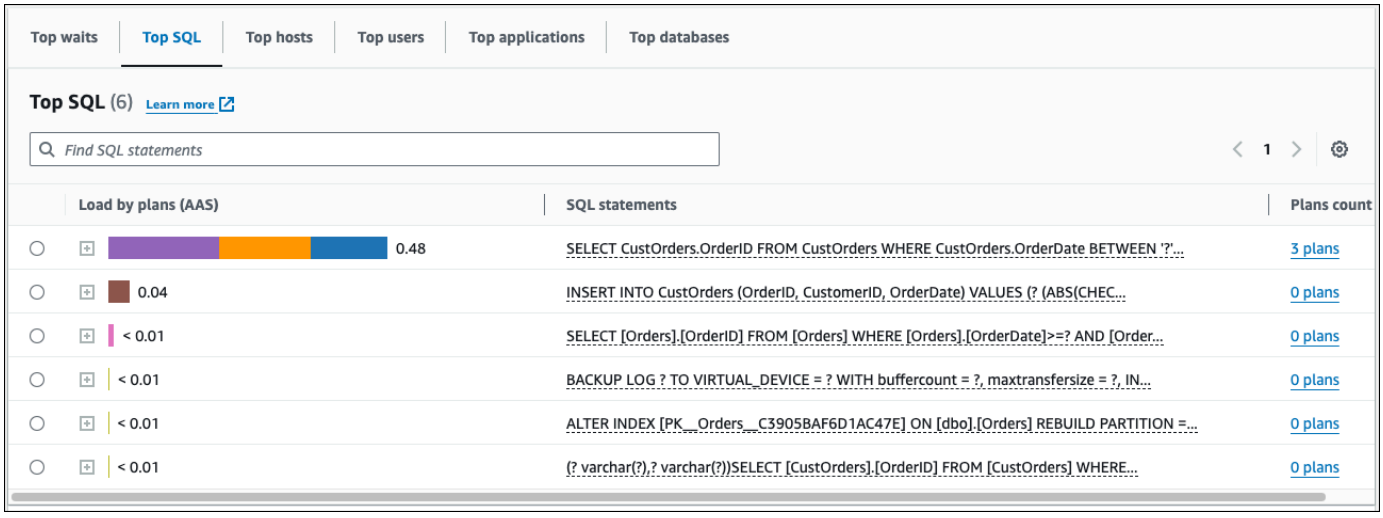
1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a SQL Server DB instance. The Performance Insights dashboard is displayed for that DB instance.
4. In the **Database load (DB load)** section, choose **Plans** next to **Slice by**.

   The Average active sessions chart shows the plans used by your top SQL statements. The plan hash values appear to the right of the color-coded squares. Each hash value uniquely identifies a plan.



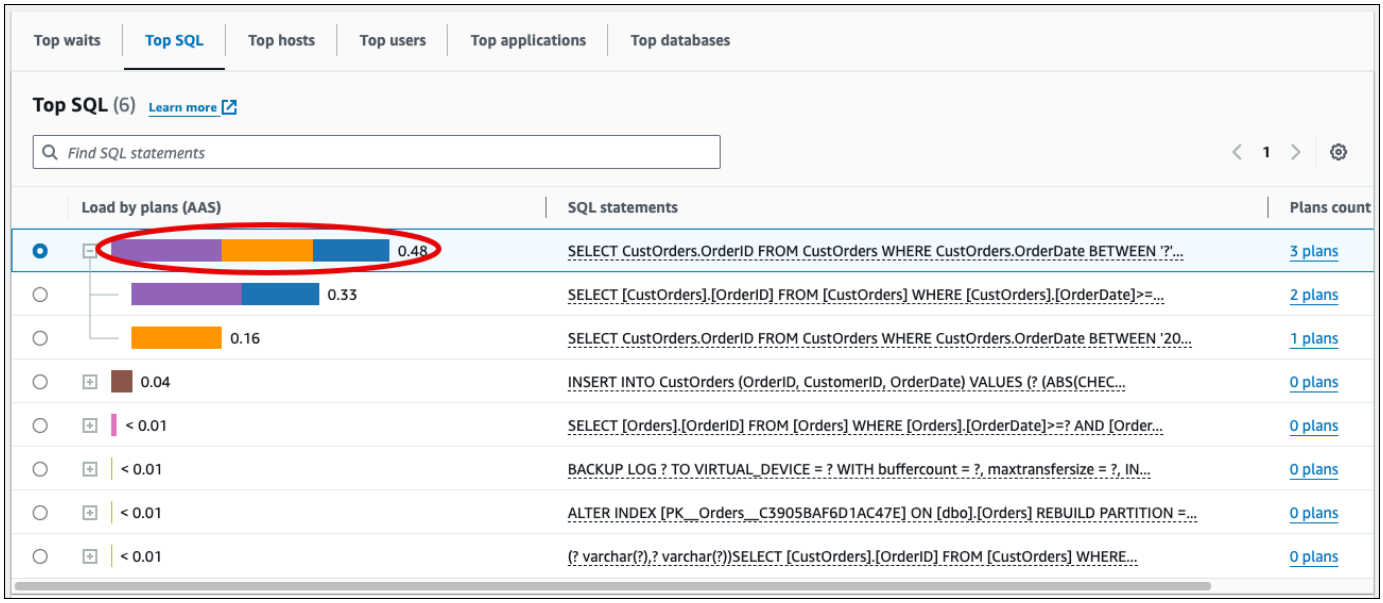
5. Scroll down to the **Top SQL** tab.

   In the following example, the top SQL digest has three plans. The presence of a question mark in the SQL statement indicates that the statement is a digest. To view the full SQL statement, choose a value in the **SQL statements** column.



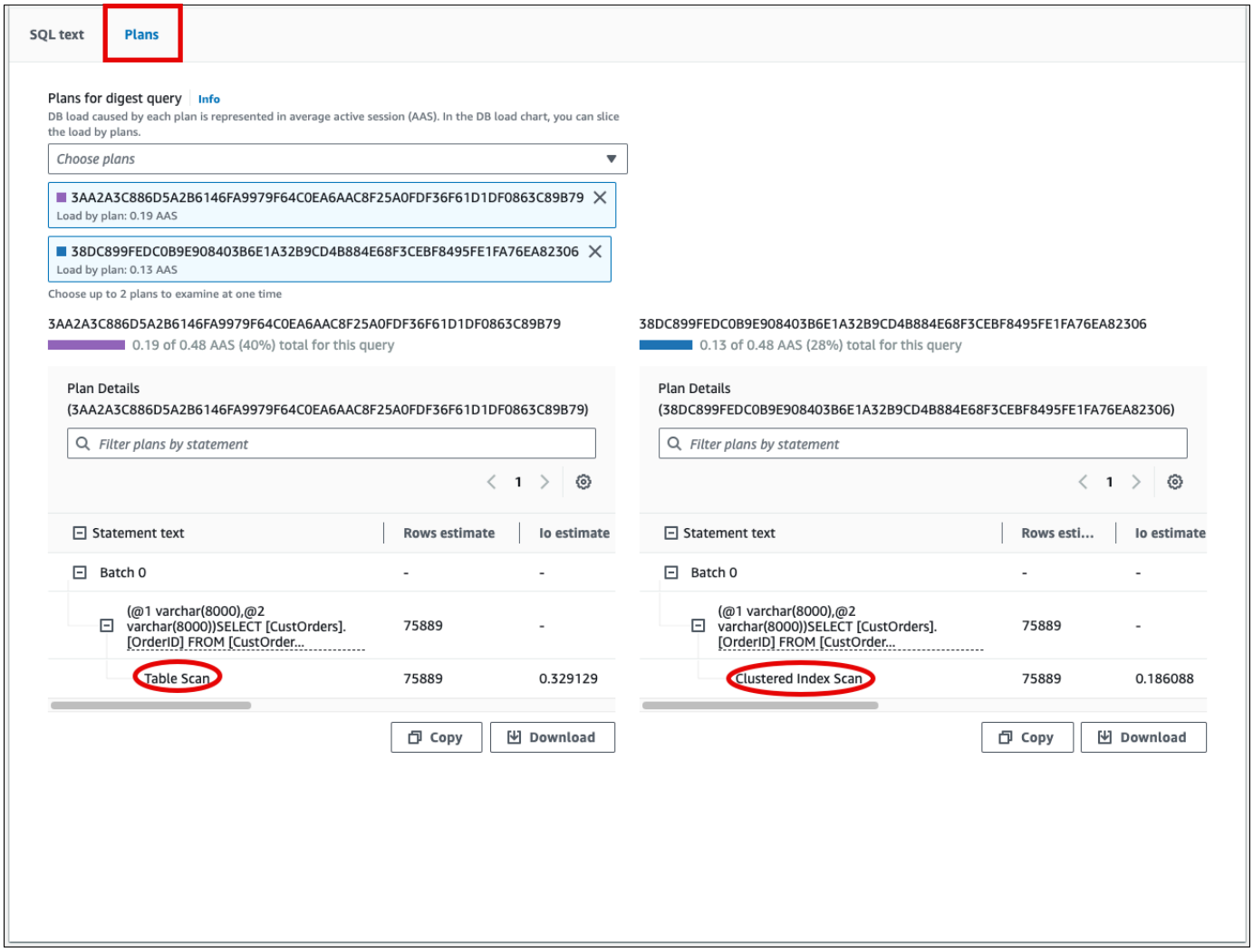
6. Choose the digest to expand it into its component statements.

   In the following example, the SELECT statement is a digest query. The component queries in the digest use three different execution plans. The colors assigned to the plans correspond to the database load chart.



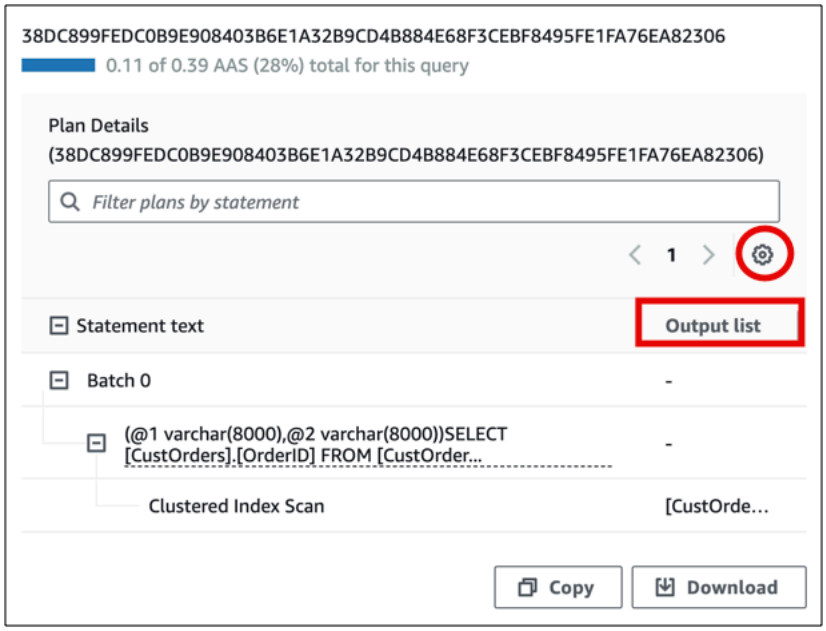
7. Scroll down and choose two **Plans** to compare from **Plans for digest query** list.

   You can view either one or two plans for a query at a time. The following screenshot compares two plans in the digest. In the following example, 40% of the average active sessions running this digest query are using the plan on the left, whereas 28% are using the plan on the right.



In the previous example, the plans differ in an important way. Step 2 in the plan on the left uses an table scan, whereas the plan on the right uses a clustered index scan. For a table with a large number of rows, a query retrieving a single row is almost always faster with a clustered index scan.

8. (Optional) Choose the **Settings** icon on the Plan Details table to customize the visibility and order of columns. The following screenshot shows the Plan Details table with the **Output list** column as the second column.



9. (Optional) Choose **Copy** to copy the plan to the clipboard, or **Download** to save the plan to your hard drive.

> **Note**
>
> Performance Insights displays estimated execution plans using a hierarchical tree table. The table includes the partial execution information for each statement. For more information about the columns in the Plan Details table, see SET SHOWPLAN_ALL in the SQL Server documentation. To display the full execution information for an estimated execution plan, choose **Download** to download the plan and then upload the plan to SQL Server Management Studio. For more information about displaying an estimated execution plan using SQL Server Management Studio, see Display an Estimated Execution Plan in the SQL Server documentation.

Viewing Performance Insights proactive recommendations

Amazon RDS Performance Insights monitors specific metrics and automatically creates thresholds by analyzing what levels might be potentially problematic for a specified resource. When the new metric values cross a predefined threshold over a given period of time, Performance Insights generates a proactive recommendation. This recommendation helps to prevent future database performance impact. To receive these proactive recommendations, you must turn on Performance Insights with a paid tier retention period.

For more information about turning on Performance Insights, see Turning Performance Insights on and off for Amazon RDS. For information about pricing and data retention for Performance Insights, see Pricing and data retention for Performance Insights.

To find out the regions, DB engines, and instance classes supported for the proactive recommendations, see Amazon RDS DB engine, Region, and instance class support for Performance Insights features.

You can view the detailed analysis and recommended investigations of proactive recommendations in the recommendation details page.

For more information about recommendations, see Recommendations from Amazon RDS.

To view the detailed analysis of a proactive recommendation

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, do any of the following:

   - Choose **Recommendations**.

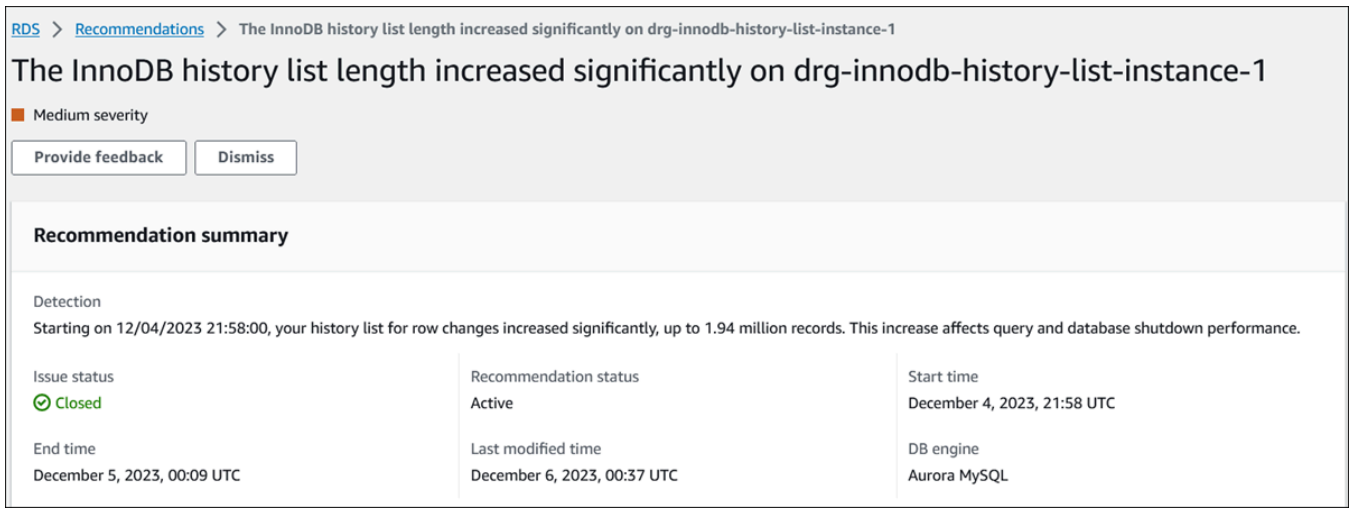
     The **Recommendations** page displays a list of recommendations sorted by the severity for all the resources in your account.
   - Choose **Databases** and then choose **Recommendations** for a resource in the databases page.

     The **Recommendations** tab displays the recommendations and its details for the selected resource.
3. Find a proactive recommendation and choose **View details**.

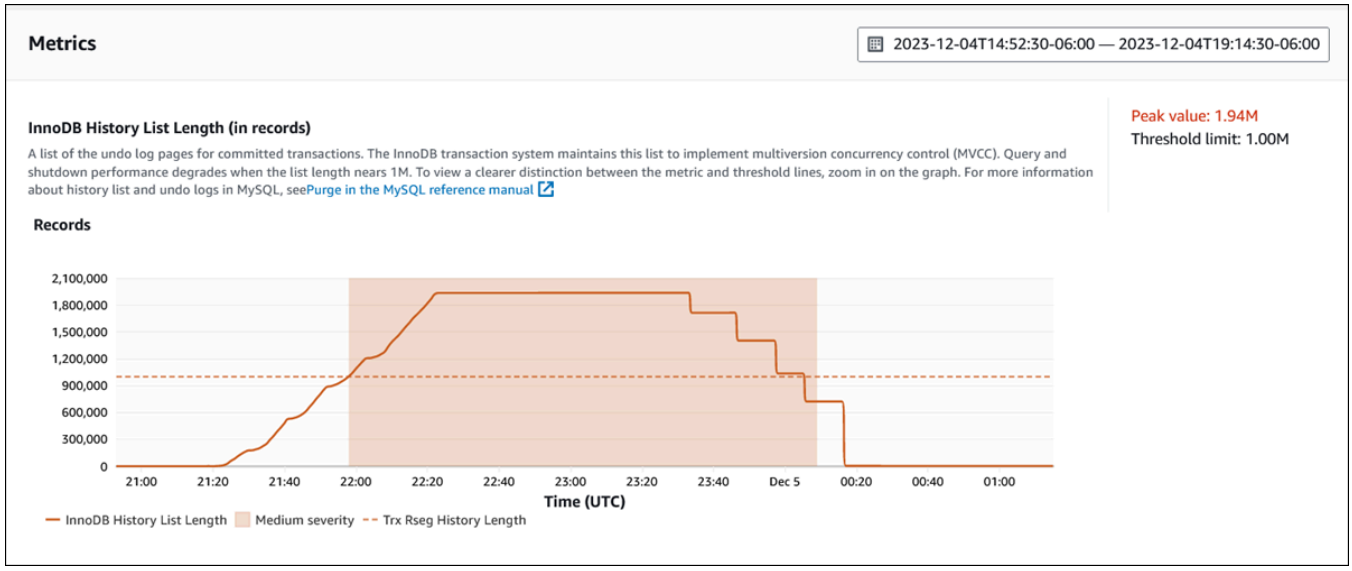
   The recommendation details page appears. The title provides the name of the affected resource with the issue detected and the severity.

   The following are the components on the recommendation details page:

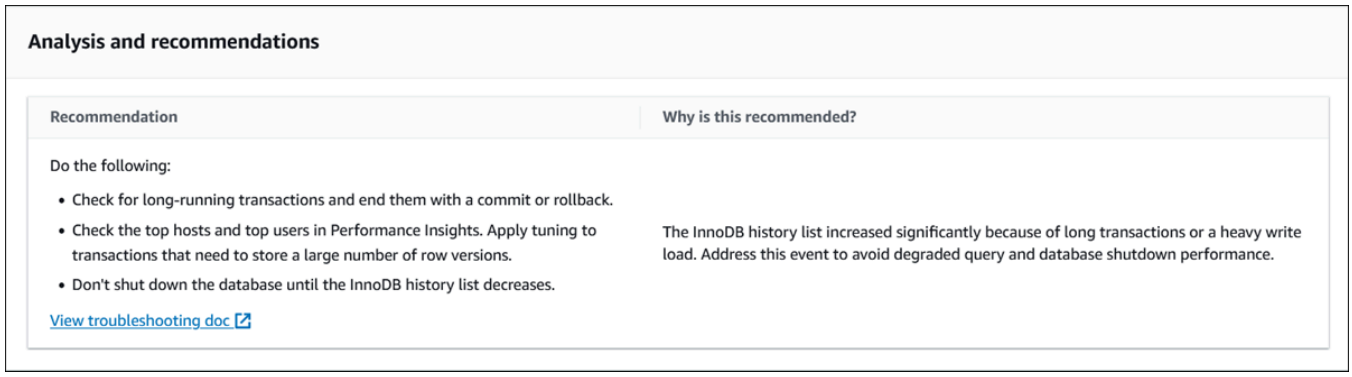
   - **Recommendation summary** – The detected issue, recommendation and issue status, issue start and end time, recommendation modified time, and the engine type.



- **Metrics** – The graphs of the detected issue. Each graph displays a threshold determined by the resource's baseline behavior and data of the metric reported from the issue start time.



- **Analysis and recommendations** – The recommendation and the reason for the suggested recommendation.



You can review the cause of the issue and then perform the suggested recommended actions to fix the issue, or choose **Dismiss** in the upper right to dismiss the recommendation.

Retrieving metrics with the Performance Insights API for Amazon RDS

When Performance Insights is turned on, the API provides visibility into instance performance. Amazon CloudWatch Logs provides the authoritative source for vended monitoring metrics for AWS services.

Performance Insights offers a domain-specific view of database load measured as average active sessions (AAS). This metric appears to API consumers as a two-dimensional time-series dataset. The time dimension of the data provides DB load data for each time point in the queried time range. Each time point decomposes overall load in relation to the requested dimensions, such as SQL, `Wait-event`, `User`, or `Host`, measured at that time point.

Amazon RDS Performance Insights monitors your Amazon RDS DB instance so that you can analyze and troubleshoot database performance. One way to view Performance Insights data is in the AWS Management Console. Performance Insights also provides a public API so that you can query your own data. You can use the API to do the following:

- Offload data into a database
- Add Performance Insights data to existing monitoring dashboards
- Build monitoring tools

To use the Performance Insights API, enable Performance Insights on one of your Amazon RDS DB instances. For information about enabling Performance Insights, see Turning Performance Insights on and off for Amazon RDS. For more information about the Performance Insights API, see the Amazon RDS Performance Insights API Reference.

The Performance Insights API provides the following operations.

Performance Insights action	AWS CLI command	Description
`CreatePerformanceAnalysisReport`	`aws pi create-performance-analysis-report`	Creates a performance analysis report for a specific time period for the DB

Performance Insights action	AWS CLI command	Description
		instance. The result is `AnalysisReportId` which is the unique identifier of the report.
DeletePerformanceAnalysisReport	aws pi delete-performance-analysis-report	Deletes a performance analysis report.
DescribeDimensionKeys	aws pi describe-dimension-keys	Retrieves the top N dimension keys for a metric for a specific time period.
GetDimensionKeyDetails	aws pi get-dimension-key-details	Retrieves the attributes of the specified dimension group for a DB instance or data source. For example, if you specify a SQL ID, and if the dimension details are available, `GetDimensionKeyDetails` retrieves the full text of the dimension `db.sql.statement` associated with this ID. This operation is useful because `GetResourceMetrics` and `DescribeDimensionKeys` don't support retrieval of large SQL statement text.
GetPerformanceAnalysisReport	aws pi get-performance-analysis-report	Retrieves the report including the insights for the report. The result includes the report status, report ID, report time details, insights, and recommendations.

Performance Insights action	AWS CLI command	Description
GetResourceMetadata	aws pi get-resource-metadata	Retrieve the metadata for different features. For example, the metadata might indicate that a feature is turned on or off on a specific DB instance.
GetResourceMetrics	aws pi get-resource-metrics	Retrieves Performance Insights metrics for a set of data sources over a time period. You can provide specific dimension groups and dimensions, and provide aggregation and filtering criteria for each group.
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	Retrieve the dimensions that can be queried for each specified metric type on a specified instance.
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	Retrieve all available metrics of the specified metric types that can be queried for a specified DB instance.
ListPerformanceAnalysisReports	aws pi list-performance-analysis-reports	Retrieves all the analysis reports available for the DB instance. The reports are listed based on the start time of each report.
ListTagsForResource	aws pi list-tags-for-resource	Lists all the metadata tags added to the resource. The list includes the name and value of the tag.

Performance Insights action	AWS CLI command	Description
TagResource	aws pi tag-resource	Adds metadata tags to the Amazon RDS resource. The tag includes a name and a value.
UntagResource	aws pi untag-resource	Removes the metadata tag from the resource.

For more information about retrieving time-series metrics and AWS CLI examples for Performance Insights, see the following topics.

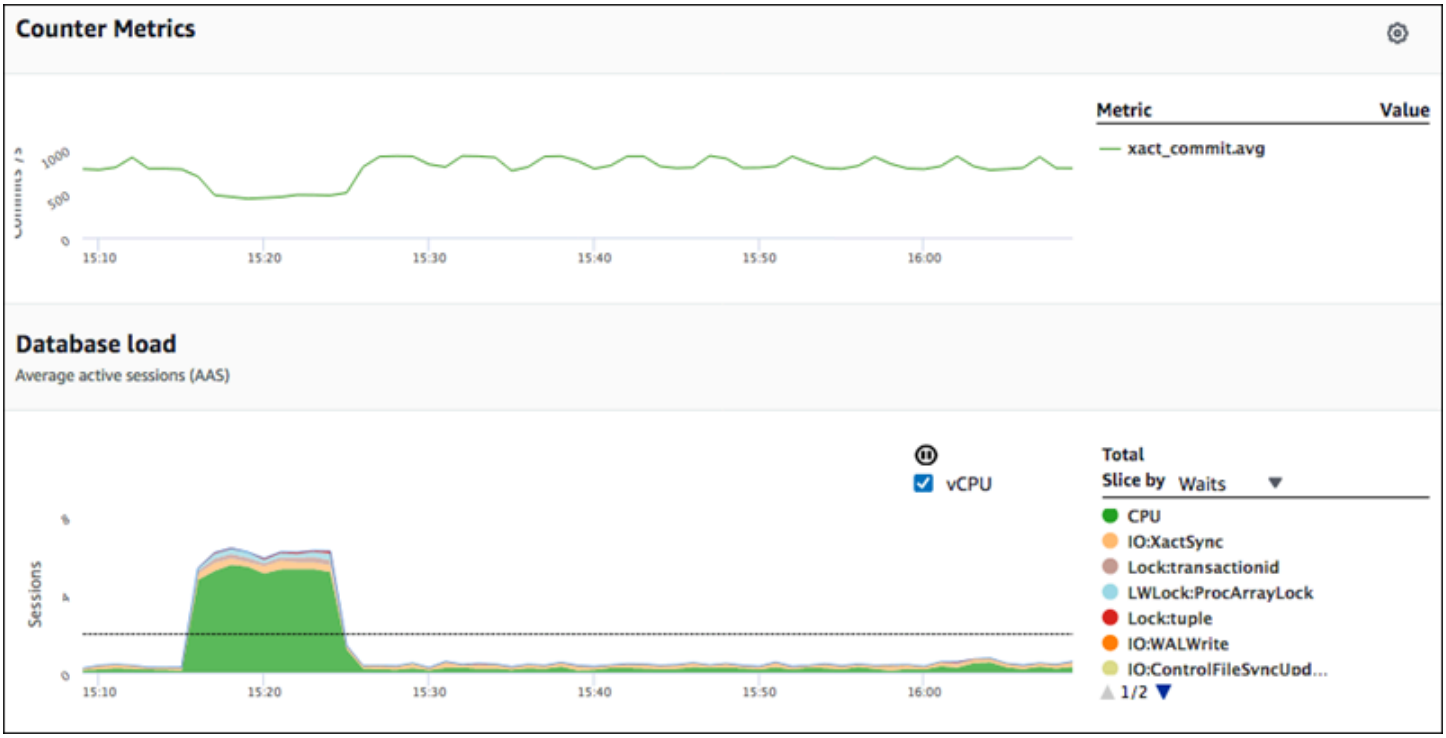
Topics

- Retrieving time-series metrics for Performance Insights
- AWS CLI examples for Performance Insights

Retrieving time-series metrics for Performance Insights

The `GetResourceMetrics` operation retrieves one or more time-series metrics from the Performance Insights data. `GetResourceMetrics` requires a metric and time period, and returns a response with a list of data points.

For example, the AWS Management Console uses `GetResourceMetrics` to populate the **Counter Metrics** chart and the **Database Load** chart, as seen in the following image.



All metrics returned by `GetResourceMetrics` are standard time-series metrics, with the exception of `db.load`. This metric is displayed in the **Database Load** chart. The `db.load` metric is different from the other time-series metrics because you can break it into subcomponents called *dimensions*. In the previous image, `db.load` is broken down and grouped by the waits states that make up the `db.load`.

> **Note**
>
> `GetResourceMetrics` can also return the `db.sampleload` metric, but the `db.load` metric is appropriate in most cases.

For information about the counter metrics returned by `GetResourceMetrics`, see Performance Insights counter metrics.

The following calculations are supported for the metrics:

- Average – The average value for the metric over a period of time. Append `.avg` to the metric name.
- Minimum – The minimum value for the metric over a period of time. Append `.min` to the metric name.

- Maximum – The maximum value for the metric over a period of time. Append `.max` to the metric name.
- Sum – The sum of the metric values over a period of time. Append `.sum` to the metric name.
- Sample count – The number of times the metric was collected over a period of time. Append `.sample_count` to the metric name.

For example, assume that a metric is collected for 300 seconds (5 minutes), and that the metric is collected one time each minute. The values for each minute are 1, 2, 3, 4, and 5. In this case, the following calculations are returned:

- Average – 3
- Minimum – 1
- Maximum – 5
- Sum – 15
- Sample count – 5

For information about using the `get-resource-metrics` AWS CLI command, see `get-resource-metrics`.

For the `--metric-queries` option, specify one or more queries that you want to get results for. Each query consists of a mandatory `Metric` and optional `GroupBy` and `Filter` parameters. The following is an example of a `--metric-queries` option specification.

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

AWS CLI examples for Performance Insights

In the following sections, learn more about the AWS Command Line Interface (AWS CLI) for Performance Insights and use AWS CLI examples.

Topics

- Built-in help for the AWS CLI for Performance Insights
- Retrieving counter metrics
- Retrieving the DB load average for top wait events
- Retrieving the DB load average for top SQL
- Retrieving the DB load average filtered by SQL
- Retrieving the full text of a SQL statement
- Creating a performance analysis report for a time period
- Retrieving a performance analysis report
- Listing all the performance analysis reports for the DB instance
- Deleting a performance analysis report
- Adding tag to a performance analysis report
- Listing all the tags for a performance analysis report
- Deleting tags from a performance analysis report

Built-in help for the AWS CLI for Performance Insights

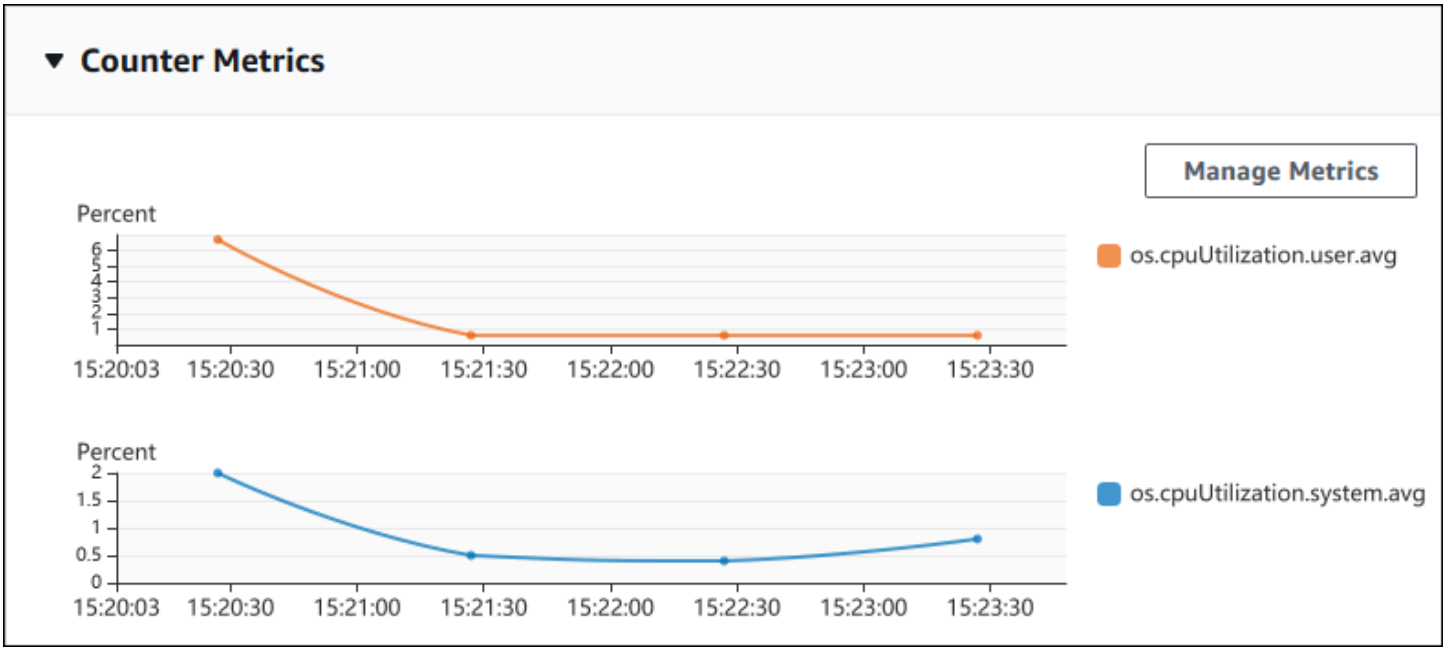
You can view Performance Insights data using the AWS CLI. You can view help for the AWS CLI commands for Performance Insights by entering the following on the command line.

```
aws pi help
```

If you don't have the AWS CLI installed, see Installing the AWS CLI in the *AWS CLI User Guide* for information about installing it.

Retrieving counter metrics

The following screenshot shows two counter metrics charts in the AWS Management Console.



The following example shows how to gather the same data that the AWS Management Console uses to generate the two counter metric charts.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

You can also make a command easier to read by specifying a file for the `--metrics-query` option. The following example uses a file called query.json for the option. The file has the following contents.

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

Run the following command to use the file.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

The preceding example specifies the following values for the options:

- `--service-type` – RDS for Amazon RDS
- `--identifier` – The resource ID for the DB instance
- `--start-time` and `--end-time` – The ISO 8601 `DateTime` values for the period to query, with multiple supported formats

It queries for a one-hour time range:

- `--period-in-seconds` – `60` for a per-minute query
- `--metric-queries` – An array of two queries, each just for one metric.

  The metric name uses dots to classify the metric in a useful category, with the final element being a function. In the example, the function is `avg` for each query. As with Amazon CloudWatch, the supported functions are `min`, `max`, `total`, and `avg`.

The response looks similar to the following.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                "Metric": "os.cpuUtilization.user.avg" //Metric1
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 10.0
                }
                //... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        },
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg" //Metric2
            },
```

```
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 12.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 13.5
                },
                //... 60 datapoints for the os.cpuUtilization.idle.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

The response has an `Identifier`, `AlignedStartTime`, and `AlignedEndTime`. B the `--period-in-seconds` value was `60`, the start and end times have been aligned to the minute. If the `--period-in-seconds` was `3600`, the start and end times would have been aligned to the hour.

The `MetricList` in the response has a number of entries, each with a `Key` and a `DataPoints` entry. Each `DataPoint` has a `Timestamp` and a `Value`. Each `Datapoints` list has 60 data points because the queries are for per-minute data over an hour, with `Timestamp1/Minute1`, `Timestamp2/Minute2`, and so on, up to `Timestamp60/Minute60`.

Because the query is for two different counter metrics, there are two elements in the response `MetricList`.

Retrieving the DB load average for top wait events

The following example is the same query that the AWS Management Console uses to generate a stacked area line graph. This example retrieves the `db.load.avg` for the last hour with load divided according to the top seven wait events. The command is the same as the command in Retrieving counter metrics. However, the query.json file has the following contents.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]
```

Run the following command.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

The example specifies the metric of `db.load.avg` and a `GroupBy` of the top seven wait events. For details about valid values for this example, see DimensionGroup in the *Performance Insights API Reference.*

The response looks similar to the following.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.5166666666666667
```

```
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.38333333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.26666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.35
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.15
                },
                //... 60 datapoints for the CPU key
            ]
        },
        //... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
    ] //end of MetricList
} //end of response
```

In this response, there are eight entries in the `MetricList`. There is one entry for the total `db.load.avg`, and seven entries each for the `db.load.avg` divided according to one of the top seven wait events. Unlike in the first example, because there was a grouping dimension, there must be one key for each grouping of the metric. There can't be only one key for each metric, as in the basic counter metric use case.

Retrieving the DB load average for top SQL

The following example groups `db.wait_events` by the top 10 SQL statements. There are two different groups for SQL statements:

- `db.sql` – The full SQL statement, such as `select * from customers where customer_id = 123`
- `db.sql_tokenized` – The tokenized SQL statement, such as `select * from customers where customer_id = ?`

When analyzing database performance, it can be useful to consider SQL statements that only differ by their parameters as one logic item. So, you can use `db.sql_tokenized` when querying. However, especially when you're interested in explain plans, sometimes it's more useful to examine full SQL statements with parameters, and query grouping by `db.sql`. There is a parent-child relationship between tokenized and full SQL, with multiple full SQL (children) grouped under the same tokenized SQL (parent).

The command in this example is the similar to the command in Retrieving the DB load average for top wait events. However, the query.json file has the following contents.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
    }
]
```

The following example uses `db.sql_tokenized`.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-29T00:00:00Z \
   --end-time   2018-10-30T00:00:00Z \
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-29T00:00:00Z ^
   --end-time   2018-10-30T00:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

This example queries over 24 hours, with a one hour period-in-seconds.

The example specifies the metric of `db.load.avg` and a `GroupBy` of the top seven wait events. For details about valid values for this example, see DimensionGroup in the *Performance Insights API Reference.*

The response looks similar to the following.

```
{
    "AlignedStartTime": 1540771200.0,
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a
 value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
                },
                //... 24 datapoints
            ]
        },
        {
            "Key": { //Next key is the top tokenized SQL
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email)
 VALUES\n( nextval(?)  ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
```

```
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

This response has 11 entries in the `MetricList` (1 total, 10 top tokenized SQL), with each entry having 24 per-hour `DataPoints`.

For tokenized SQL, there are three entries in each dimensions list:

- `db.sql_tokenized.statement` – The tokenized SQL statement.
- `db.sql_tokenized.db_id` – Either the native database ID used to refer to the SQL, or a synthetic ID that Performance Insights generates for you if the native database ID isn't available. This example returns the `pi-2372568224` synthetic ID.
- `db.sql_tokenized.id` – The ID of the query inside Performance Insights.

  In the AWS Management Console, this ID is called the Support ID. It's named this because the ID is data that AWS Support can examine to help you troubleshoot an issue with your database. AWS takes the security and privacy of your data extremely seriously, and almost all data is stored encrypted with your AWS KMS key. Therefore, nobody inside AWS can look at this data. In the example preceding, both the `tokenized.statement` and the `tokenized.db_id` are stored encrypted. If you have an issue with your database, AWS Support can help you by referencing the Support ID.

When querying, it might be convenient to specify a `Group` in `GroupBy`. However, for finer-grained control over the data that's returned, specify the list of dimensions. For example, if all that is needed is the `db.sql_tokenized.statement`, then a `Dimensions` attribute can be added to the query.json file.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.sql_tokenized",
            "Dimensions":["db.sql_tokenized.statement"],
```

```
            "Limit": 10
        }
    }
]
```

Retrieving the DB load average filtered by SQL



The preceding image shows that a particular query is selected, and the top average active sessions stacked area line graph is scoped to that query. Although the query is still for the top seven overall wait events, the value of the response is filtered. The filter causes it to take into account only sessions that are a match for the particular filter.

The corresponding API query in this example is similar to the command in [Retrieving the DB load average for top SQL](). However, the query.json file has the following contents.

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 5  },
        "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

The response looks similar to the following.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1556215200.0,
    "MetricList": [
        {
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.4878117913832196
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.192823803967328
                }
            ]
        },
        {
            "Key": {
```

```
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.1360544217687074
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.058051341890315
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/table/sql/handler"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.16241496598639457
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05163360560093349
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
```

```
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
```

```
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
                }
            ]
        }
    ],
    "AlignedEndTime": 1556222400.0
} //end of response
```

In this response, all values are filtered according to the contribution of tokenized SQL AKIAIOSFODNN7EXAMPLE specified in the query.json file. The keys also might follow a different order than a query without a filter, because it's the top five wait events that affected the filtered SQL.

Retrieving the full text of a SQL statement

The following example retrieves the full text of a SQL statement for DB instance `db-10BCD2EFGHIJ3KL4M5NO6PQRS5`. The `--group` is `db.sql`, and the `--group-identifier` is `db.sql.id`. In this example, *`my-sql-id`* represents a SQL ID retrieved by invoking `pi get-resource-metrics` or `pi describe-dimension-keys`.

Run the following command.

For Linux, macOS, or Unix:

```
aws pi get-dimension-key-details \
   --service-type RDS \
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
   --group db.sql \
   --group-identifier my-sql-id \
   --requested-dimensions statement
```

For Windows:

```
aws pi get-dimension-key-details ^
   --service-type RDS ^
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
   --group db.sql ^
   --group-identifier my-sql-id ^
   --requested-dimensions statement
```

In this example, the dimensions details are available. Thus, Performance Insights retrieves the full text of the SQL statement, without truncating it.

```
{
    "Dimensions":[
    {
        "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
 WHERE e.department_id=d.department_id",
        "Dimension": "db.sql.statement",
        "Status": "AVAILABLE"
    },
    ...
    ]
}
```

Creating a performance analysis report for a time period

The following example creates a performance analysis report with the `1682969503` start time and `1682979503` end time for the `db-loadtest-0` database.

```
aws pi create-performance-analysis-report \
        --service-type RDS \
        --identifier db-loadtest-0 \
        --start-time 1682969503 \
        --end-time 1682979503 \
        --region us-west-2
```

The response is the unique identifier `report-0234d3ed98e28fb17` for the report.

```
{
   "AnalysisReportId": "report-0234d3ed98e28fb17"
}
```

Retrieving a performance analysis report

The following example retrieves the analysis report details for the `report-0d99cc91c4422ee61` report.

```
aws pi get-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
```

```
--region us-west-2
```

The response provides the report status, ID, time details, and insights.

```
        {
    "AnalysisReport": {
        "Status": "Succeeded",
        "ServiceType": "RDS",
        "Identifier": "db-loadtest-0",
        "StartTime": 1680583486.584,
        "AnalysisReportId": "report-0d99cc91c4422ee61",
        "EndTime": 1680587086.584,
        "CreateTime": 1680587087.139,
        "Insights": [
           ... (Condensed for space)
        ]
    }
}
```

Listing all the performance analysis reports for the DB instance

The following example lists all the available performance analysis reports for the `db-loadtest-0` database.

```
aws pi list-performance-analysis-reports \
--service-type RDS \
--identifier db-loadtest-0 \
--region us-west-2
```

The response lists all the reports with the report ID, status, and time period details.

```
{
    "AnalysisReports": [
        {
            "Status": "Succeeded",
            "EndTime": 1680587086.584,
            "CreationTime": 1680587087.139,
            "StartTime": 1680583486.584,
            "AnalysisReportId": "report-0d99cc91c4422ee61"
        },
        {
```

```
            "Status": "Succeeded",
            "EndTime": 1681491137.914,
            "CreationTime": 1681491145.973,
            "StartTime": 1681487537.914,
            "AnalysisReportId": "report-002633115cc002233"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681493499.849,
            "CreationTime": 1681493507.762,
            "StartTime": 1681489899.849,
            "AnalysisReportId": "report-043b1e006b47246f9"
        },
        {
            "Status": "InProgress",
            "EndTime": 1682979503.0,
            "CreationTime": 1682979618.994,
            "StartTime": 1682969503.0,
            "AnalysisReportId": "report-01ad15f9b88bcbd56"
        }
    ]
}
```

Deleting a performance analysis report

The following example deletes the analysis report for the `db-loadtest-0` database.

```
aws pi delete-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--region us-west-2
```

Adding tag to a performance analysis report

The following example adds a tag with a key `name` and value `test-tag` to the `report-01ad15f9b88bcbd56` report.

```
aws pi tag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tags Key=name,Value=test-tag \
```

```
--region us-west-2
```

Listing all the tags for a performance analysis report

The following example lists all the tags for the `report-01ad15f9b88bcbd56` report.

```
aws pi list-tags-for-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--region us-west-2
```

The response lists the value and key for all the tags added to the report:

```
{
    "Tags": [
        {
            "Value": "test-tag",
            "Key": "name"
        }
    ]
}
```

Deleting tags from a performance analysis report

The following example deletes the `name` tag from the `report-01ad15f9b88bcbd56` report.

```
aws pi untag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tag-keys name \
--region us-west-2
```

After the tag is deleted, calling the `list-tags-for-resource` API doesn't list this tag.

Logging Performance Insights calls using AWS CloudTrail

Performance Insights runs with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Performance Insights. CloudTrail captures all API calls for Performance Insights as events. This capture includes calls from the Amazon RDS console and from code calls to the Performance Insights API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Performance Insights. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the data collected by CloudTrail, you can determine certain information. This information includes the request that was made to Performance Insights, the IP address the request was made from, who made the request, and when it was made. It also includes additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Working with Performance Insights information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Performance Insights, that activity is recorded in a CloudTrail event along with other AWS service events in the CloudTrail console in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History in *AWS CloudTrail User Guide.*

For an ongoing record of events in your AWS account, including events for Performance Insights, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following topics in *AWS CloudTrail User Guide:*

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Performance Insights operations are logged by CloudTrail and are documented in the Performance Insights API Reference. For example, calls to the `DescribeDimensionKeys` and `GetResourceMetrics` operations generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Performance Insights log file entries

A *trail* is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An *event* represents a single request from any source. Each event includes information about the requested operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `GetResourceMetrics` operation.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T19:28:46Z",
    "eventSource": "pi.amazonaws.com",
    "eventName": "GetResourceMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.67",
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",
    "requestParameters": {
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",
        "metricQueries": [
            {
                "metric": "os.cpuUtilization.user.avg"
            },
            {
                "metric": "os.cpuUtilization.idle.avg"
```

```
            }
        ],
        "startTime": "Dec 18, 2019 5:28:46 PM",
        "periodInSeconds": 60,
        "endTime": "Dec 18, 2019 7:28:46 PM",
        "serviceType": "RDS"
    },
    "responseElements": null,
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Performance Insights API and interface VPC endpoints (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and Amazon RDS Performance Insights. You can access Performance Insights as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to access Performance Insights.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for Performance Insights.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Considerations for Performance Insights

Before you set up an interface endpoint for Performance Insights, review Considerations in the *AWS PrivateLink Guide*.

Performance Insights supports making calls to all of its API actions through the interface endpoint.

By default, full access to Performance Insights is allowed through the interface endpoint. To control traffic to Performance Insights through the interface endpoint, associate a security group with the endpoint network interfaces.

Availability

Performance Insights API currently supports VPC endpoints in AWS Regions that support Performance Insights. For information about Performance Insights availability, see Supported Regions and DB engines for Performance Insights in Amazon RDS .

Create an interface endpoint for Performance Insights

You can create an interface endpoint for Performance Insights using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

Create an interface endpoint for Performance Insights using the following service name:

If you enable private DNS for the interface endpoint, you can make API requests to Performance Insights using its default Regional DNS name. For example, `pi.us-east-1.amazonaws.com`.

Creating a VPC endpoint policy for Performance Insights API

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to Performance Insights through the interface endpoint. To control the access allowed to Performance Insights from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Example: VPC endpoint policy for Performance Insights actions

The following is an example of a custom endpoint policy. When you attach this policy to your interface endpoint, it grants access to the listed Performance Insights actions for all principals on all resources.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreatePerformanceAnalysisReport",
            "rds:DeletePerformanceAnalysisReport",
            "rds:GetPerformanceAnalysisReport"
         ],
         "Resource":"*"
      }
   ]
}
```

Example: VPC endpoint policy that denies all access from a specified AWS account

The following VPC endpoint policy denies AWS account 123456789012 all access to resources using the endpoint. The policy allows all actions from other accounts.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

IP addressing for Performance Insights

IP addresses enable resources in your VPC to communicate with each other, and with resources over the internet. Performance Insights supports both IPv4 and IPv6 addressing protocols. By default, Performance Insights and Amazon VPC use the IPv4 addressing protocol. You can't turn off

this behavior. When you create a VPC, make sure to specify an IPv4 CIDR block (a range of private IPv4 addresses).

You can optionally assign an IPv6 CIDR block to your VPC and subnets, and assign IPv6 addresses from that block to RDS resources in your subnet. Support for the IPv6 protocol expands the number of supported IP addresses. By using the IPv6 protocol, you ensure that you have sufficient available addresses for the future growth of the internet. New and existing RDS resources can use IPv4 and IPv6 addresses within your VPC. Configuring, securing, and translating network traffic between the two protocols used in different parts of an application can cause operational overhead. You can standardize on the IPv6 protocol for Amazon RDS resources to simplify your network configuration. For more information about service endpoints and quotas, see Amazon Relational Database Service endpoints and quotas.

For more information about Amazon RDS IP addressing, see Amazon RDS IP addressing.

Analyzing performance anomalies with Amazon DevOps Guru for Amazon RDS

Amazon DevOps Guru is a fully managed operations service that helps developers and operators improve the performance and availability of their applications. DevOps Guru offloads the tasks associated with identifying operational issues so that you can quickly implement recommendations to improve your application. For more information, see What is Amazon DevOps Guru? in the *Amazon DevOps Guru User Guide*.

DevOps Guru detects, analyzes, and makes recommendations for existing operational issues for all Amazon RDS DB engines. DevOps Guru for RDS extends this capability by applying machine learning to Performance Insights metrics for RDS for PostgreSQL databases. These monitoring features allow DevOps Guru for RDS to detect and diagnose performance bottlenecks and recommend specific corrective actions. DevOps Guru for RDS can also detect problematic conditions in your RDS for PostgreSQL database before they occur.

You can now view these recommendations in RDS console. For more information, see Recommendations from Amazon RDS.

The following video is an overview of DevOps Guru for RDS.

For a deep dive on this subject, see Amazon DevOps Guru for RDS under the hood.

Topics

- Benefits of DevOps Guru for RDS
- How DevOps Guru for RDS works
- Setting up DevOps Guru for RDS

Benefits of DevOps Guru for RDS

If you're responsible for RDS for PostgreSQL database, you might not know that an event or regression that is affecting that database is occurring. When you learn about the issue, you might not know why it's occurring or what to do about it. Rather than turning to a database administrator (DBA) for help or relying on third-party tools, you can follow recommendations from DevOps Guru for RDS.

You gain the following advantages from the detailed analysis of DevOps Guru for RDS:

Fast diagnosis

DevOps Guru for RDS continuously monitors and analyzes database telemetry. Performance Insights, Enhanced Monitoring, and Amazon CloudWatch collect telemetry data for your database instance. DevOps Guru for RDS uses statistical and machine learning techniques to mine this data and detect anomalies. To learn more about telemetry data, see Monitoring DB load with Performance Insights on Amazon RDS and Monitoring OS metrics with Enhanced Monitoring in the *Amazon RDS User Guide*.

Fast resolution

Each anomaly identifies the performance issue and suggests avenues of investigation or corrective actions. For example, DevOps Guru for RDS might recommend that you investigate specific wait events. Or it might recommend that you tune your application pool settings to limit the number of database connections. Based on these recommendations, you can resolve performance issues more quickly than by troubleshooting manually.

Proactive insights

DevOps Guru for RDS uses metrics from your resources to detect potentially problematic behavior before it becomes a bigger problem. For example, it can detect when your database is using an increasing number of on-disk temporary tables, which could start to impact performance. DevOps Guru then provides recommendations to help you address issues before they become bigger problems.

Deep knowledge of Amazon engineers and machine learning

To detect performance issues and help you resolve bottlenecks, DevOps Guru for RDS relies on machine learning (ML) and advanced mathematical formulas. Amazon database engineers contributed to the development of the DevOps Guru for RDS findings, which encapsulate many years of managing hundreds of thousands of databases. By drawing on this collective knowledge, DevOps Guru for RDS can teach you best practices.

How DevOps Guru for RDS works

DevOps Guru for RDS collects data about your RDS for PostgreSQL databases from Amazon RDS Performance Insights. The most important metric is `DBLoad`. DevOps Guru for RDS consumes the Performance Insights metrics, analyzes them with machine learning, and publishes insights to the dashboard.

An *insight* is a collection of related anomalies that were detected by DevOps Guru.

In DevOps Guru for RDS, an *anomaly* is a pattern that deviates from what is considered normal performance for your RDS for PostgreSQL database.

Proactive insights

A *proactive insight* lets you know about problematic behavior before it occurs. It contains anomalies with recommendations and related metrics to help you address issues in your RDS for PostgreSQL databases before become bigger problems. These insights are published in the DevOps Guru dashboard.

For example, DevOps Guru might detect that your RDS for PostgreSQL database is creating many on-disk temporary tables. If not addressed, this trend might lead to performance issues. Each proactive insight includes recommendations for corrective behavior and links to relevant topics in Tuning RDS for PostgreSQL with Amazon DevOps Guru proactive insights. For more information, see Working with insights in DevOps Guru in the *Amazon DevOps Guru User Guide*.

Reactive insights

A *reactive insight* identifies anomalous behavior as it occurs. If DevOps Guru for RDS finds performance issues in your RDS for PostgreSQL DB instances, it publishes a reactive insight in the DevOps Guru dashboard. For more information, see Working with insights in DevOps Guru in the *Amazon DevOps Guru User Guide*.

Causal anomalies

A *causal anomaly* is a top-level anomaly within a reactive insight. **Database load (DB load)** is the causal anomaly for DevOps Guru for RDS.

An anomaly measures performance impact by assigning a severity level of **High**, **Medium**, or **Low**. To learn more, see Key concepts for DevOps Guru for RDS in the *Amazon DevOps Guru User Guide*.

If DevOps Guru detects a current anomaly on your DB instance, you're alerted in the **Databases** page of the RDS console. The console also alerts you to anomalies that occurred in the past 24 hours. To go to the anomaly page from the RDS console, choose the link in the alert message. The RDS console also alerts you in the page for your RDS for PostgreSQL DB instance.

Contextual anomalies

A *contextual anomaly* is a finding within **Database load (DB load)** that is related to a reactive insight. Each contextual anomaly describes a specific RDS for PostgreSQL performance issue that

requires investigation. For example, DevOps Guru for RDS might recommend that you consider increasing CPU capacity or investigate wait events that are contributing to DB load.

> **Important**
>
> We recommend that you test any changes on a test instance before modifying a production instance. In this way, you understand the impact of the change.

To learn more, see Analyzing anomalies in Amazon RDS in the *Amazon DevOps Guru User Guide*.

Setting up DevOps Guru for RDS

To allow DevOps Guru for Amazon RDS to publish insights for a RDS for PostgreSQL database, complete the following tasks.

Topics

- Configuring IAM access policies for DevOps Guru for RDS
- Turning on Performance Insights for your RDS for PostgreSQL DB instances
- Turning on DevOps Guru and specifying resource coverage

Configuring IAM access policies for DevOps Guru for RDS

To view alerts from DevOps Guru in the RDS console, your AWS Identity and Access Management (IAM) user or role must have either of the following policies:

- The AWS managed policy `AmazonDevOpsGuruConsoleFullAccess`
- The AWS managed policy `AmazonDevOpsGuruConsoleReadOnlyAccess` and either of the following policies:
  - The AWS managed policy `AmazonRDSFullAccess`
  - A customer managed policy that includes `pi:GetResourceMetrics` and `pi:DescribeDimensionKeys`

For more information, see Configuring access policies for Performance Insights.

Turning on Performance Insights for your RDS for PostgreSQL DB instances

DevOps Guru for RDS relies on Performance Insights for its data. Without Performance Insights, DevOps Guru publishes anomalies, but doesn't include the detailed analysis and recommendations.

When you create or modify a RDS for PostgreSQL DB instance, you can turn on Performance Insights. For more information, see Turning Performance Insights on and off for Amazon RDS.

Turning on DevOps Guru and specifying resource coverage

You can turn on DevOps Guru to have it monitor your RDS for PostgreSQL databases in either of the following ways.

Topics

- Turning on DevOps Guru in the RDS console
- Adding RDS for PostgreSQL resources in the DevOps Guru console
- Adding RDS for PostgreSQL resources using CloudFormation

Turning on DevOps Guru in the RDS console

You can take multiple paths in the Amazon RDS console to turn on DevOps Guru.

Topics

- Turning on DevOps Guru when you create an RDS for PostgreSQL database
- Turning on DevOps Guru from the notification banner
- Responding to a permissions error when you turn on DevOps Guru

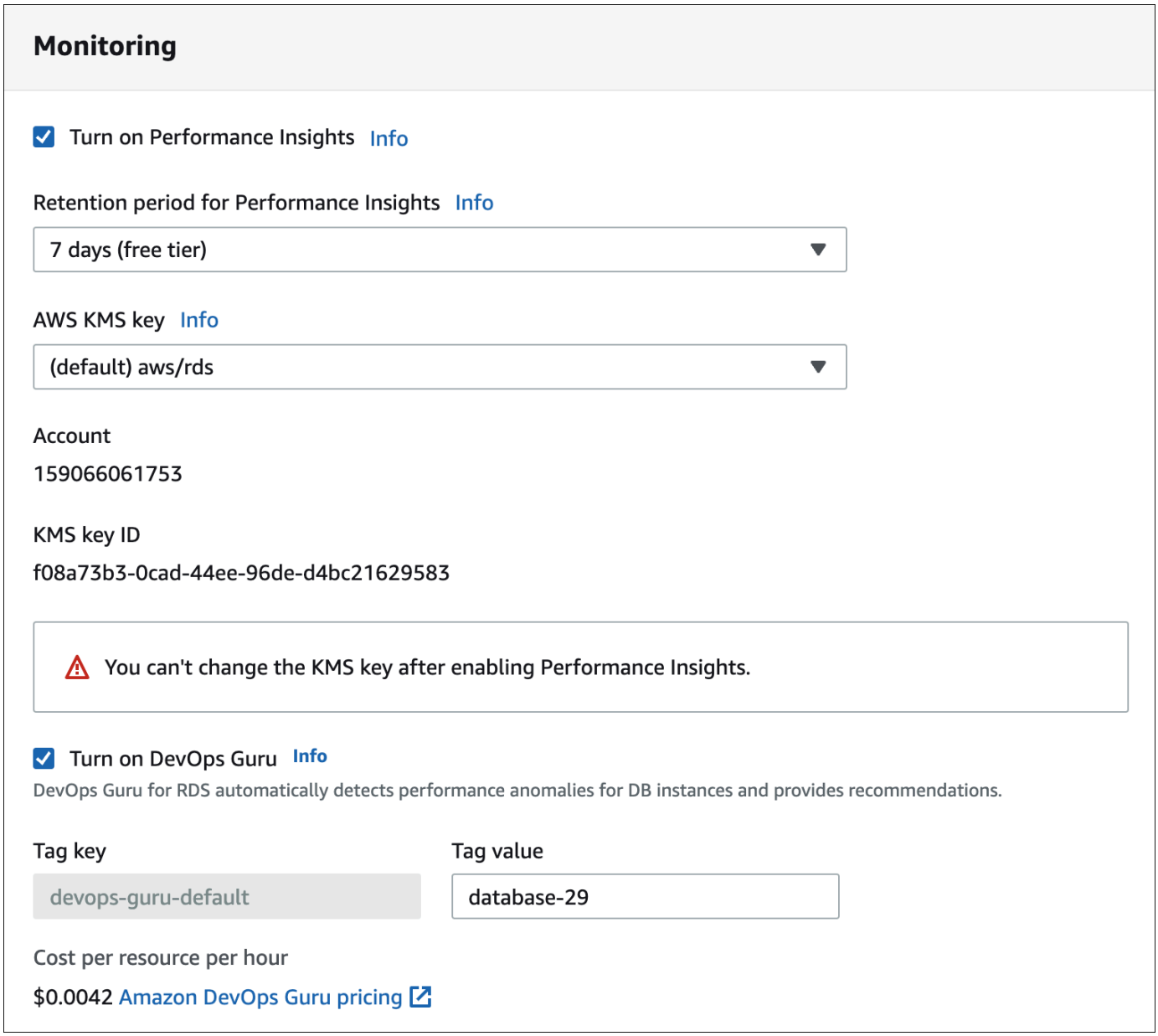
Turning on DevOps Guru when you create an RDS for PostgreSQL database

The creation workflow includes a setting that turns on DevOps Guru coverage for your database. This setting is turned on by default when you choose the **Production** template.

To turn on DevOps Guru when you create an RDS for PostgreSQL database

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Follow the steps in Creating a DB instance, up to but not including the step where you choose monitoring settings.

3. In **Monitoring**, choose **Turn on Performance Insights**. For DevOps Guru for RDS to provide detailed analysis of performance anomalies, Performance Insights must be turned on.
4. Choose **Turn on DevOps Guru**.



5. Create a tag for your database so that DevOps Guru can monitor it. Do the following:
   - In the text field for **Tag key**, enter a name that begins with **`Devops-Guru-`**.
   - In the text field for **Tag value**, enter any value. For example, if you enter **`rds-database-1`** for the name of your RDS for PostgreSQL database, you can also enter **`rds-database-1`** as the tag value.

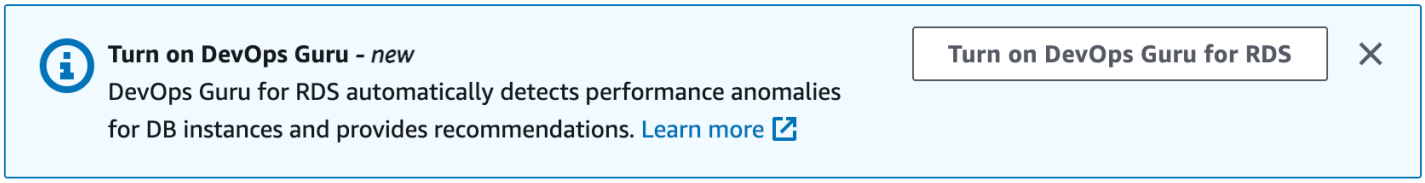
For more information about tags, see "Use tags to identify resources in your DevOps Guru applications" in the *Amazon DevOps Guru User Guide*.

6. Complete the remaining steps in Creating a DB instance.

Turning on DevOps Guru from the notification banner

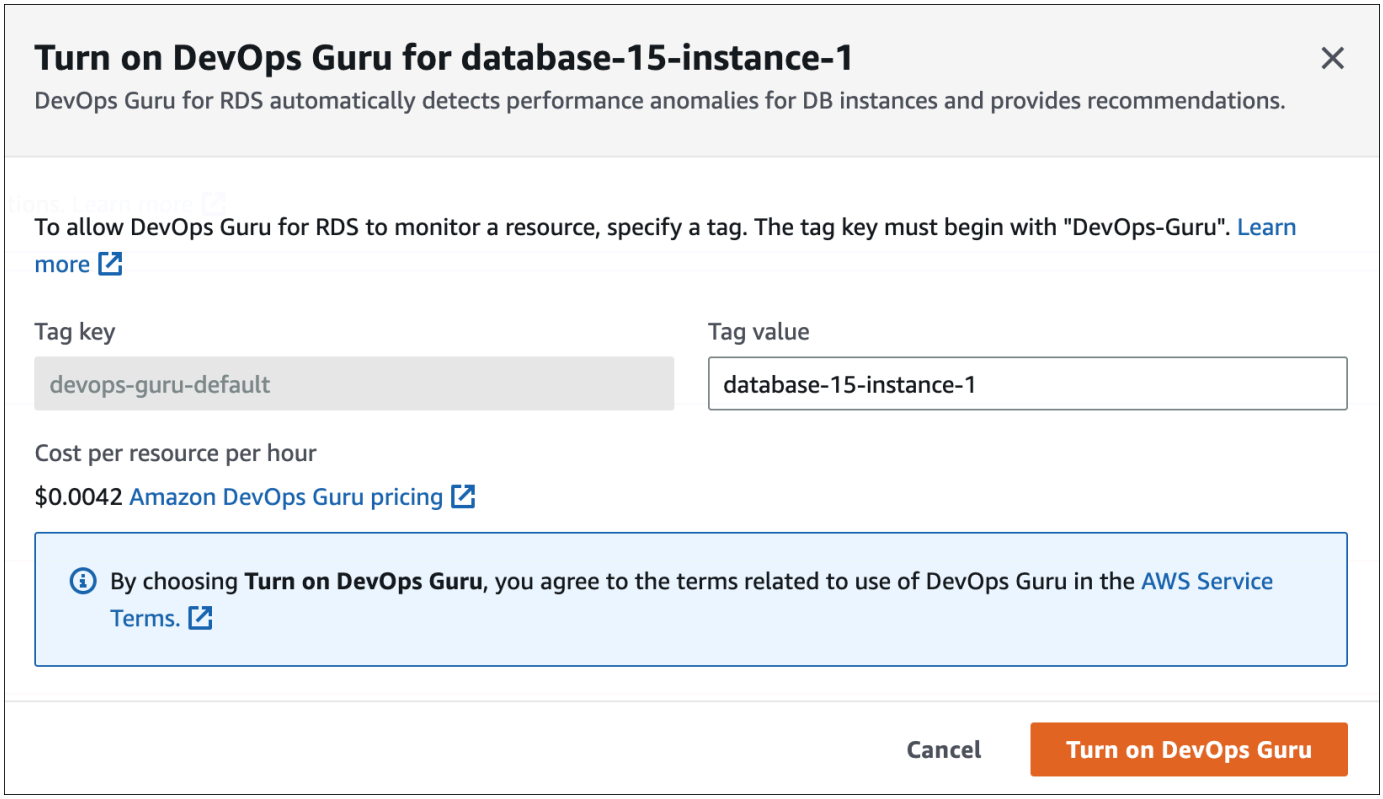
If your resources aren't covered by DevOps Guru, Amazon RDS notifies you with a banner in the following locations:

- The **Monitoring** tab of a DB cluster instance
- The Performance Insights dashboard



To turn on DevOps Guru for your RDS for PostgreSQL database

1. In the banner, choose **Turn on DevOps Guru for RDS**.
2. Enter a tag key name and value. For more information about tags, see "Use tags to identify resources in your DevOps Guru applications" in the *Amazon DevOps Guru User Guide*.



3. Choose **Turn on DevOps Guru**.

Responding to a permissions error when you turn on DevOps Guru

If you turn on DevOps Guru from the RDS console when you create a database, RDS might display the following banner about missing permissions.

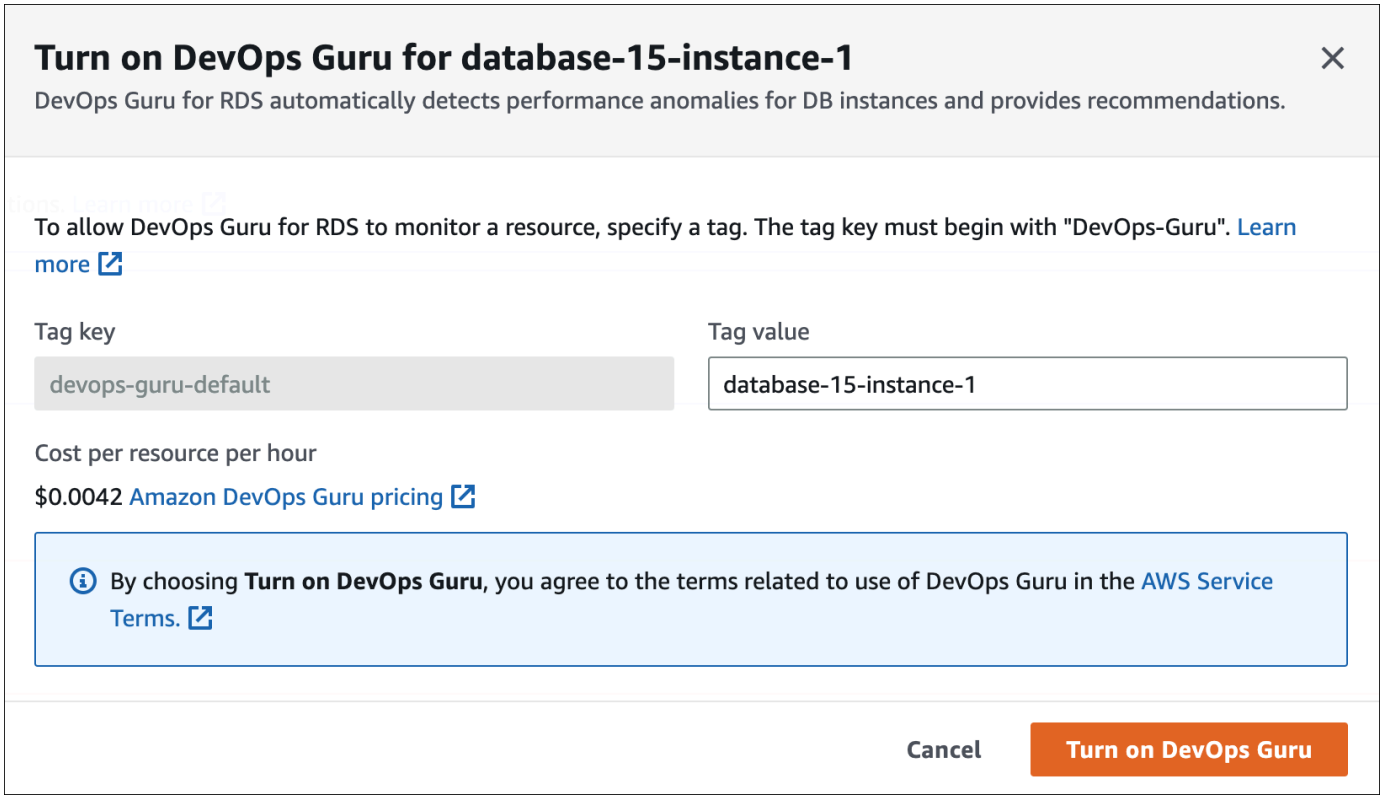


To respond to a permissions error

1. Grant your IAM user or role the user managed role `AmazonDevOpsGuruConsoleFullAccess`. For more information, see Configuring IAM access policies for DevOps Guru for RDS.
2. Open the RDS console.
3. In the navigation pane, choose **Performance Insights**.
4. Choose a DB instance in the cluster that you just created.
5. Choose the switch to turn on **DevOps Guru for RDS**.

DevOps Guru for RDS

6. Choose a tag value. For more information, see "Use tags to identify resources in your DevOps Guru applications" in the *Amazon DevOps Guru User Guide*.



7. Choose **Turn on DevOps Guru**.

Adding RDS for PostgreSQL resources in the DevOps Guru console

You can specify your DevOps Guru resource coverage on the DevOps Guru console. Follow the step described in Specify your DevOps Guru resource coverage in the *Amazon DevOps Guru User Guide*. When you edit your analyzed resources, choose one of the following options:

- Choose **All account resources** to analyze all supported resources, including the RDS for PostgreSQL databases, in your AWS account and Region.
- Choose **CloudFormation stacks** to analyze the RDS for PostgreSQL databases that are in stacks you choose. For more information, see Use AWS CloudFormation stacks to identify resources in your DevOps Guru applications in the *Amazon DevOps Guru User Guide*.
- Choose **Tags** to analyze the RDS for PostgreSQL databases that you have tagged. For more information, see Use tags to identify resources in your DevOps Guru applications in the *Amazon DevOps Guru User Guide*.

For more information, see Enable DevOps Guru in the *Amazon DevOps Guru User Guide*.

Adding RDS for PostgreSQL resources using CloudFormation

You can use tags to add coverage for your RDS for PostgreSQL resources to your CloudFormation templates. The following procedure assumes that you have a CloudFormation template both for your RDS for PostgreSQL DB instance and DevOps Guru stack.

To specify an RDS for PostgreSQL DB instance using a CloudFormation tag

1. In the CloudFormation template for your DB instance, define a tag using a key/value pair.

   The following example assigns the value `my-db-instance1` to `Devops-guru-cfn-default` for an RDS for PostgreSQL DB instance.

   ```
   MyDBInstance1:
     Type: "AWS::RDS::DBInstance"
     Properties:
       DBInstanceIdentifier: my-db-instance1
       Tags:
         - Key: Devops-guru-cfn-default
           Value: devopsguru-my-db-instance1
   ```

2. In the CloudFormation template for your DevOps Guru stack, specify the same tag in your resource collection filter.

   The following example configures DevOps Guru to provide coverage for the resource with the tag value `my-db-instance1`.

   ```
   DevOpsGuruResourceCollection:
     Type: AWS::DevOpsGuru::ResourceCollection
     Properties:
       ResourceCollectionFilter:
         Tags:
           - AppBoundaryKey: "Devops-guru-cfn-default"
             TagValues:
             - "devopsguru-my-db-instance1"
   ```

   The following example provides coverage for all resources within the application boundary `Devops-guru-cfn-default`.

   ```
   DevOpsGuruResourceCollection:
   ```

```
Type: AWS::DevOpsGuru::ResourceCollection
Properties:
  ResourceCollectionFilter:
    Tags:
      - AppBoundaryKey: "Devops-guru-cfn-default"
        TagValues:
        - "*"
```

For more information, see AWS::DevOpsGuru::ResourceCollection and AWS::RDS::DBInstance in the *CloudFormation User Guide*.

Monitoring OS metrics with Enhanced Monitoring

With Enhanced Monitoring, you can monitor the operating system of your DB instance in real time. When you want to see how different processes or threads use the CPU, Enhanced Monitoring metrics are useful.

Topics

- Overview of Enhanced Monitoring
- Setting up and enabling Enhanced Monitoring
- Viewing OS metrics in the RDS console
- Viewing OS metrics using CloudWatch Logs

Overview of Enhanced Monitoring

Amazon RDS provides metrics in real time for the operating system (OS) that your DB instance runs on. You can view all the system metrics and process information for your RDS DB instances on the console. You can manage which metrics you want to monitor for each instance and customize the dashboard according to your requirements. For descriptions of the Enhanced Monitoring metrics, see OS metrics in Enhanced Monitoring.

RDS delivers the metrics from Enhanced Monitoring into your Amazon CloudWatch Logs account. You can create metrics filters in CloudWatch from CloudWatch Logs and display the graphs on the CloudWatch dashboard. You can consume the Enhanced Monitoring JSON output from CloudWatch Logs in a monitoring system of your choice. For more information, see Enhanced Monitoring in the Amazon RDS FAQs.

Topics

- Enhanced Monitoring availability
- Differences between CloudWatch and Enhanced Monitoring metrics
- Retention of Enhanced Monitoring metrics
- Cost of Enhanced Monitoring

Enhanced Monitoring availability

Enhanced Monitoring is available for the following database engines:

- Db2
- MariaDB
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL

Differences between CloudWatch and Enhanced Monitoring metrics

A *hypervisor* creates and runs virtual machines (VMs). Using a hypervisor, an instance can support multiple guest VMs by virtually sharing memory and CPU. CloudWatch gathers metrics about CPU utilization from the hypervisor for a DB instance. In contrast, Enhanced Monitoring gathers its metrics from an agent on the DB instance.

You might find differences between the CloudWatch and Enhanced Monitoring measurements, because the hypervisor layer performs a small amount of work. The differences can be greater if your DB instances use smaller instance classes. In this scenario, more virtual machines (VMs) are probably managed by the hypervisor layer on a single physical instance.

For descriptions of the Enhanced Monitoring metrics, see OS metrics in Enhanced Monitoring. For more information about CloudWatch metrics, see the *Amazon CloudWatch User Guide*.

Retention of Enhanced Monitoring metrics

By default, Enhanced Monitoring metrics are stored for 30 days in the CloudWatch Logs. This retention period is different from typical CloudWatch metrics.

To modify the amount of time the metrics are stored in the CloudWatch Logs, change the retention for the `RDSOSMetrics` log group in the CloudWatch console. For more information, see Change log data retention in CloudWatch logs in the *Amazon CloudWatch Logs User Guide*.

Cost of Enhanced Monitoring

Enhanced Monitoring metrics are stored in the CloudWatch Logs instead of in CloudWatch metrics. The cost of Enhanced Monitoring depends on the following factors:

- You are charged for Enhanced Monitoring only if you exceed the amount of data transfer and storage provided by Amazon CloudWatch Logs. Charges are based on CloudWatch Logs data transfer and storage rates.

- The amount of information transferred for an RDS instance is directly proportional to the defined granularity for the Enhanced Monitoring feature. A smaller monitoring interval results in more frequent reporting of OS metrics and increases your monitoring cost. To manage costs, set different granularities for different instances in your accounts.
- Usage costs for Enhanced Monitoring are applied for each DB instance that Enhanced Monitoring is enabled for. Monitoring a large number of DB instances is more expensive than monitoring only a few.
- DB instances that support a more compute-intensive workload have more OS process activity to report and higher costs for Enhanced Monitoring.

For more information about pricing, see Amazon CloudWatch pricing.

Setting up and enabling Enhanced Monitoring

To use Enhanced Monitoring, you must create an IAM role, and then enable Enhanced Monitoring.

Topics

- Creating an IAM role for Enhanced Monitoring
- Turning Enhanced Monitoring on and off
- Protecting against the confused deputy problem

Creating an IAM role for Enhanced Monitoring

Enhanced Monitoring requires permission to act on your behalf to send OS metric information to CloudWatch Logs. You grant Enhanced Monitoring permissions using an AWS Identity and Access Management (IAM) role. You can either create this role when you enable Enhanced Monitoring or create it beforehand.

Topics

- Creating the IAM role when you enable Enhanced Monitoring
- Creating the IAM role before you enable Enhanced Monitoring

Creating the IAM role when you enable Enhanced Monitoring

When you enable Enhanced Monitoring in the RDS console, Amazon RDS can create the required IAM role for you. The role is named `rds-monitoring-role`. RDS uses this role for the specified DB instance, read replica, or Multi-AZ DB cluster.

To create the IAM role when enabling Enhanced Monitoring

1. Follow the steps in Turning Enhanced Monitoring on and off.
2. Set **Monitoring Role** to **Default** in the step where you choose a role.

Creating the IAM role before you enable Enhanced Monitoring

You can create the required role before you enable Enhanced Monitoring. When you enable Enhanced Monitoring, specify your new role's name. You must create this required role if you enable Enhanced Monitoring using the AWS CLI or the RDS API.

The user that enables Enhanced Monitoring must be granted the `PassRole` permission. For more information, see Example 2 in Granting a user permissions to pass a role to an AWS service in the *IAM User Guide*.

To create an IAM role for Amazon RDS enhanced monitoring

1. Open the IAM console at https://console.aws.amazon.com.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Choose the **AWS service** tab, and then choose **RDS** from the list of services.
5. Choose **RDS - Enhanced Monitoring**, and then choose **Next**.
6. Ensure that the **Permissions policies** shows **AmazonRDSEnhancedMonitoringRole**, and then choose **Next**.
7. For **Role name**, enter a name for your role. For example, enter **`emaccess`**.

   The trusted entity for your role is the AWS service **monitoring.rds.amazonaws.com**.
8. Choose **Create role**.

Turning Enhanced Monitoring on and off

You can manage Enhanced Monitoring using the AWS Management Console, AWS CLI, or RDS API. You can set different granularities for metric collection on each DB instance.

Console

You can turn on Enhanced Monitoring when you create a DB instance, Multi-AZ DB cluster, or read replica, or when you modify a DB instance or Multi-AZ DB cluster. If you modify a DB instance to turn on Enhanced Monitoring, you don't need to reboot your DB instance for the change to take effect.

You can turn on Enhanced Monitoring in the RDS console when you do one of the following actions in the **Databases** page:

- **Create a DB instance or Multi-AZ DB cluster** – Choose **Create database**.
- **Create a read replica** – Choose **Actions**, then **Create read replica**.
- **Modify a DB instance or Multi-AZ DB cluster** – Choose **Modify**.

To turn Enhanced Monitoring on or off in the RDS console

1. Scroll to **Additional configuration**.
2. In **Monitoring**, choose **Enable Enhanced Monitoring** for your DB instance or read replica. Deselect the option to disable Enhanced Monitoring.
3. Set the **Monitoring Role** property to the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose **Default** to have RDS create a role for you named `rds-monitoring-role`.
4. Set the **Granularity** property to the interval, in seconds, between points when metrics are collected for your DB instance or read replica. The **Granularity** property can be set to one of the following values: `1`, `5`, `10`, `15`, `30`, or `60`.

   The fastest that the RDS console refreshes is every 5 seconds. If you set the granularity to 1 second in the RDS console, you still see updated metrics only every 5 seconds. You can retrieve 1-second metric updates by using CloudWatch Logs.

AWS CLI

To turn on Enhanced Monitoring using the AWS CLI, in the following commands, set the `--monitoring-interval` option to a value other than `0` and set the `--monitoring-role-arn` option to the role you created in Creating an IAM role for Enhanced Monitoring.

- create-db-instance
- create-db-instance-read-replica
- modify-db-instance
- create-db-cluster (Multi-AZ DB cluster)
- modify-db-cluster (Multi-AZ DB cluster)

The `--monitoring-interval` option specifies the interval, in seconds, between points when Enhanced Monitoring metrics are collected. Valid values for the option are `0`, `1`, `5`, `10`, `15`, `30`, and `60`.

To turn off Enhanced Monitoring using the AWS CLI, set the `--monitoring-interval` option to `0` in these commands.

Example

The following example turns on Enhanced Monitoring for a DB instance:

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

The following example turns on Enhanced Monitoring for a Multi-AZ DB cluster:

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

RDS API

To turn on Enhanced Monitoring using the RDS API, set the `MonitoringInterval` parameter to a value other than `0` and set the `MonitoringRoleArn` parameter to the role you created in Creating an IAM role for Enhanced Monitoring. Set these parameters in the following actions:

- CreateDBInstance
- CreateDBInstanceReadReplica
- ModifyDBInstance
- CreateDBCluster (Multi-AZ DB cluster)
- ModifyDBCluster (Multi-AZ DB cluster)

The `MonitoringInterval` parameter specifies the interval, in seconds, between points when Enhanced Monitoring metrics are collected. Valid values are `0`, `1`, `5`, `10`, `15`, `30`, and `60`.

To turn off Enhanced Monitoring using the RDS API, set `MonitoringInterval` to `0`.

Protecting against the confused deputy problem

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service

can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account. For more information, see The confused deputy problem.

To limit the permissions to the resource that Amazon RDS can give another service, we recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in a trust policy for your Enhanced Monitoring role. If you use both global condition context keys, they must use the same account ID.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. For Amazon RDS, set `aws:SourceArn` to `arn:aws:rds:`*`Region`*`:`*`my-account-id`*`:db:`*`dbname`*.

The following example uses the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in a trust policy to prevent the confused deputy problem.

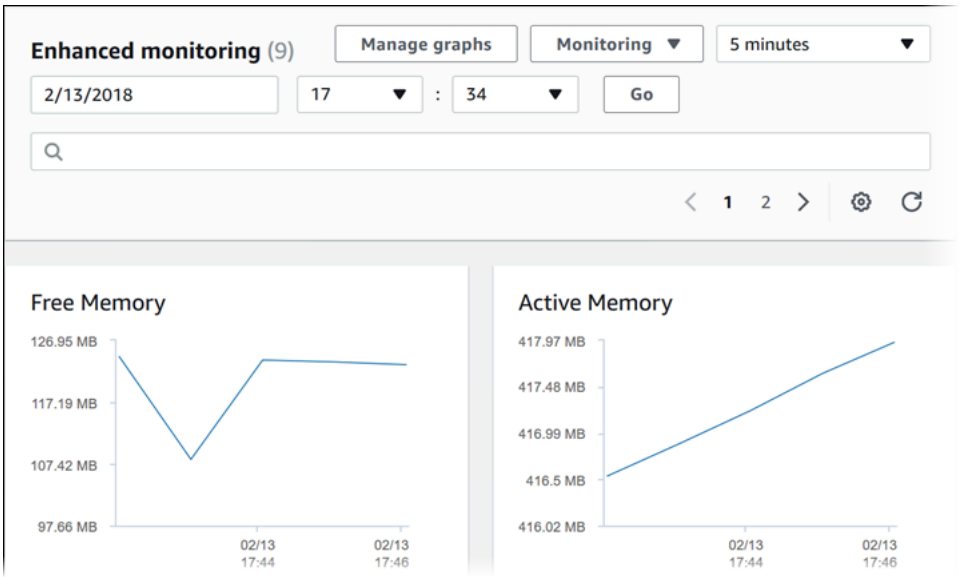
JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "monitoring.rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:rds:Region:my-123456789012:db:dbname"
        },
        "StringEquals": {
          "aws:SourceAccount": "my-123456789012"
        }
      }
    }
  ]
}
```

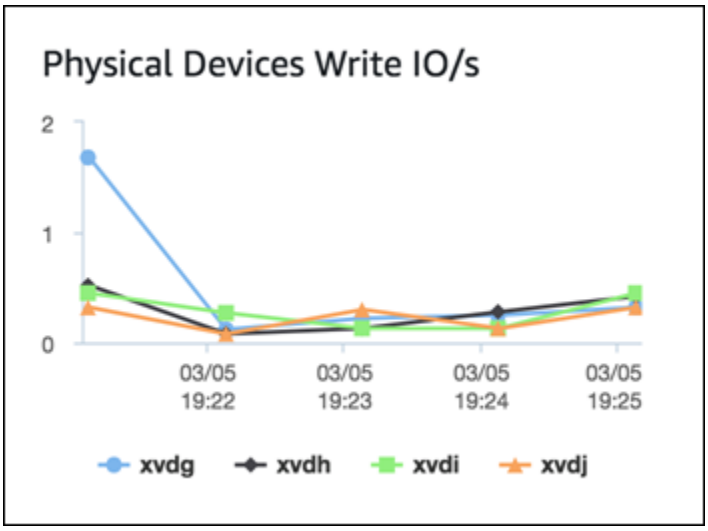
Viewing OS metrics in the RDS console

You can view OS metrics reported by Enhanced Monitoring in the RDS console by choosing **Enhanced monitoring** for **Monitoring**.

The following example shows the Enhanced Monitoring page. For descriptions of the Enhanced Monitoring metrics, see OS metrics in Enhanced Monitoring.



Some DB instances use more than one disk for the DB instance's data storage volume. On those DB instances, the **Physical Devices** graphs show metrics for each one of the disks. For example, the following graph shows metrics for four disks.

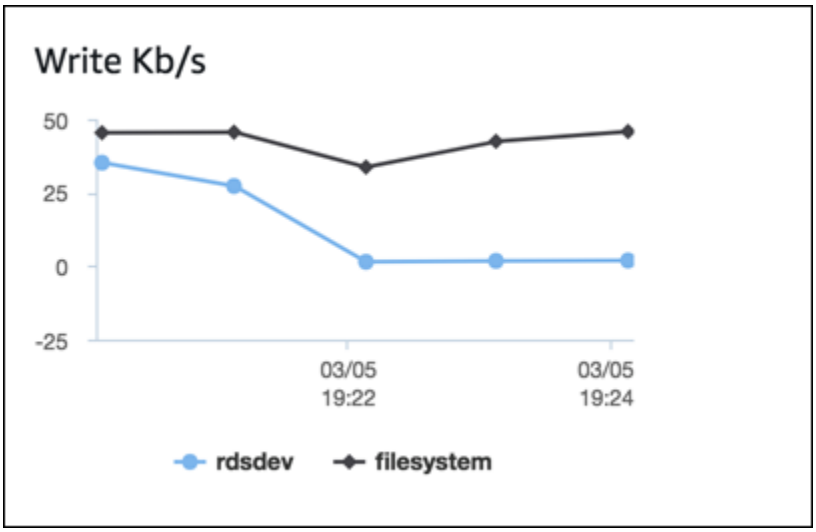


> **Note**
>
> Currently, **Physical Devices** graphs are not available for Microsoft SQL Server DB instances.

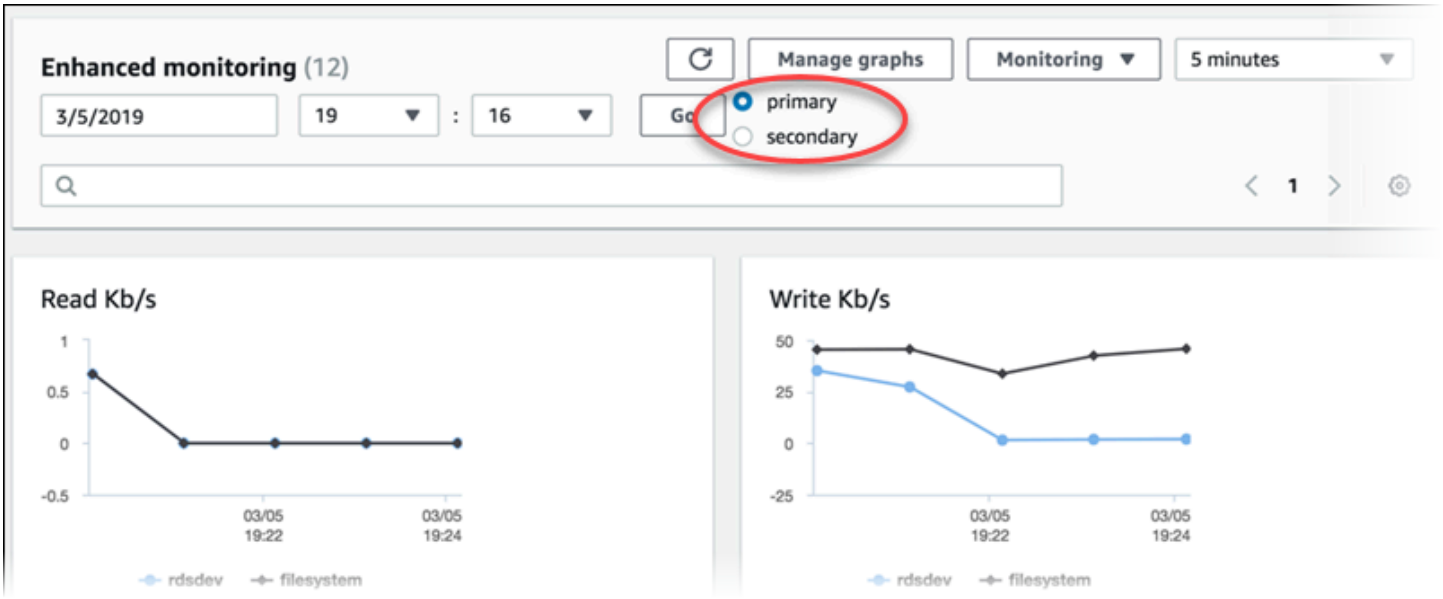
When you are viewing aggregated **File system** graphs, the **rdsdbdata*** device relates to the `rdsfilesys/rdsdbdata*` file system, where all database files and logs are stored. The **rootfilesys** device relates to the `/` file system (also known as root), where files related to the operating system are stored. When using additional storage volumes, view the `rdsdbdata2`, `rdsdbdata3`, and `rdsdbdata4` volume metrics for volume specific information.

When you are viewing aggregated **Disk I/O** graphs, the **rdsdbdata** device relates to the primary `/rdsdbdata` storage volume. When using additional storage volumes, view the `rdsdbdata2`, `rdsdbdata3`, and `rdsdbdata4` volume metrics for volume specific information. The filesystem device relates to the /file system (also known as root), where files related to the operating system are stored.

The **rdsdev** device name is deprecated. The **rdsdev** device only relates to the primary `/rdsdbdata` storage volume and does not include metrics from additional storage volumes.



If the DB instance is a Multi-AZ deployment, you can view the OS metrics for the primary DB instance and its Multi-AZ standby replica. In the **Enhanced monitoring** view, choose **primary** to view the OS metrics for the primary DB instance, or choose **secondary** to view the OS metrics for the standby replica.

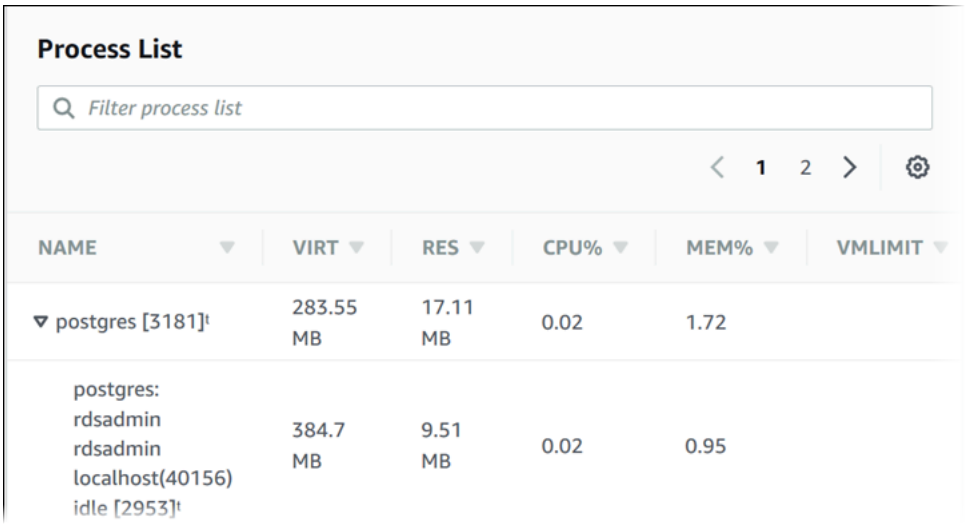


For more information about Multi-AZ deployments, see [Configuring and managing a Multi-AZ deployment for Amazon RDS](#).

> **Note**
>
> Currently, viewing OS metrics for a Multi-AZ standby replica is not supported for MariaDB DB instances.

If you want to see details for the processes running on your DB instance, choose **OS process list** for **Monitoring**.

The **Process List** view is shown following.



The Enhanced Monitoring metrics shown in the **Process list** view are organized as follows:

- **RDS child processes** – Shows a summary of the RDS processes that support the DB instance, for example `mysqld` for MySQL DB instances. Process threads appear nested beneath the parent process. Process threads show CPU utilization only as other metrics are the same for all threads for the process. The console displays a maximum of 100 processes and threads. The results are a combination of the top CPU consuming and memory consuming processes and threads. If there are more than 50 processes and more than 50 threads, the console displays the top 50 consumers in each category. This display helps you identify which processes are having the greatest impact on performance.
- **RDS processes** – Shows a summary of the resources used by the RDS management agent, diagnostics monitoring processes, and other AWS processes that are required to support RDS DB instances.
- **OS processes** – Shows a summary of the kernel and system processes, which generally have minimal impact on performance.

The items listed for each process are:

- **VIRT** – Displays the virtual size of the process.
- **RES** – Displays the actual physical memory being used by the process.
- **CPU%** – Displays the percentage of the total CPU bandwidth being used by the process.
- **MEM%** – Displays the percentage of the total memory being used by the process.

The monitoring data that is shown in the RDS console is retrieved from Amazon CloudWatch Logs. You can also retrieve the metrics for a DB instance as a log stream from CloudWatch Logs. For more information, see Viewing OS metrics using CloudWatch Logs.

Enhanced Monitoring metrics are not returned during the following:

- A failover of the DB instance.
- Changing the instance class of the DB instance (scale compute).

Enhanced Monitoring metrics are returned during a reboot of a DB instance because only the database engine is rebooted. Metrics for the operating system are still reported.

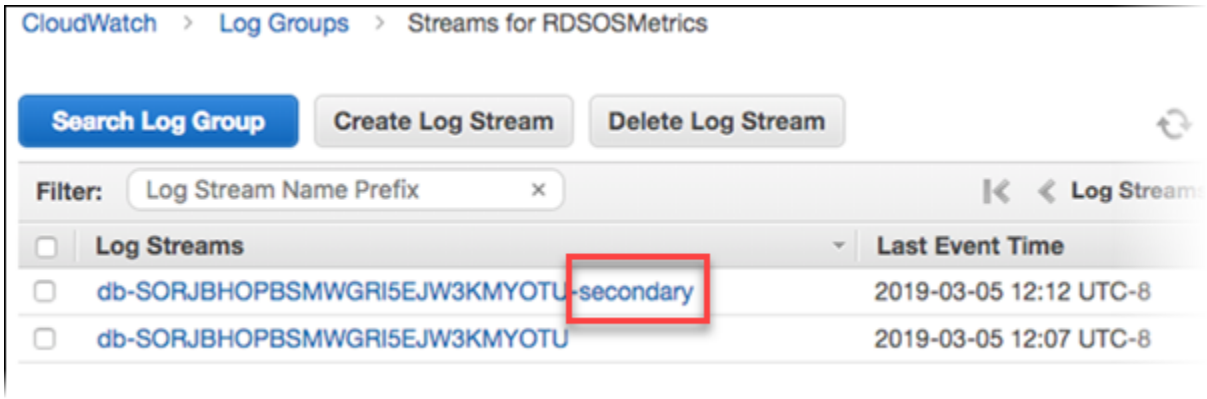
Viewing OS metrics using CloudWatch Logs

After you have enabled Enhanced Monitoring for your DB instance or Multi-AZ DB cluster, you can view the metrics for it using CloudWatch Logs, with each log stream representing a single DB instance or DB cluster being monitored. The log stream identifier is the resource identifier (`DbiResourceId`) for the DB instance or DB cluster.

To view Enhanced Monitoring log data

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, choose the AWS Region that your DB instance or Multi-AZ DB cluster is in. For more information, see Regions and endpoints in the *Amazon Web Services General Reference*.
3. Choose **Logs** in the navigation pane.
4. Choose **RDSOSMetrics** from the list of log groups.

   In a Multi-AZ DB instance deployment, log files with `-secondary` appended to the name are for the Multi-AZ standby replica.



5. Choose the log stream that you want to view from the list of log streams.

Metrics reference for Amazon RDS

In this reference, you can find descriptions of Amazon RDS metrics for Amazon CloudWatch, Performance Insights, and Enhanced Monitoring.

Topics

- Amazon CloudWatch metrics for Amazon RDS
- Amazon CloudWatch dimensions for Amazon RDS
- Amazon CloudWatch metrics for Amazon RDS Performance Insights
- Performance Insights counter metrics
- SQL statistics for Performance Insights
- OS metrics in Enhanced Monitoring

Amazon CloudWatch metrics for Amazon RDS

Amazon CloudWatch metrics provide insights into the performance and health of Amazon RDS instances and clusters, allowing you to monitor system behavior and make data-driven decisions. These metrics help track resource utilization, database activity, and operational efficiency, offering visibility into how your instances are performing.

This reference outlines the specific metrics available for Amazon RDS and explains how to interpret and use them to optimize database performance, troubleshoot issues, and ensure high availability.

Amazon RDS publishes metrics to Amazon CloudWatch in the `AWS/RDS` and `AWS/Usage` namespaces.

Topics

- Amazon CloudWatch instance-level metrics for Amazon RDS
- Amazon CloudWatch usage metrics for Amazon RDS

Amazon CloudWatch instance-level metrics for Amazon RDS

The `AWS/RDS` namespace in Amazon CloudWatch includes the following instance-level metrics.

Note

The Amazon RDS console might display metrics in units that are different from the units sent to Amazon CloudWatch. For example, the Amazon RDS console might display a metric in megabytes (MB), while the metric is sent to Amazon CloudWatch in bytes.

Metric	Description	Applies to	Units
`BinLogDiskUsage`	The amount of disk space occupied by binary logs. If automatic backups are enabled for MySQL and MariaDB instances, including read replicas, binary logs are created.	MariaDB MySQL	Bytes
`BurstBalance`	The percent of General Purpose SSD (gp2) burst-bucket I/O credits available.	All	Percent
`CheckpointLag`	The amount of time since the most recent checkpoint.		Seconds
`ConnectionAttempts`	The number of attempts to connect to an instance, whether successful or not.	MySQL	Count
`CPUUtilization`	The percentage of CPU utilization.	All	Percent
`CPUCreditUsage`	The number of CPU credits spent by the instance for CPU utilization. One CPU credit equals one vCPU running at 100 percent utilization for one minute or an equivalent combination of vCPUs, utilization, and time. For example, you might have one vCPU running at 50 percent utilization for two minutes or two vCPUs running at 25 percent utilization for two minutes.		Credits (vCPU-minutes)

Metric	Description	Applies to	Units
	This metric applies only to `db.t2`, `db.t3`, and `db.t4g` instances. **Note** We recommend using the T DB instance classes only for development and test servers, or other non-production servers. For more details on the T instance classes, see DB instance class types CPU credit metrics are available at a five-minute frequency only. If you specify a period greater than five minutes, use the `Sum` statistic instead of the `Average` statistic.		

Metric	Description	Applies to	Units
`CPUCreditBalance`	The number of earned CPU credits that an instance has accrued since it was launched or started. For T2 Standard, the `CPUCreditBalance` also includes the number of launch credits that have been accrued. Credits are accrued in the credit balance after they are earned, and removed from the credit balance when they are spent. The credit balance has a maximum limit, determined by the instance size. After the limit is reached, any new credits that are earned are discarded. For T2 Standard, launch credits don't count towards the limit. The credits in the `CPUCreditBalance` are available for the instance to spend to burst beyond its baseline CPU utilization. When an instance is running, credits in the `CPUCreditBalance` don't expire. When the instance stops, the `CPUCreditBalance` does not persist, and all accrued credits are lost. CPU credit metrics are available at a five-minute frequency only. This metric applies only to `db.t2`, `db.t3`, and `db.t4g` instances.		Credits (vCPU-minutes)

Metric	Description	Applies to	Units
	Note We recommend using the T DB instance classes only for development and test servers, or other non-production servers. For more details on the T instance classes, see DB instance class types Launch credits work the same way in Amazon RDS as they do in Amazon EC2. For more information, see Launch credits in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.		
`CPUSurplusCreditBalance`	The number of surplus credits that have been spent by an unlimited instance when its `CPUCreditBalance` value is zero. The `CPUSurplusCreditBalance` value is paid down by earned CPU credits. If the number of surplus credits exceeds the maximum number of credits that the instance can earn in a 24-hour period, the spent surplus credits above the maximum incur an additional charge. CPU credit metrics are available at a 5-minute frequency only.	All	Credits (vCPU-minutes)

Metric	Description	Applies to	Units
`CPUSurplusCreditsCharged`	The number of spent surplus credits that are not paid down by earned CPU credits, and which thus incur an additional charge. Spent surplus credits are charged when any of the following occurs: • The spent surplus credits exceed the maximum number of credits that the instance can earn in a 24-hour period. Spent surplus credits above the maximum are charged at the end of the hour. • The instance is stopped or terminated. • The instance is switched from `unlimited` to `standard`. CPU credit metrics are available at a 5-minute frequency only.	All	Credits (vCPU-minutes)

Metric	Description	Applies to	Units
`DatabaseConnections`	The number of client network connections to the database instance. The number of database sessions can be higher than the metric value because the metric value doesn't include the following: • Sessions that no longer have a network connection but which the database hasn't cleaned up • Sessions created by the database engine for its own purposes • Sessions created by the database engine's parallel execution capabilities • Sessions created by the database engine job scheduler • Amazon RDS connections	All	Count
`DiskQueueDepth`	The number of outstanding I/Os (read/write requests) waiting to access the disk.	All	Count
`DiskQueueDepthLogVolume`	The number of outstanding I/Os (read/write requests) waiting to access the log volume disk.	DB instances with dedicated log volume enabled	Count

Metric	Description	Applies to	Units
EBSByteBalance%	The percentage of throughput credits remaining in the burst bucket of your RDS database. This metric is available for basic monitoring only. The metric value is based on the throughput of all volumes, including the root volume, rather than on only those volumes containing database files. To find the instance sizes that support this metric, see the instance sizes with an asterisk (*) in the EBS optimized by default table in *Amazon EC2 User Guide*. The Sum statistic is not applicable to this metric.	All	Percent

Metric	Description	Applies to	Units
`EBSIOBalance%`	The percentage of I/O credits remaining in the burst bucket of your RDS database. This metric is available for basic monitoring only. The metric value is based on the IOPS of all volumes, including the root volume, rather than on only those volumes containing database files. To find the instance sizes that support this metric, see Amazon EBS–optimized instance types in *Amazon EC2 User Guide*. The `Sum` statistic isn't applicable to this metric. This metric is different from `BurstBalance` . To learn how to use this metric, see Improving application performance and reducing costs with Amazon EBS-Optimized Instance burst capability.	All	Percent
`FailedSQLServerAgentJobsCount`	The number of failed Microsoft SQL Server Agent jobs during the last minute.	Microsoft SQL Server	Count per minute
`FreeableMemory`	The amount of available random access memory. For MariaDB, MySQL, Oracle, and PostgreSQL DB instances, this metric reports the value of the `MemAvailable` field of `/proc/meminfo` .	All	Bytes

Metric	Description	Applies to	Units
FreeLocalStorage	The amount of available local storage space. This metric only applies to DB instance classes with NVMe SSD instance store volumes. For information about Amazon EC2 instances with NVMe SSD instance store volumes, see Instance store volumes. The equivalent RDS DB instance classes have the same instance store volumes. For example, the db.m6gd and db.r6gd DB instance classes have NVMe SSD instance store volumes.		Bytes
FreeLocalStoragePercent	The percentage of available local storage space. This metric only applies to DB instance classes with NVMe SSD instance store volumes. For information about Amazon EC2 instances with NVMe SSD instance store volumes, see Instance store volumes. The equivalent RDS DB instance classes have the same instance store volumes. For example, the db.m6gd and db.r6gd DB instance classes have NVMe SSD instance store volumes.		Percent
FreeStorageSpace	The amount of available storage space.	All	Bytes

Metric	Description	Applies to	Units
FreeStorageSpaceLogVolume	The amount of available storage space on the log volume.	DB instances with dedicated log volume enabled	Bytes
IamDbAuthConnectionRequests	The number of connection requests using IAM authentication to the DB instance.	All	Count
MaximumUsedTransactionIDs	The maximum transaction IDs that have been used.	PostgreSQL	Count
NetworkReceiveThroughput	The incoming (receive) network traffic on the DB instance, including both customer database traffic and Amazon RDS traffic used for monitoring and replication.	All	Bytes per second
NetworkTransmitThroughput	The outgoing (transmit) network traffic on the DB instance, including both customer database traffic and Amazon RDS traffic used for monitoring and replication.	All	Bytes per second
OldestLogicalReplicationSlotLag	The lagging size of the Amazon RDS commits a transaction on the source database and the time when RDS applies the transaction on the replica database.	PostgreSQL	Bytes
OldestReplicationSlotLag	The lagging size of the replica lagging the most in terms of write-ahead log (WAL) data received.	PostgreSQL	Bytes

Metric	Description	Applies to	Units
ReadIOPS	The average number of disk read I/O operations per second.	All	Count per second
ReadIOPSLocalStorage	The average number of disk read I/O operations to local storage per second. This metric only applies to DB instance classes with NVMe SSD instance store volumes. For information about Amazon EC2 instances with NVMe SSD instance store volumes, see Instance store volumes. The equivalent RDS DB instance classes have the same instance store volumes. For example, the db.m6gd and db.r6gd DB instance classes have NVMe SSD instance store volumes.		Count per second
ReadIOPSLogVolume	The average number of disk read I/O operations per second for the log volume.	DB instances with dedicated log volume enabled	Count per second
ReadLatency	The average amount of time taken per disk I/O operation.	All	Seconds

Metric	Description	Applies to	Units
`ReadLatencyLocalStorage`	The average amount of time taken per disk I/O operation for local storage. This metric only applies to DB instance classes with NVMe SSD instance store volumes. For information about Amazon EC2 instances with NVMe SSD instance store volumes, see Instance store volumes. The equivalent RDS DB instance classes have the same instance store volumes. For example, the db.m6gd and db.r6gd DB instance classes have NVMe SSD instance store volumes.		Seconds
`ReadLatencyLogVolume`	The average amount of time taken per disk I/O operation for the log volume.	DB instances with dedicated log volume enabled	Seconds
`ReadThroughput`	The average number of bytes read from disk per second.	All	Bytes per second

Metric	Description	Applies to	Units
ReadThroughputLocalStorage	The average number of bytes read from disk per second for local storage. This metric only applies to DB instance classes with NVMe SSD instance store volumes. For information about Amazon EC2 instances with NVMe SSD instance store volumes, see Instance store volumes. The equivalent RDS DB instance classes have the same instance store volumes. For example, the db.m6gd and db.r6gd DB instance classes have NVMe SSD instance store volumes.		Bytes per second
ReadThroughputLogVolume	The average number of bytes read from disk per second for the log volume.	DB instances with dedicated log volume enabled	Bytes per second
ReplicaLag	For read replica configurations, the amount of time a read replica DB instance lags behind the source DB instance. Applies to MariaDB, Microsoft SQL Server, MySQL, Oracle, and PostgreSQL read replicas. For Multi-AZ DB clusters, the difference in time between the latest transaction on the writer DB instance and the latest applied transaction on a reader DB instance.		Seconds

Metric	Description	Applies to	Units
`ReplicationChannelLag`	For multi-source replica configurations, the amount of time a particular channel on the multi-source replica lags behind the source DB instance. For more information, see the section called "Monitoring multi-source replication channels".	MySQL	Seconds
`ReplicationSlotDiskUsage`	The disk space used by replication slot files.	PostgreSQL	Bytes
`SwapUsage`	The amount of swap space used on the DB instance.	MariaDB MySQL Oracle PostgreSQL	Bytes
`TempDbAvailableDataSpace`	The amount of available data space on the tempdb and the volume where tempdb is located. Use this metric to monitor tempdb data space availability and plan capacity accordingly. Low values may indicate the need to increase storage or optimize queries that heavily use tempdb.	SQL Server	Bytes

Metric	Description	Applies to	Units
TempDbAvailableLogSpace	The amount of available log space on the tempdb and the volume where tempdb is located. Use this metric to monitor tempdb log space availability and prevent transaction log full conditions. Critical for workloads with large transactions or high concurrency that generate significant log activity.	SQL Server	Bytes
TempDbDataFileUsage	The percentage of data files used on the tempdb. This metric doesn't account for potential file growth. Use this metric to monitor tempdb data file utilization and identify potential performance bottlenecks. High values may indicate the need to optimize queries that create large temporary objects or increase tempdb size.	SQL Server	Percent
TempDbLogFileUsage	The percentage of log files used on the tempdb. This metric doesn't account for potential file growth. Use this metric to monitor tempdb log file utilization and prevent performance issues. High values may indicate long-running transactions or excessive logging activity that could impact overall database performance.	SQL Server	Percent

Metric	Description	Applies to	Units
TransactionLogsDiskUsage	The disk space used by transaction logs.	PostgreSQL	Bytes
TransactionLogsGeneration	The size of transaction logs generated per second.	PostgreSQL	Bytes per second
WriteIOPS	The average number of disk write I/O operations per second.	All	Count per second
WriteIOPSLocalStorage	The average number of disk write I/O operations per second on local storage. This metric only applies to DB instance classes with NVMe SSD instance store volumes. For information about Amazon EC2 instances with NVMe SSD instance store volumes, see Instance store volumes. The equivalent RDS DB instance classes have the same instance store volumes. For example, the db.m6gd and db.r6gd DB instance classes have NVMe SSD instance store volumes.		Count per second
WriteIOPSLogVolume	The average number of disk write I/O operations per second for the log volume.	DB instances with dedicated log volume enabled	Count per second
WriteLatency	The average amount of time taken per disk I/O operation.	All	Seconds

Metric	Description	Applies to	Units
`WriteLatencyLocalStorage`	The average amount of time taken per disk I/O operation on local storage. This metric only applies to DB instance classes with NVMe SSD instance store volumes. For information about Amazon EC2 instances with NVMe SSD instance store volumes, see Instance store volumes. The equivalent RDS DB instance classes have the same instance store volumes. For example, the db.m6gd and db.r6gd DB instance classes have NVMe SSD instance store volumes.		Seconds
`WriteLatencyLogVolume`	The average amount of time taken per disk I/O operation for the log volume.	DB instances with dedicated log volume enabled	Seconds
`WriteThroughput`	The average number of bytes written to disk per second.	All	Bytes per second
`WriteThroughputLogVolume`	The average number of bytes written to disk per second for the log volume.	DB instances with dedicated log volume enabled	Bytes per second

Metric	Description	Applies to	Units
`WriteThroughputLocalStorage`	The average number of bytes written to disk per second for local storage. This metric only applies to DB instance classes with NVMe SSD instance store volumes. For information about Amazon EC2 instances with NVMe SSD instance store volumes, see Instance store volumes. The equivalent RDS DB instance classes have the same instance store volumes. For example, the db.m6gd and db.r6gd DB instance classes have NVMe SSD instance store volumes.		Bytes per second

Amazon CloudWatch usage metrics for Amazon RDS

The `AWS/Usage` namespace in Amazon CloudWatch includes account-level usage metrics for your Amazon RDS service quotas. CloudWatch collects usage metrics automatically for all AWS Regions.

For more information, see CloudWatch usage metrics in the *Amazon CloudWatch User Guide*. For more information about quotas, see Quotas and constraints for Amazon RDS and Requesting a quota increase in the *Service Quotas User Guide*.

Metric	Description	Units*
`AllocatedStorage`	The total storage for all DB instances. The sum excludes temporary migration instances.	Gigabytes
`AuthorizationsPerDBSecurityGroup`	The number of ingress rules per DB security group in your AWS account. The used value is the highest number of ingress rules in a DB security group in the account. Other DB security groups in the account might have a lower number of ingress rules.	Count

Metric	Description	Units*
CustomEndpointsPerDBCluster	The number of custom endpoints per DB cluster in your AWS account. The used value is the highest number of custom endpoints in a DB cluster in the account. Other DB clusters in the account might have a lower number of custom endpoints.	Count
CustomEngineVersions	The number of custom engine versions (CEVs) for Amazon RDS Custom in your AWS account.	Count
DBClusterParameterGroups	The number of DB cluster parameter groups in your AWS account. The count excludes default parameter groups.	Count
DBClusterRoles	The number of associated AWS Identity and Access Management (IAM) roles per DB cluster in your AWS account. The used value is the highest number of associated IAM roles for a DB cluster in the account. Other DB clusters in the account might have a lower number of associated IAM roles.	Count
DBClusters	The number of Amazon Aurora DB clusters in your AWS account.	Count
DBInstanceRoles	The number of associated AWS Identity and Access Management (IAM) roles per DB instance in your AWS account. The used value is the highest number of associated IAM roles for a DB instance in the account. Other DB instances in the account might have a lower number of associated IAM roles.	Count
DBInstances	The number of DB instances in your AWS account.	Count
DBParameterGroups	The number of DB parameter groups in your AWS account. The count excludes the default DB parameter groups.	Count

Metric	Description	Units*
DBSecurityGroups	The number of security groups in your AWS account. The count excludes the default security group and the default VPC security group.	Count
DBSubnetGroups	The number of DB subnet groups in your AWS account. The count excludes the default subnet group.	Count
EventSubscriptions	The number of event notification subscriptions in your AWS account.	Count
Integrations	The number of zero-ETL integrations with Amazon Redshift in your AWS account.	Count
ManualClusterSnapshots	The number of manually created DB cluster snapshots in your AWS account. The count excludes invalid snapshots.	Count
ManualSnapshots	The number of manually created DB snapshots in your AWS account. The count excludes invalid snapshots.	Count
OptionGroups	The number of option groups in your AWS account. The count excludes the default option groups.	Count
Proxies	The number of RDS proxies in your AWS account.	Count
ReadReplicasPerMaster	The number of read replicas per DB instance in your account. The used value is the highest number of read replicas for a DB instance in the account. Other DB instances in the account might have a lower number of read replicas.	Count
ReservedDBInstances	The number of reserved DB instances in your AWS account. The count excludes retired or declined instances.	Count
SubnetsPerDBSubnetGroup	The number of subnets per DB subnet group in your AWS account. The highest number of subnets for a DB subnet group in the account. Other DB subnet groups in the account might have a lower number of subnets.	Count

Note

Amazon RDS doesn't publish units for usage metrics to CloudWatch. The units only appear in the documentation.

Amazon CloudWatch dimensions for Amazon RDS

You can filter Amazon RDS metrics data by using any dimension in the following table.

Dimension	Filters the requested data for . . .
`DBInstanceIdentifier`	A specific DB instance.
`DatabaseClass`	All instances in a database class. For example, you can aggregate metrics for all instances that belong to the database class `db.r5.large` .
`EngineName`	The identified engine name only. For example, you can aggregate metrics for all instances that have the engine name `postgres`.
`SourceRegion`	The specified Region only. For example, you can aggregate metrics for all DB instances in the `us-east-1` Region.
`DbInstanceIdentifier, VolumeName`	The metrics per-volume for a single instance. RDS captures metrics for multiple storage volumes.

Note

If you are using additional storage volumes, you can see aggregate storage metrics under the `DBInstanceIdentifier` dimension. To see per-volume storage metrics, use the `DbInstanceIdentifier, VolumeName` dimensions.

Amazon CloudWatch metrics for Amazon RDS Performance Insights

Performance Insights automatically publishes some metrics to Amazon CloudWatch. The same data can be queried from Performance Insights, but having the metrics in CloudWatch makes it easy to add CloudWatch alarms. It also makes it easy to add the metrics to existing CloudWatch Dashboards.

Metric	Description
DBLoad	The number of active sessions for the database. Typically, you want the data for the average number of active sessions. In Performance Insights, this data is queried as `db.load.avg` .
DBLoadCPU	The number of active sessions where the wait event type is CPU. In Performance Insights, this data is queried as `db.load.avg` , filtered by the wait event type CPU.
DBLoadNonCPU	The number of active sessions where the wait event type is not CPU.
DBLoadRelativeToNumVCPUs	The ratio of the DB load to the number of virtual CPUs for the database.

Note

These metrics are published to CloudWatch only if there is load on the DB instance.

You can examine these metrics using the CloudWatch console, the AWS CLI, or the CloudWatch API. You can also examine other Performance Insights counter metrics using a special metric math function. For more information, see Querying other Performance Insights counter metrics in CloudWatch.

For example, you can get the statistics for the `DBLoad` metric by running the get-metric-statistics command.

```
aws cloudwatch get-metric-statistics \
    --region us-west-2 \
    --namespace AWS/RDS \
    --metric-name DBLoad  \
    --period 60 \
    --statistics Average \
    --start-time 1532035185 \
    --end-time 1532036185 \
    --dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

This example generates output similar to the following.

```
{
  "Datapoints": [
  {
  "Timestamp": "2021-07-19T21:30:00Z",
  "Unit": "None",
  "Average": 2.1
  },
  {
  "Timestamp": "2021-07-19T21:34:00Z",
  "Unit": "None",
  "Average": 1.7
  },
  {
  "Timestamp": "2021-07-19T21:35:00Z",
  "Unit": "None",
  "Average": 2.8
  },
  {
  "Timestamp": "2021-07-19T21:31:00Z",
  "Unit": "None",
  "Average": 1.5
  },
  {
  "Timestamp": "2021-07-19T21:32:00Z",
  "Unit": "None",
  "Average": 1.8
  },
  {
  "Timestamp": "2021-07-19T21:29:00Z",
  "Unit": "None",
  "Average": 3.0
```

```
},
{
"Timestamp": "2021-07-19T21:33:00Z",
"Unit": "None",
"Average": 2.4
}
],
"Label": "DBLoad"
}
```

For more information about CloudWatch, see What is Amazon CloudWatch? in the *Amazon CloudWatch User Guide*.

Querying other Performance Insights counter metrics in CloudWatch

> **Note**
>
> If you enable the Advanced mode of Database Insights, Amazon RDS publishes Performance Insights counter metrics to Amazon CloudWatch. With Database Insights, you don't need to use the `DB_PERF_INSIGHTS` metric math function. You can use the CloudWatch Database Insights dashboard to search, query, and set alarms for Performance Insights counter metrics.

You can query, alarm, and graphs on RDS Performance Insights metrics from CloudWatch. You can access information about your DB instance by using the `DB_PERF_INSIGHTS` metric math function for CloudWatch. This function allows you to use the Performance Insights metrics that are not directly reported to CloudWatch to create a new time series.

You can use the new Metric Math function by clicking on the **Add Math** drop-down menu in the **Select metric** screen in the CloudWatch console. You can use it to create alarms and graphs on Performance Insights metrics or on combinations of CloudWatch and Performance Insights metrics, including high-resolution alarms for sub-minute metrics. You can also use the function programmatically by including the Metric Math expression in a `get-metric-data` request. For more information, see Metric math syntax and functions and Create an alarm on Performance Insights counter metrics from an AWS database.

Performance Insights counter metrics

Counter metrics are operating system and database performance metrics in the Performance Insights dashboard. To help identify and analyze performance problems, you can correlate counter metrics with DB load. You must append a statistic function to the metric to get the metric values. For example, the supported functions for `os.memory.active` metric are `.avg`, `.min`, `.max`, `.sum`, and `.sample_count`.

The counter metrics are collected one time each minute. The OS metrics collection depends on whether Enhanced Monitoring is turned on or off. If Enhanced Monitoring is turned off, the OS metrics are collected one time each minute. If Enhanced Monitoring is turned on, the OS metrics are collected for the selected time period. For more information about turning Enhanced Monitoring on or off, see Turning Enhanced Monitoring on and off.

Topics

- Performance Insights operating system counters
- Performance Insights counters for Amazon RDS for MariaDB and MySQL
- Performance Insights counters for Amazon RDS for Microsoft SQL Server
- Performance Insights counters for Amazon RDS for Oracle
- Performance Insights counters for Amazon RDS for PostgreSQL

Performance Insights operating system counters

The following operating system counters, which are prefixed with `os`, are available with Performance Insights for all RDS engines except RDS for SQL Server .

You can use `ListAvailableResourceMetrics` API for the list of available counter metrics for your DB instance. For more information, see ListAvailableResourceMetrics in the Amazon RDS Performance Insights API Reference guide.

Counter	Type	Unit	Metric	Description
Active	Memory	Kilobytes	os.memory .active	The amount of assigned memory, in kilobytes.

Counter	Type	Unit	Metric	Description
Buffers	Memory	Kilobytes	os.memory .buffers	The amount of memory used for buffering I/O requests prior to writing to the storage device, in kilobytes.
Cached	Memory	Kilobytes	os.memory .cached	The amount of memory used for caching file system–based I/O, in kilobytes.
DB Cache	Memory	Bytes	os.memory .db.cache	The amount of memory used for page cache by database process in cluding tmpfs (shmem), in bytes.
DB Resident Set Size	Memory	Bytes	os.memory .db.resid entSetSize	The amount of memory used for anonymous and swap cache by database process not including tmpfs (shmem), in bytes.

Counter	Type	Unit	Metric	Description
DB Swap	Memory	Bytes	os.memory .db.swap	The amount of memory used for swap by database process, in bytes.
Dirty	Memory	Kilobytes	os.memory.dirty	The amount of memory pages in RAM that have been modified but not written to their related data block in storage, in kilobytes.
Free	Memory	Kilobytes	os.memory.free	The amount of unassigne d memory, in kilobytes.
Huge Pages Free	Memory	Pages	os.memory .hugePagesFree	The number of free huge pages. Huge pages are a feature of the Linux kernel.
Huge Pages Rsvd	Memory	Pages	os.memory .hugePagesRsvd	The number of committed huge pages.
Huge Pages Size	Memory	Kilobytes	os.memory .hugePagesSize	The size for each huge pages unit, in kilobytes.

Counter	Type	Unit	Metric	Description
Huge Pages Surp	Memory	Pages	os.memory .hugePagesSurp	The number of available surplus huge pages over the total.
Huge Pages Total	Memory	Pages	os.memory .hugePagesTotal	The total number of huge pages.
Inactive	Memory	Kilobytes	os.memory .inactive	The amount of least-fre quently used memory pages, in kilobytes.
Mapped	Memory	Kilobytes	os.memory .mapped	The total amount of file-system contents that is memory mapped inside a process address space, in kilobytes.
Out of Memory Kill Count	Memory	Kills	os.memory .outOfMem oryKillCount	The number of OOM kills that happened over the last collectio n interval.
Page Tables	Memory	Kilobytes	os.memory .pageTables	The amount of memory used by page tables, in kilobytes.

Counter	Type	Unit	Metric	Description
Slab	Memory	Kilobytes	os.memory.slab	The amount of reusable kernel data structures, in kilobytes.
Total	Memory	Kilobytes	os.memory.total	The total amount of memory, in kilobytes.
Writeback	Memory	Kilobytes	os.memory .writeback	The amount of dirty pages in RAM that are still being written to the backing storage, in kilobytes.
Guest	Cpu Utilization	Percentage	os.cpuUti lization.guest	The percentage of CPU in use by guest programs.
Idle	Cpu Utilization	Percentage	os.cpuUti lization.idle	The percentag e of CPU that is idle.
Irq	Cpu Utilization	Percentage	os.cpuUti lization.irq	The percentag e of CPU in use by software interrupts.
Nice	Cpu Utilization	Percentage	os.cpuUti lization.nice	The percentag e of CPU in use by programs running at lowest priority.

Counter	Type	Unit	Metric	Description
Steal	Cpu Utilization	Percentage	os.cpuUtilization.steal	The percentage of CPU in use by other virtual machines.
System	Cpu Utilization	Percentage	os.cpuUtilization.system	The percentage of CPU in use by the kernel.
Total	Cpu Utilization	Percentage	os.cpuUtilization.total	The total percentage of the CPU in use. This value includes the nice value.
User	Cpu Utilization	Percentage	os.cpuUtilization.user	The percentage of CPU in use by user programs.
Wait	Cpu Utilization	Percentage	os.cpuUtilization.wait	The percentage of CPU unused while waiting for I/O access.
Read IOs PS	Disk IO	Requests per second	os.diskIO.<devicename>.readIOsPS	The number of read operations per second.
Write IOs PS	Disk IO	Requests per second	os.diskIO.<devicename>.writeIOsPS	The number of write operations per second.

Counter	Type	Unit	Metric	Description
Avg Queue Len	Disk IO	Requests	os.diskIO.<devicename>.avgQueueLen	The number of requests waiting in the I/O device's queue.
Avg Req Sz	Disk IO	Requests	os.diskIO.<devicename>.avgReqSz	The number of requests waiting in the I/O device's queue.
Await	Disk IO	Milliseconds	os.diskIO.<devicename>.await	The number of milliseconds required to respond to requests, including queue time and service time.
Read IOs PS	Disk IO	Requests	os.diskIO.<devicename>.readIOsPS	The number of read operations per second.
Read KB	Disk IO	Kilobytes	os.diskIO.<devicename>.readKb	The total number of kilobytes read.
Read KB PS	Disk IO	Kilobytes per second	os.diskIO.<devicename>.readKbPS	The number of kilobytes read per second.

Counter	Type	Unit	Metric	Description
Rrqm PS	Disk IO	Requests per second	os.diskIO.<devicename>.rrqmPS	The number of merged read requests queued per second.
TPS	Disk IO	Transactions per second	os.diskIO.<devicename>.tps	The number of I/O transactions per second.
Util	Disk IO	Percentage	os.diskIO.<devicename>.util	The percentage of CPU time during which requests were issued.
Write KB	Disk IO	Kilobytes	os.diskIO.<devicename>.writeKb	The total number of kilobytes written.
Write KB PS	Disk IO	Kilobytes per second	os.diskIO.<devicename>.writeKbPS	The number of kilobytes written per second.
Wrqm PS	Disk IO	Requests per second	os.diskIO.<devicename>.wrqmPS	The number of merged write requests queued per second.
Blocked	Tasks	Tasks	os.tasks.blocked	The number of tasks that are blocked.

Counter	Type	Unit	Metric	Description
Running	Tasks	Tasks	os.tasks.running	The number of tasks that are running.
Sleeping	Tasks	Tasks	os.tasks.sleeping	The number of tasks that are sleeping.
Stopped	Tasks	Tasks	os.tasks.stopped	The number of tasks that are stopped.
Total	Tasks	Tasks	os.tasks.total	The total number of tasks.
Zombie	Tasks	Tasks	os.tasks.zombie	The number of child tasks that are inactive with an active parent task.
One	Load Average Minute	Processes	os.loadAverageMinute.one	The number of processes requesting CPU time over the last minute.
Fifteen	Load Average Minute	Processes	os.loadAverageMinute.fifteen	The number of processes requesting CPU time over the last 15 minutes.

Counter	Type	Unit	Metric	Description
Five	Load Average Minute	Processes	os.loadAverageMinute.five	The number of processes requesting CPU time over the last 5 minutes.
Cached	Swap	Kilobytes	os.swap.cached	The amount of swap memory, in kilobytes, used as cache memory.
Free	Swap	Kilobytes	os.swap.free	The amount of swap memory free, in kilobytes.
In	Swap	Kilobytes	os.swap.in	The amount of memory, in kilobytes, swapped in from disk.
Out	Swap	Kilobytes	os.swap.out	The amount of memory, in kilobytes, swapped out to disk.
Total	Swap	Kilobytes	os.swap.total	The total amount of swap memory available in kilobytes.

Counter	Type	Unit	Metric	Description
Max Files	File Sys	Files	os.fileSys.maxFiles	The maximum number of files that can be created for the file system across all storage volumes.
Used Files	File Sys	Files	os.fileSys.usedFiles	The number of files in the file system across all storage volumes.
Used File Percent	File Sys	Files	os.fileSys.usedFilePercent	The percentage of available files in use across all storage volumes.
Used Percent	File Sys	Percentage	os.fileSys.usedPercent	The percentage of the file-system disk space in use across all storage volumes.
Used	File Sys	Kilobytes	os.fileSys.used	The amount of disk space used by files in the file system across all storage volumes, in kilobytes.

Counter	Type	Unit	Metric	Description
Total	File Sys	Kilobytes	os.fileSys.total	The total disk space available for the file system across all storage volumes, in kilobytes.
Max Files	File Sys	Files	os.fileSys.<volume Name>.maxFiles	The maximum number of files that can be created for the storage volume.
Used Files	File Sys	Files	os.fileSys.<volume Name>.usedFiles	The number of files in the storage volume.
Used File Percent	File Sys	Files	os.fileSys.<volume Name>.usedFilePercent	The percentage of available files in use in the storage volume.
Used Percent	File Sys	Percentage	os.fileSys.<volume Name>.usedPercent	The percentage of the storage volume disk space in use.
Used	File Sys	Kilobytes	os.fileSys.<volume Name>.used	The amount of disk space used by files in the storage volume, in kilobytes.

Counter	Type	Unit	Metric	Description
Total	File Sys	Kilobytes	os.fileSys.<volume Name>.total	The total disk space available in the storage volume, in kilobytes.
Rx	Network	Bytes per second	os.network.rx	The number of bytes received per second.
Tx	Network	Bytes per second	os.network.tx	The number of bytes uploaded per second.
Acu Utilization	General	Percentage	os.general.acuUtilization	The percentage of current capacity out of the maximum configured capacity.
Max Configured Acu	General	ACUs	os.general.maxConfiguredAcu	The maximum capacity configured by the user, in Aurora capacity units (ACUs).
Min Configured Acu	General	ACUs	os.general.minConfiguredAcu	The minimum capacity configured by the user, in ACUs.

Counter	Type	Unit	Metric	Description
Num VCPUs	General	vCPUs	os.general.numVCPUs	The number of virtual CPUs (vCPUs) for the DB instance.
Serverless Database Capacity	General	ACUs	os.general.serverlessDatabaseCapacity	The current capacity of the instance, in ACUs.

Performance Insights counters for Amazon RDS for MariaDB and MySQL

The following database counters are available with Performance Insights for Amazon RDS for MariaDB and MySQL.

Topics

- Native counters for RDS for MariaDB and RDS for MySQL
- Non-native counters for Amazon RDS for MariaDB and MySQL

Native counters for RDS for MariaDB and RDS for MySQL

Native metrics are defined by the database engine and not by Amazon RDS. For definitions of these native metrics, see Server Status Variables (for 8.0) and Server Status Variables (for 8.4) in the MySQL documentation.

Counter	Type	Unit	Metric
Com_analyze	SQL	Queries per second	db.SQL.Com_analyze
Com_optimize	SQL	Queries per second	db.SQL.Com_optimize
Com_select	SQL	Queries per second	db.SQL.Com_select

Counter	Type	Unit	Metric
Connections	SQL	The number of connection attempts per minute (successful or not) to the MySQL server	db.Users.Connections
Innodb_rows_deleted	SQL	Rows per second	db.SQL.Innodb_rows_deleted
Innodb_rows_inserted	SQL	Rows per second	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	Rows per second	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	Rows per second	db.SQL.Innodb_rows_updated
Select_full_join	SQL	Queries per second	db.SQL.Select_full_join
Select_full_range_join	SQL	Queries per second	db.SQL.Select_full_range_join
Select_range	SQL	Queries per second	db.SQL.Select_range
Select_range_check	SQL	Queries per second	db.SQL.Select_range_check
Select_scan	SQL	Queries per second	db.SQL.Select_scan

Counter	Type	Unit	Metric
Slow_queries	SQL	Queries per second	db.SQL.Slow_queries
Sort_merge_passes	SQL	Queries per second	db.SQL.Sort_merge_passes
Sort_range	SQL	Queries per second	db.SQL.Sort_range
Sort_rows	SQL	Queries per second	db.SQL.Sort_rows
Sort_scan	SQL	Queries per second	db.SQL.Sort_scan
Questions	SQL	Queries per second	db.SQL.Questions
Innodb_row_lock_time	Locks	Milliseconds (average)	db.Locks.Innodb_row_lock_time
Table_locks_immediate	Locks	Requests per second	db.Locks.Table_locks_immediate
Table_locks_waited	Locks	Requests per second	db.Locks.Table_locks_waited
Aborted_clients	Users	Connections	db.Users.Aborted_clients
Aborted_connects	Users	Connections	db.Users.Aborted_connects
max_connections	Users	Connections	db.User.max_connections
Threads_created	Users	Connections	db.Users.Threads_created
Threads_running	Users	Connections	db.Users.Threads_running

Counter	Type	Unit	Metric
Innodb_data_writes	I/O	Operations per second	db.IO.Innodb_data_writes
Innodb_dblwr_writes	I/O	Operations per second	db.IO.Innodb_dblwr_writes
Innodb_log_write_requests	I/O	Operations per second	db.IO.Innodb_log_write_requests
Innodb_log_writes	I/O	Operations per second	db.IO.Innodb_log_writes
Innodb_pages_written	I/O	Pages per second	db.IO.Innodb_pages_written
Created_tmp_disk_tables	Temp	Tables per second	db.Temp.Created_tmp_disk_tables
Created_tmp_tables	Temp	Tables per second	db.Temp.Created_tmp_tables
Innodb_buffer_pool_pages_data	Cache	Pages	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_total	Cache	Pages	db.Cache.Innodb_buffer_pool_pages_total
Innodb_buffer_pool_read_requests	Cache	Pages per second	db.Cache.Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads	Cache	Pages per second	db.Cache.Innodb_buffer_pool_reads
Opened_tables	Cache	Tables	db.Cache.Opened_tables
Opened_table_definitions	Cache	Tables	db.Cache.Opened_table_definitions

Counter	Type	Unit	Metric
Qcache_hits	Cache	Queries	db.Cache.Qcache_hits

Non-native counters for Amazon RDS for MariaDB and MySQL

Non-native counter metrics are counters defined by Amazon RDS. A non-native metric can be a metric that you get with a specific query. A non-native metric also can be a derived metric, where two or more native counters are used in calculations for ratios, hit rates, or latencies.

Counter	Type	Unit	Metric	Description	Definition
innodb_buffer_pool_hits	Cache	Reads	db.Cache.innoDB_buffer_pool_hits	The number of reads that InnoDB could satisfy from the buffer pool.	`innodb_buffer_pool_read_requests - innodb_buffer_pool_reads`
innodb_buffer_pool_hit_rate	Cache	Percentage	db.Cache.innoDB_buffer_pool_hit_rate	The percentage of reads that InnoDB could satisfy from the buffer pool.	`100 * innodb_buffer_pool_read_requests / (innodb_buffer_pool_read_requests + innodb_buffer_pool_reads)`

Counter	Type	Unit	Metric	Description	Definition
innodb_buffer_pool_usage	Cache	Percentage	db.Cache.innoDB_buffer_pool_usage	The percentage of the InnoDB buffer pool that contains data (pages). **Note** When using compr d tables, this value can vary. For more inform on, see the info rmatio about Innod ffer_ _page	`Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0`

Counter	Type	Unit	Metric	Description	Definition
				ta and Innodffer_pagetal in Server Status Variab (for 8.0) and Server Status Variab (for 8.4) in the MySQl do cumen on.	
query_cache_hit_rate	Cache	Percentage	db.Cache.query_cache_hit_rate	MySQL result set cache (query cache) hit ratio.	`Qcache_hits / (QCache_hits + Com_select) * 100`

Counter	Type	Unit	Metric	Description	Definition
innodb_datafile_writes_to_disk	I/O	Writes	db.IO.innoDB_datafile_writes_to_disk	The number of InnoDB data file writes to disk, excluding double write and redo logging write operations.	`Innodb_data_writes - Innodb_log_writes - Innodb_dblwr_writes`
innodb_rows_changed	SQL	Rows	db.SQL.innodb_rows_changed	The total InnoDB row operations.	`db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated`
active_transactions	Transactions	Transactions	db.Transactions.active_transactions	The total active transactions.	`SELECT COUNT(1) AS active_transactions FROM INFORMATION_SCHEMA.INNODB_TRX`

Counter	Type	Unit	Metric	Description	Definition
trx_rseg_history_len	Transactions	None	db.Transactions.trx_rseg_history_len	A list of the undo log pages for committed transactions that is maintained by the InnoDB transaction system to implement multi-version concurrency control. For more information about undo log records details, see InnoDB Multi-Versioning (for 8.0) and InnoDB Multi-Versioning	`SELECT COUNT AS trx_rseg_history_len FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='trx_rseg_history_len'`

Counter	Type	Unit	Metric	Description	Definition
				(for 8.4) in the MySQL documentation.	
innodb_deadlocks	Locks	Locks	db.Locks.innodb_deadlocks	The total number of deadlocks.	`SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_deadlocks'`
innodb_lock_timeouts	Locks	Locks	db.Locks.innodb_lock_timeouts	The total number of locks that timed out.	`SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_timeouts'`

Counter	Type	Unit	Metric	Description	Definition
innodb_row_lock_waits	Locks	Locks	db.Locks.innodb_row_lock_waits	The total number of row locks that resulted in a wait.	`SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_row_lock_waits'`

Performance Insights counters for Amazon RDS for Microsoft SQL Server

The following database counters are available with Performance Insights for RDS for Microsoft SQL Server.

Native counters for RDS for Microsoft SQL Server

Native metrics are defined by the database engine and not by Amazon RDS. You can find definitions for these native metrics in Use SQL Server Objects in the Microsoft SQL Server documentation.

Counter	Type	Unit	Metric
Forwarded Records	Access Methods	Records per second	db.Access Methods.Forwarded Records
Page Splits	Access Methods	Splits per second	db.Access Methods.Page Splits
Buffer cache hit ratio	Buffer Manager	Ratio	db.Buffer Manager.Buffer cache hit ratio
Page life expectancy	Buffer Manager	Expectancy in seconds	db.Buffer Manager.Page life expectancy

Counter	Type	Unit	Metric
Page lookups	Buffer Manager	Lookups per second	db.Buffer Manager.Page lookups
Page reads	Buffer Manager	Reads per second	db.Buffer Manager.Page reads
Page writes	Buffer Manager	Writes per second	db.Buffer Manager.Page writes
Active Transactions	Databases	Transactions	db.Databases.Active Transactions (_Total)
Log Bytes Flushed	Databases	Bytes flushed per second	db.Databases.Log Bytes Flushed (_Total)
Log Flush Waits	Databases	Waits per second	db.Databases.Log Flush Waits (_Total)
Log Flushes	Databases	Flushes per second	db.Databases.Log Flushes (_Total)
Write Transactions	Databases	Transactions per second	db.Databases.Write Transactions (_Total)
Processes blocked	General Statistics	Processes blocked	db.General Statistics.Processes blocked
User Connections	General Statistics	Connections	db.General Statistics.User Connections
Latch Waits	Latches	Waits per second	db.Latches.Latch Waits
Number of Deadlocks	Locks	Deadlocks per second	db.Locks.Number of Deadlocks (_Total)

Counter	Type	Unit	Metric
Memory Grants Pending	Memory Manager	Memory grants	db.Memory Manager.Memory Grants Pending
Batch Requests	SQL Statistics	Requests per second	db.SQL Statistic s.Batch Requests
SQL Compilations	SQL Statistics	Compilations per second	db.SQL Statistics.SQL Compilations
SQL Re-Compilations	SQL Statistics	Re-compilations per second	db.SQL Statistics.SQL Re-Compilations

Performance Insights counters for Amazon RDS for Oracle

The following database counters are available with Performance Insights for RDS for Oracle.

Native counters for RDS for Oracle

Native metrics are defined by the database engine and not by Amazon RDS. You can find definitions for these native metrics in Statistics Descriptions in the Oracle documentation.

> **Note**
>
> For the `CPU used by this session` counter metric, the unit has been transformed from the native centiseconds to active sessions to make the value easier to use. For example, CPU send in the DB Load chart represents the demand for CPU. The counter metric `CPU used by this session` represents the amount of CPU used by Oracle sessions. You can compare CPU send to the `CPU used by this session` counter metric. When demand for CPU is higher than CPU used, sessions are waiting for CPU time.

Counter	Type	Unit	Metric
CPU used by this session	User	Active sessions	db.User.CPU used by this session

Counter	Type	Unit	Metric
SQL*Net roundtrips to/from client	User	Roundtrips per second	db.User.SQL*Net roundtrips to/from client
Bytes received via SQL*Net from client	User	Bytes per second	db.User.bytes received via SQL*Net from client
User commits	User	Commits per second	db.User.user commits
Logons cumulative	User	Logons per second	db.User.logons cumulative
User calls	User	Calls per second	db.User.user calls
Bytes sent via SQL*Net to client	User	Bytes per second	db.User.bytes sent via SQL*Net to client
User rollbacks	User	Rollbacks per second	db.User.user rollbacks
Redo size	Redo	Bytes per second	db.Redo.redo size
Parse count (total)	SQL	Parses per second	db.SQL.parse count (total)
Parse count (hard)	SQL	Parses per second	db.SQL.parse count (hard)
Table scan rows gotten	SQL	Rows per second	db.SQL.table scan rows gotten
Sorts (memory)	SQL	Sorts per second	db.SQL.sorts (memory)
Sorts (disk)	SQL	Sorts per second	db.SQL.sorts (disk)
Sorts (rows)	SQL	Sorts per second	db.SQL.sorts (rows)

Counter	Type	Unit	Metric
Physical read bytes	Cache	Bytes per second	db.Cache.physical read bytes
DB block gets	Cache	Blocks per second	db.Cache.db block gets
DBWR checkpoints	Cache	Checkpoints per minute	db.Cache.DBWR checkpoints
Physical reads	Cache	Reads per second	db.Cache.physical reads
Consistent gets from cache	Cache	Gets per second	db.Cache.consistent gets from cache
DB block gets from cache	Cache	Gets per second	db.Cache.db block gets from cache
Consistent gets	Cache	Gets per second	db.Cache.consistent gets

Performance Insights counters for Amazon RDS for PostgreSQL

The following database counters are available with Performance Insights for Amazon RDS for PostgreSQL.

Topics

- Native counters for Amazon RDS for PostgreSQL
- Non-native counters for Amazon RDS for PostgreSQL

Native counters for Amazon RDS for PostgreSQL

Native metrics are defined by the database engine and not by Amazon RDS. You can find definitions for these native metrics in Viewing Statistics in the PostgreSQL documentation.

Counter	Type	Unit	Metric
blks_hit	Cache	Blocks per second	db.Cache.blks_hit
buffers_alloc	Cache	Blocks per second	db.Cache.buffers_alloc
buffers_checkpoint	Checkpoint	Blocks per second	db.Checkpoint.buffers_checkpoint
checkpoint_sync_time	Checkpoint	Milliseconds per checkpoint	db.Checkpoint.checkpoint_sync_time
checkpoint_write_time	Checkpoint	Milliseconds per checkpoint	db.Checkpoint.checkpoint_write_time
checkpoints_req	Checkpoint	Checkpoints per minute	db.Checkpoint.checkpoints_req
checkpoints_timed	Checkpoint	Checkpoints per minute	db.Checkpoint.checkpoints_timed
maxwritten_clean	Checkpoint	Bgwriter clean stops per minute	db.Checkpoint.maxwritten_clean
deadlocks	Concurrency	Deadlocks per minute	db.Concurrency.deadlocks
blk_read_time	I/O	Milliseconds	db.IO.blk_read_time
blks_read	I/O	Blocks per second	db.IO.blks_read
buffers_backend	I/O	Blocks per second	db.IO.buffers_backend
buffers_backend_fsync	I/O	Blocks per second	db.IO.buffers_backend_fsync
buffers_clean	I/O	Blocks per second	db.IO.buffers_clean
tup_deleted	SQL	Tuples per second	db.SQL.tup_deleted
tup_fetched	SQL	Tuples per second	db.SQL.tup_fetched

Counter	Type	Unit	Metric
tup_inserted	SQL	Tuples per second	db.SQL.tup_inserted
tup_returned	SQL	Tuples per second	db.SQL.tup_returned
tup_updated	SQL	Tuples per second	db.SQL.tup_updated
temp_bytes	Temp	Bytes per second	db.Temp.temp_bytes
temp_files	Temp	Files per minute	db.Temp.temp_files
xact_commit	Transactions	Commits per second	db.Transactions.xact_commit
xact_rollback	Transactions	Rollbacks per second	db.Transactions.xact_rollback
numbackends	User	Connections	db.User.numbackends
archived_count	Write-ahead log (WAL)	Files per minute	db.WAL.archived_count

Non-native counters for Amazon RDS for PostgreSQL

Non-native counter metrics are counters defined by Amazon RDS. A non-native metric can be a metric that you get with a specific query. A non-native metric also can be a derived metric, where two or more native counters are used in calculations for ratios, hit rates, or latencies.

Counter	Type	Unit	Metric	Description	Definition
checkpoint_sync_latency	Checkpoint	db.Checkpoint.checkpoint_sync_latency		The total amount of time that has been spent in the portion of checkpoint processing where	`checkpoint_sync_time / (checkpoints_timed + checkpoints_req)`

Counter	Type	Unit	Metric	Description	Definition
				files are synchronized to disk.	
checkpoint_write_latency	Checkpoint	db.Checkpoint.checkpoint_write_latency	The total amount of time that has been spent in the portion of checkpoint processing where files are written to disk.	`checkpoint_write_time / (checkpoints_timed + checkpoints_req)`	
read_latency	I/O	db.IO.rd_late	The time spent reading data file blocks by backends in this instance.	`blk_read_time / blks_read`	
idle_in_transaction_aborted_count	State	Sessio	db.state.idle_in_transaction_aborted_count	The number of sessions in the `idle in transaction (aborted)` state.	Not applicable
idle_in_transaction_count	State	Sessio	db.state.idle_in_transaction_count	The number of sessions in the `idle in transaction` state.	Not applicable
idle_in_transaction_max_time	State	Secon	db.state.idle_in_transaction_max_time	The duration of the longest running transaction in the `idle in transaction` state, in seconds.	Not applicable

Counter	Type	Unit	Metric	Description	Definition
active_transactions	Transactions	Transactions	db.Transactions.active_transactions	The number of active transactions.	Not applicable
blocked_transactions	Transactions	Transactions	db.Transactions.blocked_transactions	The number of blocked transactions.	Not applicable
oldest_active_logical_replication_slot_xid_age	Transactions	db.Transactions.oldest_active_logical_replication_slot_xid_age	The age of the oldest transaction in an active logical replication slot. For more information, see Logical replication slot.	–	
oldest_inactive_logical_replication_slot_xid_age	Transactions	db.Transactions.oldest_inactive_logical_replication_slot_xid_age	The age of the oldest transaction in an inactive logical replication slot. For more information, see Logical replication slot.	–	
oldest_prepared_transaction_xid_age	Transactions	db.Transactions.oldest_prepared_transaction_xid_age	The age of the oldest prepared transaction. For more information, see Prepared transaction.	–	

Counter	Type	Unit	Metric	Description	Definition
oldest_running_transaction_xid_age	Transactions	db.Transactions.oldest_running_transaction_age	The age of the oldest running transaction. For more information, see Active statement for the oldest running active transaction and Idle in transaction for the oldest running idle-in-transaction.	–	
oldest_hot_standby_feedback_xid_age	Transactions	db.Transactions.oldest_hot_standby_feedback_age	The age of the oldest running transaction on a read replica with `hot_standby_feedback` enabled. For more information, see Read replicas.	–	
max_used_xact_ids	Transactions	Transactions	db.Transactions.max_used_xact_ids	The number of transactions that haven't been vacuumed.	Not applicable
max_connections	Users	Connections	db.User.max_connections	The maximum number of connections allowed for a DB instance as configured in `max_connections` parameter.	Not applicable

Counter	Type	Unit	Metric	Description	Definition
archive_failed_count	WAL	Files per minute	db.WAL.archive_failed_count	The number of failed attempts for archiving WAL files, in files per minute.	Not applicable

SQL statistics for Performance Insights

SQL statistics are performance-related metrics about SQL queries that are collected by Performance Insights. Performance Insights gathers statistics for each second that a query is running and for each SQL call. The SQL statistics are an average for the selected time range.

A SQL digest is a composite of all queries having a given pattern but not necessarily having the same literal values. The digest replaces literal values with a question mark. For example, `SELECT * FROM emp WHERE lname= ?`. This digest might consist of the following child queries:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

All engines support SQL statistics for digest queries.

For the region, DB engine, and instance class support information for this feature, see Amazon RDS DB engine, Region, and instance class support for Performance Insights features

Topics

- SQL statistics for MariaDB and MySQL
- SQL statistics for Amazon RDS for Oracle
- SQL statistics for Amazon RDS for SQL Server
- SQL statistics for RDS PostgreSQL

SQL statistics for MariaDB and MySQL

MariaDB and MySQL collect SQL statistics only at the digest level. No statistics are shown at the statement level.

Topics

- Digest statistics for MariaDB and MySQL
- Per-second statistics for MariaDB and MySQL
- Per-call statistics for MariaDB and MySQL
- Primary statistics for MariaDB and MySQL

Digest statistics for MariaDB and MySQL

Performance Insights collects SQL digest statistics from the `events_statements_summary_by_digest` table. The `events_statements_summary_by_digest` table is managed by your database.

The digest table doesn't have an eviction policy. When the table is full, the AWS Management Console shows the following message:

```
Performance Insights is unable to collect SQL Digest statistics on new queries because
 the table events_statements_summary_by_digest is full.
Please truncate events_statements_summary_by_digest table to clear the issue. Check the
 User Guide for more details.
```

In this situation, MariaDB and MySQL don't track SQL queries. To address this issue, Performance Insights automatically truncates the digest table when both of the following conditions are met:

- The table is full.
- Performance Insights manages the Performance Schema automatically.

  For automatic management, the `performance_schema` parameter must be set to `0` and the **Source** must not be set to `user`. If Performance Insights isn't managing the Performance Schema automatically, see Overview of the Performance Schema for Performance Insights on Amazon RDS for MariaDB or MySQL.

In the AWS CLI, check the source of a parameter value by running the describe-db-parameters command.

Per-second statistics for MariaDB and MySQL

The following SQL statistics are available for MariaDB and MySQL DB instances.

Metric	Unit
db.sql_tokenized.stats.count_star_per_sec	Calls per second
db.sql_tokenized.stats.sum_timer_wait_per_sec	Average latency per second (in ms)
db.sql_tokenized.stats.sum_select_full_join_per_sec	Select full join per second
db.sql_tokenized.stats.sum_select_range_check_per_sec	Select range check per second
db.sql_tokenized.stats.sum_select_scan_per_sec	Select scan per second
db.sql_tokenized.stats.sum_sort_merge_passes_per_sec	Sort merge passes per second
db.sql_tokenized.stats.sum_sort_scan_per_sec	Sort scans per second
db.sql_tokenized.stats.sum_sort_range_per_sec	Sort ranges per second
db.sql_tokenized.stats.sum_sort_rows_per_sec	Sort rows per second
db.sql_tokenized.stats.sum_rows_affected_per_sec	Rows affected per second
db.sql_tokenized.stats.sum_rows_examined_per_sec	Rows examined per second
db.sql_tokenized.stats.sum_rows_sent_per_sec	Rows sent per second
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_sec	Created temporary disk tables per second
db.sql_tokenized.stats.sum_created_tmp_tables_per_sec	Created temporary tables per second

Metric	Unit
db.sql_tokenized.stats.sum_lock_time_per_sec	Lock time per second (in ms)

Per-call statistics for MariaDB and MySQL

The following metrics provide per call statistics for a SQL statement.

Metric	Unit
db.sql_tokenized.stats.sum_timer_wait_per_call	Average latency per call (in ms)
db.sql_tokenized.stats.sum_select_full_join_per_call	Select full joins per call
db.sql_tokenized.stats.sum_select_range_check_per_call	Select range check per call
db.sql_tokenized.stats.sum_select_scan_per_call	Select scans per call
db.sql_tokenized.stats.sum_sort_merge_passes_per_call	Sort merge passes per call
db.sql_tokenized.stats.sum_sort_scan_per_call	Sort scans per call
db.sql_tokenized.stats.sum_sort_range_per_call	Sort ranges per call
db.sql_tokenized.stats.sum_sort_rows_per_call	Sort rows per call
db.sql_tokenized.stats.sum_rows_affected_per_call	Rows affected per call
db.sql_tokenized.stats.sum_rows_examined_per_call	Rows examined per call

Metric	Unit
db.sql_tokenized.stats.sum_rows_sent_per_call	Rows sent per call
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_call	Created temporary disk tables per call
db.sql_tokenized.stats.sum_created_tmp_tables_per_call	Created temporary tables per call
db.sql_tokenized.stats.sum_lock_time_per_call	Lock time per call (in ms)

Primary statistics for MariaDB and MySQL

The following SQL statistics are available for MariaDB and MySQL DB instances.

Metric	Unit
db.sql_tokenized.stats.count_star	Calls
db.sql_tokenized.stats.sum_timer_wait	Wait time (in ms)
db.sql_tokenized.stats.sum_select_full_join	Select full join
db.sql_tokenized.stats.sum_select_range_check	Select range checks
db.sql_tokenized.stats.sum_select_scan	Select scans
db.sql_tokenized.stats.sum_sort_merge_passes	Sort merge passes
db.sql_tokenized.stats.sum_sort_scan	Sort scans
db.sql_tokenized.stats.sum_sort_range	Sort ranges
db.sql_tokenized.stats.sum_sort_rows	Sort rows
db.sql_tokenized.stats.sum_rows_affected	Rows affected

Metric	Unit
db.sql_tokenized.stats.sum_rows_examined	Rows examined
db.sql_tokenized.stats.sum_rows_sent	Rows sent
db.sql_tokenized.stats.sum_created_tmp_disk_tables	Created temporary disk tables
db.sql_tokenized.stats.sum_created_tmp_tables	Created temporary tables
db.sql_tokenized.stats.sum_lock_time	Lock time (in ms)

SQL statistics for Amazon RDS for Oracle

Amazon RDS for Oracle collects SQL statistics both at the statement and digest level. At the statement level, the ID column represents the value of `V$SQL.SQL_ID`. At the digest level, the ID column shows the value of `V$SQL.FORCE_MATCHING_SIGNATURE`.

If the ID is `0` at the digest level, Oracle Database has determined that this statement is not suitable for reuse. In this case, the child SQL statements could belong to different digests. However, the statements are grouped together under the `digest_text` for the first SQL statement collected.

Topics

- Per-second statistics for Oracle
- Per-call statistics for Oracle
- Primary statistics for Oracle

Per-second statistics for Oracle

The following metrics provide per-second statistics for an Oracle SQL query.

Metric	Unit
db.sql.stats.executions_per_sec	Number of executions per second
db.sql.stats.elapsed_time_per_sec	Average active executions (AAE)

Metric	Unit
db.sql.stats.rows_processed_per_sec	Rows processed per second
db.sql.stats.buffer_gets_per_sec	Buffer gets per second
db.sql.stats.physical_read_requests_per_sec	Physical reads per second
db.sql.stats.physical_write_requests_per_sec	Physical writes per second
db.sql.stats.total_sharable_mem_per_sec	Total shareable memory per second (in bytes)
db.sql.stats.cpu_time_per_sec	CPU time per second (in ms)

The following metrics provide per-second statistics for an Oracle SQL digest query.

Metric	Unit
db.sql_tokenized.stats.executions_per_sec	Number of executions per second
db.sql_tokenized.stats.elapsed_time_per_sec	Average active executions (AAE)
db.sql_tokenized.stats.rows_processe d_per_sec	Rows processed per second
db.sql_tokenized.stats.buffer_gets_per_sec	Buffer gets per second
db.sql_tokenized.stats.physical_read_requests _per_sec	Physical reads per second
db.sql_tokenized.stats.physical_write_request s_per_sec	Physical writes per second
db.sql_tokenized.stats.total_sharabl e_mem_per_sec	Total shareable memory per second (in bytes)
db.sql_tokenized.stats.cpu_time_per_sec	CPU time per second (in ms)

Per-call statistics for Oracle

The following metrics provide per-call statistics for an Oracle SQL statement.

Metric	Unit
db.sql.stats.elapsed_time_per_exec	Elapsed time per executions (in ms)
db.sql.stats.rows_processed_per_exec	Rows processed per execution
db.sql.stats.buffer_gets_per_exec	Buffer gets per execution
db.sql.stats.physical_read_requests_per_exec	Physical reads per execution
db.sql.stats.physical_write_requests_per_exec	Physical writes per execution
db.sql.stats.total_sharable_mem_per_exec	Total shareable memory per execution (in bytes)
db.sql.stats.cpu_time_per_exec	CPU time per execution (in ms)

The following metrics provide per-call statistics for an Oracle SQL digest query.

Metric	Unit
db.sql_tokenized.stats.elapsed_time_per_exec	Elapsed time per executions (in ms)
db.sql_tokenized.stats.rows_processed_per_exec	Rows processed per execution
db.sql_tokenized.stats.buffer_gets_per_exec	Buffer gets per execution
db.sql_tokenized.stats.physical_read_requests_per_exec	Physical reads per execution
db.sql_tokenized.stats.physical_write_requests_per_exec	Physical writes per execution
db.sql_tokenized.stats.total_sharable_mem_per_exec	Total shareable memory per execution (in bytes)

Metric	Unit
db.sql_tokenized.stats.cpu_time_per_exec	CPU time per execution (in ms)

Primary statistics for Oracle

The following metrics provide primary statistics for an Oracle SQL query.

Metric	Unit
db.sql.stats.executions	Number of executions
db.sql.stats.elapsed_time	Elapsed time (in ms)
db.sql.stats.rows_processed	Rows processed
db.sql.stats.buffer_gets	Buffer gets
db.sql.stats.physical_read_requests	Physical reads
db.sql.stats.physical_write_requests	Physical writes
db.sql.stats.total_sharable_mem	Total shareable memory (in bytes)
db.sql.stats.cpu_time	CPU time (in ms)

The following metrics provide primary statistics for an Oracle SQL digest query.

Metric	Unit
db.sql_tokenized.stats.executions	Number of executions
db.sql_tokenized.stats.elapsed_time	Elapsed time (in ms)
db.sql_tokenized.stats.rows_processed	Rows processed
db.sql_tokenized.stats.buffer_gets	Buffer gets
db.sql_tokenized.stats.physical_read_requests	Physical reads

Metric	Unit
db.sql_tokenized.stats.physical_write_requests	Physical writes
db.sql_tokenized.stats.total_sharable_mem	Total shareable memory (in bytes)
db.sql_tokenized.stats.cpu_time	CPU time (in ms)

SQL statistics for Amazon RDS for SQL Server

Amazon RDS for SQL Server collects SQL statistics both at the statement and digest level. At the statement level, the ID column represents the value of `sql_handle`. At the digest level, the ID column shows the value of `query_hash`.

SQL Server returns NULL values for `query_hash` for a few statements. For example, ALTER INDEX, CHECKPOINT, UPDATE STATISTICS, COMMIT TRANSACTION, FETCH NEXT FROM Cursor, and a few INSERT statements, SELECT @<variable>, conditional statements, and executable stored procedures. In this case, the `sql_handle` value is displayed as the ID at the digest level for that statement.

Topics

- Per-second statistics for SQL Server
- Per-call statistics for SQL Server
- Primary statistics for SQL Server

Per-second statistics for SQL Server

The following metrics provide per-second statistics for a SQL Server SQL query.

Metric	Unit
db.sql.stats.execution_count_per_sec	Number of executions per second
db.sql.stats.total_elapsed_time_per_sec	Total elapsed time per second
db.sql.stats.total_rows_per_sec	Total rows processed per second
db.sql.stats.total_logical_reads_per_sec	Total logical reads per second

Metric	Unit
db.sql.stats.total_logical_writes_per_sec	Total logical writes per second
db.sql.stats.total_physical_reads_per_sec	Total physical reads per second
db.sql.stats.total_worker_time_per_sec	Total CPU time (in ms)

The following metrics provide per-second statistics for a SQL Server SQL digest query.

Metric	Unit
db.sql_tokenized.stats.execution_count_per_sec	Number of execution per second
db.sql_tokenized.stats.total_elapsed_time_per_sec	Total elapsed time per second
db.sql_tokenized.stats.total_rows_per_sec	Total rows processed per second
db.sql_tokenized.stats.total_logical_reads_per_sec	Total logical reads per second
db.sql_tokenized.stats.total_logical_writes_per_sec	Total logical writes per second
db.sql_tokenized.stats.total_physical_reads_per_sec	Total physical reads per second
db.sql_tokenized.stats.total_worker_time_per_sec	Total CPU time (in ms)

Per-call statistics for SQL Server

The following metrics provide per-call statistics for a SQL Server SQL statement.

Metric	Unit
db.sql.stats.total_elapsed_time_per_call	Total elapsed time per execution (in ms)
db.sql.stats.total_rows_per_call	Total rows processed per execution
db.sql.stats.total_logical_reads_per_call	Total logical reads per execution
db.sql.stats.total_logical_writes_per_call	Total logical writes per execution
db.sql.stats.total_physical_reads_per_call	Total physical reads per execution
db.sql.stats.total_worker_time_per_call	Total CPU time per execution (in ms)

The following metrics provide per-call statistics for a SQL Server SQL digest query.

Metric	Unit
db.sql_tokenized.stats.total_elapsed_time_per_call	Total elapsed time per execution
db.sql_tokenized.stats.total_rows_per_call	Total rows processed per execution
db.sql_tokenized.stats.total_logical_reads_per_call	Total logical reads per execution
db.sql_tokenized.stats.total_logical_writes_per_call	Total logical writes per execution
db.sql_tokenized.stats.total_physical_reads_per_call	Total physical reads per execution
db.sql_tokenized.stats.total_worker_time_per_call	Total CPU time per execution (in ms)

Primary statistics for SQL Server

The following metrics provide primary statistics for a SQL Server SQL query.

Metric	Unit
db.sql.stats.execution_count	Number of executions
db.sql.stats.total_elapsed_time	Total elapsed time (in ms)
db.sql.stats.total_rows	Total rows processed
db.sql.stats.total_logical_reads	Total logical reads
db.sql.stats.total_logical_writes	Total logical writes
db.sql.stats.total_physical_reads	Total physical reads
db.sql.stats.total_worker_time	Total CPU time (in ms)

The following metrics provide primary statistics for a SQL Server SQL digest query.

Metric	Unit
db.sql_tokenized.stats.execution_count	Number of execution
db.sql_tokenized.stats.total_elapsed_time	Total elapsed time (in ms)
db.sql_tokenized.stats.total_rows	Total rows processed
db.sql_tokenized.stats.total_logical_reads	Total logical reads
db.sql_tokenized.stats.total_logical_writes	Total logical writes
db.sql_tokenized.stats.total_physical_reads	Total physical reads
db.sql_tokenized.stats.total_worker_time	Total CPU time (in ms)

SQL statistics for RDS PostgreSQL

For each SQL call and for each second that a query runs, Performance Insights collects SQL statistics. RDS for PostgreSQL collect SQL statistics only at the digest–level. No statistics are shown at the statement-level.

Following, you can find information about digest-level statistics for RDS for PostgreSQL.

Topics

- Digest statistics for RDS PostgreSQL
- Per-second digest statistics for RDS PostgreSQL
- Per-call digest statistics for RDS PostgreSQL
- Primary statistics for RDS PostgreSQL

Digest statistics for RDS PostgreSQL

To view SQL digest statistics, RDS PostgreSQL must load the `pg_stat_statements` library. For PostgreSQL DB instances that are compatible with PostgreSQL 11 or later, the database loads this library by default. For PostgreSQL DB instances that are compatible with PostgreSQL 10 or earlier, enable this library manually. To enable it manually, add `pg_stat_statements` to `shared_preload_libraries` in the DB parameter group associated with the DB instance. Then reboot your DB instance. For more information, see Parameter groups for Amazon RDS.

> **Note**
>
> Performance Insights can only collect statistics for queries in `pg_stat_activity` that aren't truncated. By default, PostgreSQL databases truncate queries longer than 1,024 bytes. To increase the query size, change the `track_activity_query_size` parameter in the DB parameter group associated with your DB instance. When you change this parameter, a DB instance reboot is required.

Per-second digest statistics for RDS PostgreSQL

The following SQL digest statistics are available for PostgreSQL DB instances.

Metric	Unit
db.sql_tokenized.stats.calls_per_sec	Calls per second
db.sql_tokenized.stats.rows_per_sec	Rows per second
db.sql_tokenized.stats.total_time_per_sec	Average active executions per second (AAE)

Metric	Unit
db.sql_tokenized.stats.shared_blks_hit_per_sec	Block hits per second
db.sql_tokenized.stats.shared_blks_read_per_sec	Block reads per second
db.sql_tokenized.stats.shared_blks_dirtied_per_sec	Blocks dirtied per second
db.sql_tokenized.stats.shared_blks_written_per_sec	Block writes per second
db.sql_tokenized.stats.local_blks_hit_per_sec	Local block hits per second
db.sql_tokenized.stats.local_blks_read_per_sec	Local block reads per second
db.sql_tokenized.stats.local_blks_dirtied_per_sec	Local block dirtied per second
db.sql_tokenized.stats.local_blks_written_per_sec	Local block writes per second
db.sql_tokenized.stats.temp_blks_written_per_sec	Temporary writes per second
db.sql_tokenized.stats.temp_blks_read_per_sec	Temporary reads per second
db.sql_tokenized.stats.blk_read_time_per_sec	Average concurrent reads per second
db.sql_tokenized.stats.blk_write_time_per_sec	Average concurrent writes per second

Per-call digest statistics for RDS PostgreSQL

The following metrics provide per call statistics for a SQL statement.

Metric	Unit
db.sql_tokenized.stats.rows_per_call	Rows per call
db.sql_tokenized.stats.avg_latency_per_call	Average latency per call (in ms)
db.sql_tokenized.stats.shared_blks_hit_per_call	Block hits per call
db.sql_tokenized.stats.shared_blks_read_per_call	Block reads per call
db.sql_tokenized.stats.shared_blks_written_per_call	Block writes per call
db.sql_tokenized.stats.shared_blks_dirtied_per_call	Blocks dirtied per call
db.sql_tokenized.stats.local_blks_hit_per_call	Local block hits per call
db.sql_tokenized.stats.local_blks_read_per_call	Local block reads per call
db.sql_tokenized.stats.local_blks_dirtied_per_call	Local block dirtied per call
db.sql_tokenized.stats.local_blks_written_per_call	Local block writes per call
db.sql_tokenized.stats.temp_blks_written_per_call	Temporary block writes per call
db.sql_tokenized.stats.temp_blks_read_per_call	Temporary block reads per call
db.sql_tokenized.stats.blk_read_time_per_call	Read time per call (in ms)
db.sql_tokenized.stats.blk_write_time_per_call	Write time per call (in ms)

Primary statistics for RDS PostgreSQL

The following SQL statistics are available for PostgreSQL DB instances.

Metric	Unit
db.sql_tokenized.stats.calls	Calls
db.sql_tokenized.stats.rows	Rows
db.sql_tokenized.stats.total_time	Total time (in ms)
db.sql_tokenized.stats.shared_blks_hit	Block hits
db.sql_tokenized.stats.shared_blks_read	Block reads
db.sql_tokenized.stats.shared_blks_dirtied	Blocks dirtied
db.sql_tokenized.stats.shared_blks_written	Block writes
db.sql_tokenized.stats.local_blks_hit	Local block hits
db.sql_tokenized.stats.local_blks_read	Local block reads
db.sql_tokenized.stats.local_blks_dirtied	Local blocks dirtied
db.sql_tokenized.stats.local_blks_written	Local block writes
db.sql_tokenized.stats.temp_blks_written	Temporary writes
db.sql_tokenized.stats.temp_blks_read	Temporary reads
db.sql_tokenized.stats.blk_read_time	Average concurrent reads (in ms)
db.sql_tokenized.stats.blk_write_time	Average concurrent writes (in ms)

For more information about these metrics, see pg_stat_statements in the PostgreSQL documentation.

OS metrics in Enhanced Monitoring

Amazon RDS provides metrics in real time for the operating system (OS) that your DB instance runs on. RDS delivers the metrics from Enhanced Monitoring to your Amazon CloudWatch Logs account. The following tables list the OS metrics available using Amazon CloudWatch Logs.

Topics

- OS metrics for Db2, MariaDB, MySQL, Oracle, and PostgreSQL
- OS metrics for Microsoft SQL Server

OS metrics for Db2, MariaDB, MySQL, Oracle, and PostgreSQL

RDS collects disk metrics for `rdsdbdata` as well as additional storage volumes present in the OS. To view OS metrics in CloudWatch Logs, see Viewing OS metrics using CloudWatch Logs.

Group	Metric	Console name	Description
`General`	`engine`	Not applicable	The database engine for the DB instance.
	`instanceID`	Not applicable	The DB instance identifier.
	`instanceResourceID`	Not applicable	An immutable identifier for the DB instance that is unique to an AWS Region, also used as the log stream identifier.
	`numVCPUs`	Not applicable	The number of virtual CPUs for the DB instance.
	`timestamp`	Not applicable	The time at which the metrics were taken.
	`uptime`	Not applicable	The amount of time that the DB instance has been active.

Group	Metric	Console name	Description
	version	Not applicable	The version of the OS metrics' stream JSON format.
cpuUtilization	guest	**CPU Guest**	The percentage of CPU in use by guest programs.
	idle	**CPU Idle**	The percentage of CPU that is idle.
	irq	**CPU IRQ**	The percentage of CPU in use by software interrupts.
	nice	**CPU Nice**	The percentage of CPU in use by programs running at lowest priority.
	steal	**CPU Steal**	The percentage of CPU in use by other virtual machines.
	system	**CPU System**	The percentage of CPU in use by the kernel.
	total	**CPU Total**	The total percentage of the CPU in use. This value includes the nice value.
	user	**CPU User**	The percentage of CPU in use by user programs.
	wait	**CPU Wait**	The percentage of CPU unused while waiting for I/O access.
diskIO	avgQueueLen	**Avg Queue Size**	The number of requests waiting in the I/O device's queue.
	avgReqSz	**Ave Request Size**	The average request size, in kilobytes.
	await	**Disk I/O Await**	The number of milliseconds required to respond to requests, including queue time and service time.

Group	Metric	Console name	Description
	device	Not applicable	The identifier of the disk device in use.
	readIOsPS	**Read IO/s**	The number of read operations per second.
	readKb	**Read Total**	The total number of kilobytes read.
	readKbPS	**Read Kb/s**	The number of kilobytes read per second.
	readLatency	**Read Latency**	The elapsed time between the submission of a read I/O request and its completion, in milliseconds. This metric is only available for Amazon Aurora.
	readThroughput	**Read Throughput**	The amount of network throughput used by requests to the DB cluster, in bytes per second. This metric is only available for Amazon Aurora.
	rrqmPS	**Rrqms**	The number of merged read requests queued per second.
	tps	**TPS**	The number of I/O transactions per second.
	util	**Disk I/O Util**	The percentage of CPU time during which requests were issued.
	writeIOsPS	**Write IO/s**	The number of write operations per second.
	writeKb	**Write Total**	The total number of kilobytes written.
	writeKbPS	**Write Kb/s**	The number of kilobytes written per second.

Group	Metric	Console name	Description
	writeLatency	**Write Latency**	The average elapsed time between the submission of a write I/O request and its completion, in milliseconds. This metric is only available for Amazon Aurora.
	writeThroughput	**Write Throughput**	The amount of network throughput used by responses from the DB cluster, in bytes per second. This metric is only available for Amazon Aurora.
	wrqmPS	**Wrqms**	The number of merged write requests queued per second.
physicalDeviceIO	avgQueueLen	**Physical Devices Avg Queue Size**	The number of requests waiting in the I/O device's queue.
	avgReqSz	**Physical Devices Ave Request Size**	The average request size, in kilobytes.
	await	**Physical Devices Disk I/O Await**	The number of milliseconds required to respond to requests, including queue time and service time.
	device	Not applicable	The identifier of the disk device in use.
	mountPoint	Not applicable	The path to the file system.

Group	Metric	Console name	Description
	readIOsPS	**Physical Devices Read IO/s**	The number of read operations per second.
	readKb	**Physical Devices Read Total**	The total number of kilobytes read.
	readKbPS	**Physical Devices Read Kb/s**	The number of kilobytes read per second.
	rrqmPS	**Physical Devices Rrqms**	The number of merged read requests queued per second.
	tps	**Physical Devices TPS**	The number of I/O transactions per second.
	util	**Physical Devices Disk I/O Util**	The percentage of CPU time during which requests were issued.
	writeIOsPS	**Physical Devices Write IO/s**	The number of write operations per second.
	writeKb	**Physical Devices Write Total**	The total number of kilobytes written.

Group	Metric	Console name	Description
	writeKbPS	**Physical Devices Write Kb/s**	The number of kilobytes written per second.
	wrqmPS	**Physical Devices Wrqms**	The number of merged write requests queued per second.
fileSys	maxFiles	**Max Inodes**	The maximum number of files that can be created for the file system.
	mountPoint	Not applicable	The path to the file system. When this is /rdsdbdata* , it represents the aggregate of all storage volumes.
	name	Not applicable	The name of the file system.
	total	**Total Filesystem**	The total number of disk space available for the file system, in kilobytes.
	used	**Used Filesystem**	The amount of disk space used by files in the file system, in kilobytes.
	usedFilePercent	**Used Inodes**	The percentage of available files in use.
	usedFiles	**Used%**	The number of files in the file system.
	usedPercent	**Used Filesystem**	The percentage of the file-system disk space in use.
loadAverageMinute	fifteen	**Load Avg 15 min**	The number of processes requesting CPU time over the last 15 minutes.

Group	Metric	Console name	Description
	five	**Load Avg 5 min**	The number of processes requesting CPU time over the last 5 minutes.
	one	**Load Avg 1 min**	The number of processes requesting CPU time over the last minute.
memory	active	**Active Memory**	The amount of assigned memory, in kilobytes.
	buffers	**Buffered Memory**	The amount of memory used for buffering I/O requests prior to writing to the storage device, in kilobytes.
	cached	**Cached Memory**	The amount of memory used for caching file system–based I/O.
	dirty	**Dirty Memory**	The amount of memory pages in RAM that have been modified but not written to their related data block in storage, in kilobytes.
	free	**Free Memory**	The amount of unassigned memory, in kilobytes.
	hugePagesFree	**Huge Pages Free**	The number of free huge pages. Huge pages are a feature of the Linux kernel.
	hugePagesRsvd	**Huge Pages Rsvd**	The number of committed huge pages.
	hugePagesSize	**Huge Pages Size**	The size for each huge pages unit, in kilobytes.
	hugePagesSurp	**Huge Pages Surp**	The number of available surplus huge pages over the total.

Group	Metric	Console name	Description
	hugePagesTotal	**Huge Pages Total**	The total number of huge pages.
	inactive	**Inactive Memory**	The amount of least-frequently used memory pages, in kilobytes.
	mapped	**Mapped Memory**	The total amount of file-system contents that is memory mapped inside a process address space, in kilobytes.
	pageTables	**Page Tables**	The amount of memory used by page tables, in kilobytes.
	slab	**Slab Memory**	The amount of reusable kernel data structures, in kilobytes.
	total	**Total Memory**	The total amount of memory, in kilobytes.
	writeback	**Writeback Memory**	The amount of dirty pages in RAM that are still being written to the backing storage, in kilobytes.
network	interface	Not applicable	The identifier for the network interface being used for the DB instance.
	rx	**RX**	The number of bytes received per second.
	tx	**TX**	The number of bytes uploaded per second.
processList	cpuUsedPc	**CPU %**	The percentage of CPU used by the process.
	id	Not applicable	The identifier of the process.

Group	Metric	Console name	Description
	memoryUsedPc	**MEM%**	The percentage of memory used by the process.
	name	Not applicable	The name of the process.
	parentID	Not applicable	The process identifier for the parent process of the process.
	rss	**RES**	The amount of RAM allocated to the process, in kilobytes.
	tgid	Not applicable	The thread group identifier, which is a number representing the process ID to which a thread belongs. This identifier is used to group threads from the same process.
	vss	**VIRT**	The amount of virtual memory allocated to the process, in kilobytes.
swap	total	**Swap**	The amount of swap memory available, in kilobytes.
	in	**Swaps in**	The amount of memory, in kilobytes, swapped in from disk.
	out	**Swaps out**	The amount of memory, in kilobytes, swapped out to disk.
	free	**Free Swap**	The amount of swap memory free, in kilobytes.
	cached	**Committed Swap**	The amount of swap memory, in kilobytes, used as cache memory.
tasks	blocked	**Tasks Blocked**	The number of tasks that are blocked.

Group	Metric	Console name	Description
	running	**Tasks Running**	The number of tasks that are running.
	sleeping	**Tasks Sleeping**	The number of tasks that are sleeping.
	stopped	**Tasks Stopped**	The number of tasks that are stopped.
	total	**Tasks Total**	The total number of tasks.
	zombie	**Tasks Zombie**	The number of child tasks that are inactive with an active parent task.

OS metrics for Microsoft SQL Server

RDS collects disk metrics for `rdsdbdata` as well as additional storage volumes present in the OS. To view OS metrics in CloudWatch Logs, see Viewing OS metrics using CloudWatch Logs.

Group	Metric	Console name	Description
General	engine	Not applicable	The database engine for the DB instance.
	instanceID	Not applicable	The DB instance identifier.
	instanceResourceID	Not applicable	An immutable identifier for the DB instance that is unique to an AWS Region, also used as the log stream identifier.
	numVCPUs	Not applicable	The number of virtual CPUs for the DB instance.

Group	Metric	Console name	Description
	timestamp	Not applicable	The time at which the metrics were taken.
	uptime	Not applicable	The amount of time that the DB instance has been active.
	version	Not applicable	The version of the OS metrics' stream JSON format.
cpuUtilization	idle	**CPU Idle**	The percentage of CPU that is idle.
	kern	**CPU Kernel**	The percentage of CPU in use by the kernel.
	user	**CPU User**	The percentage of CPU in use by user programs.
disks	name	Not applicable	The identifier for the disk.
	totalKb	**Total Disk Space**	The total space of the disk, in kilobytes.
	usedKb	**Used Disk Space**	The amount of space used on the disk, in kilobytes.
	usedPc	**Used Disk Space %**	The percentage of space used on the disk.
	availKb	**Available Disk Space**	The space available on the disk, in kilobytes.
	availPc	**Available Disk Space %**	The percentage of space available on the disk.
	rdCountPS	**Reads/s**	The number of read operations per second

Group	Metric	Console name	Description
	rdBytesPS	**Read Kb/s**	The number of bytes read per second.
	wrCountPS	**Write IO/s**	The number of write operations per second.
	wrBytesPS	**Write Kb/s**	The amount of bytes written per second.
memory	commitTotKb	**Commit Total**	The amount of pagefile-backed virtual address space in use, that is, the current commit charge. This value is composed of main memory (RAM) and disk (pagefiles).
	commitLimitKb	**Maximum Commit**	The maximum possible value for the commitTotKb metric. This value is the sum of the current pagefile size plus the physical memory available for pageable contents, excluding RAM that is assigned to nonpageable areas.
	commitPeakKb	**Commit Peak**	The largest value of the commitTotKb metric since the operating system was last started.
	kernTotKb	**Total Kernel Memory**	The sum of the memory in the paged and nonpaged kernel pools, in kilobytes.
	kernPagedKb	**Paged Kernel Memory**	The amount of memory in the paged kernel pool, in kilobytes.
	kernNonpagedKb	**Nonpaged Kerenel Memory**	The amount of memory in the nonpaged kernel pool, in kilobytes.
	pageSize	**Page Size**	The size of a page, in bytes.

Group	Metric	Console name	Description
	physTotKb	**Total Memory**	The amount of physical memory, in kilobytes.
	physAvailKb	**Available Memory**	The amount of available physical memory, in kilobytes.
	sqlServerTotKb	**SQL Server Total Memory**	The amount of memory committed to SQL Server, in kilobytes.
	sysCacheKb	**System Cache**	The amount of system cache memory, in kilobytes.
network	interface	Not applicable	The identifier for the network interface being used for the DB instance.
	rdBytesPS	**Network Read Kb/s**	The number of bytes received per second.
	wrBytesPS	**Network Write Kb/s**	The number of bytes sent per second.
processList	cpuUsedPc	**Used %**	The percentage of CPU used by the process.
	memUsedPc	**MEM%**	The percentage of total memory used by the process.
	name	Not applicable	The name of the process.
	pid	Not applicable	The identifier of the process. This value is not present for processes that are owned by Amazon RDS.

Group	Metric	Console name	Description
	ppid	Not applicable	The process identifier for the parent of this process. This value is only present for child processes.
	tid	Not applicable	The thread identifier. This value is only present for threads. The owning process can be identified by using the pid value.
	workingSetKb	Not applicable	The amount of memory in the private working set plus the amount of memory that is in use by the process and can be shared with other processes, in kilobytes.
	workingSetPrivKb	Not applicable	The amount of memory that is in use by a process, but can't be shared with other processes, in kilobytes.
	workingSetShareableKb	Not applicable	The amount of memory that is in use by a process and can be shared with other processes, in kilobytes.
	virtKb	Not applicable	The amount of virtual address space the process is using, in kilobytes. Use of virtual address space doesn't necessarily imply corresponding use of either disk or main memory pages.
system	handles	**Handles**	The number of handles that the system is using.
	processes	**Processes**	The number of processes running on the system.
	threads	**Threads**	The number of threads running on the system.

Monitoring events, logs, and streams in an Amazon RDS DB instance

When you monitor your Amazon RDS databases and your other AWS solutions, your goal is to maintain the following:

- Reliability
- Availability
- Performance
- Security

Monitoring metrics in an Amazon RDS instance explains how to monitor your instance using metrics. A complete solution must also monitor database events, log files, and activity streams. AWS provides you with the following monitoring tools:

- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services. EventBridge routes that data to targets such as AWS Lambda. This way, you can monitor events that happen in services and build event-driven architectures. For more information, see the Amazon EventBridge User Guide.
- *Amazon CloudWatch Logs* provides a way to monitor, store, and access your log files from Amazon RDS instances, AWS CloudTrail, and other sources. Amazon CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account. CloudTrail delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.
- *Database Activity Streams* is an Amazon RDS feature that provides a near real-time stream of the activity in your DB instance. Amazon RDS pushes activities to an Amazon Kinesis data stream. The Kinesis stream is created automatically. From Kinesis, you can configure AWS services such as Amazon Data Firehose and AWS Lambda to consume the stream and store the data.

Topics

- Viewing logs, events, and streams in the Amazon RDS console
- Monitoring Amazon RDS events
- Monitoring Amazon RDS log files
- Monitoring Amazon RDS API calls in AWS CloudTrail
- Monitoring Amazon RDS with Database Activity Streams
- Monitoring threats with Amazon GuardDuty RDS Protection

Viewing logs, events, and streams in the Amazon RDS console

Amazon RDS integrates with AWS services to show information about logs, events, and database activity streams in the RDS console.

The **Logs & events** tab for your RDS DB instance shows the following information:

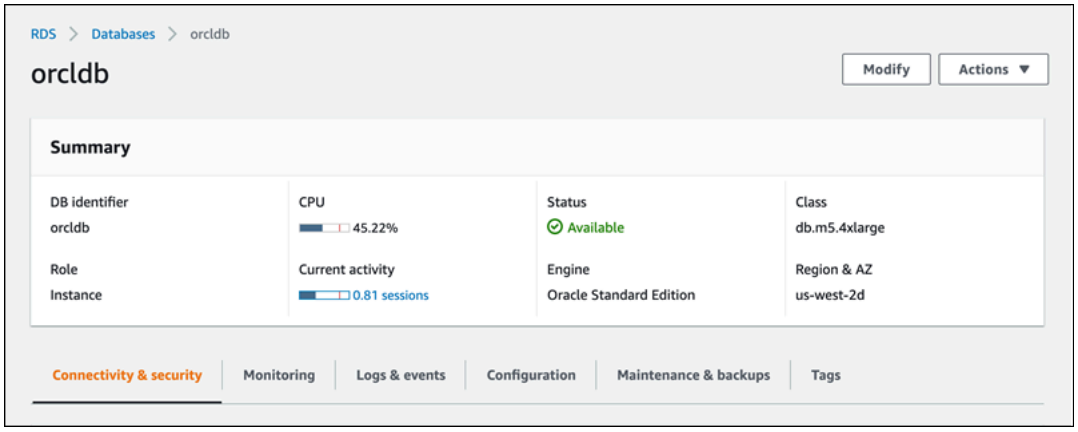
- **Amazon CloudWatch alarms** – Shows any metric alarms that you have configured for the DB instance. If you haven't configured alarms, you can create them in the RDS console. For more information, see Monitoring Amazon RDS metrics with Amazon CloudWatch.
- **Recent events** – Shows a summary of events (environment changes) for your RDS DB instance . For more information, see Viewing Amazon RDS events.
- **Logs** – Shows database log files generated by a DB instance. For more information, see Monitoring Amazon RDS log files.

The **Configuration** tab displays information about database activity streams.

To view logs, events, and streams for your DB instance in the RDS console

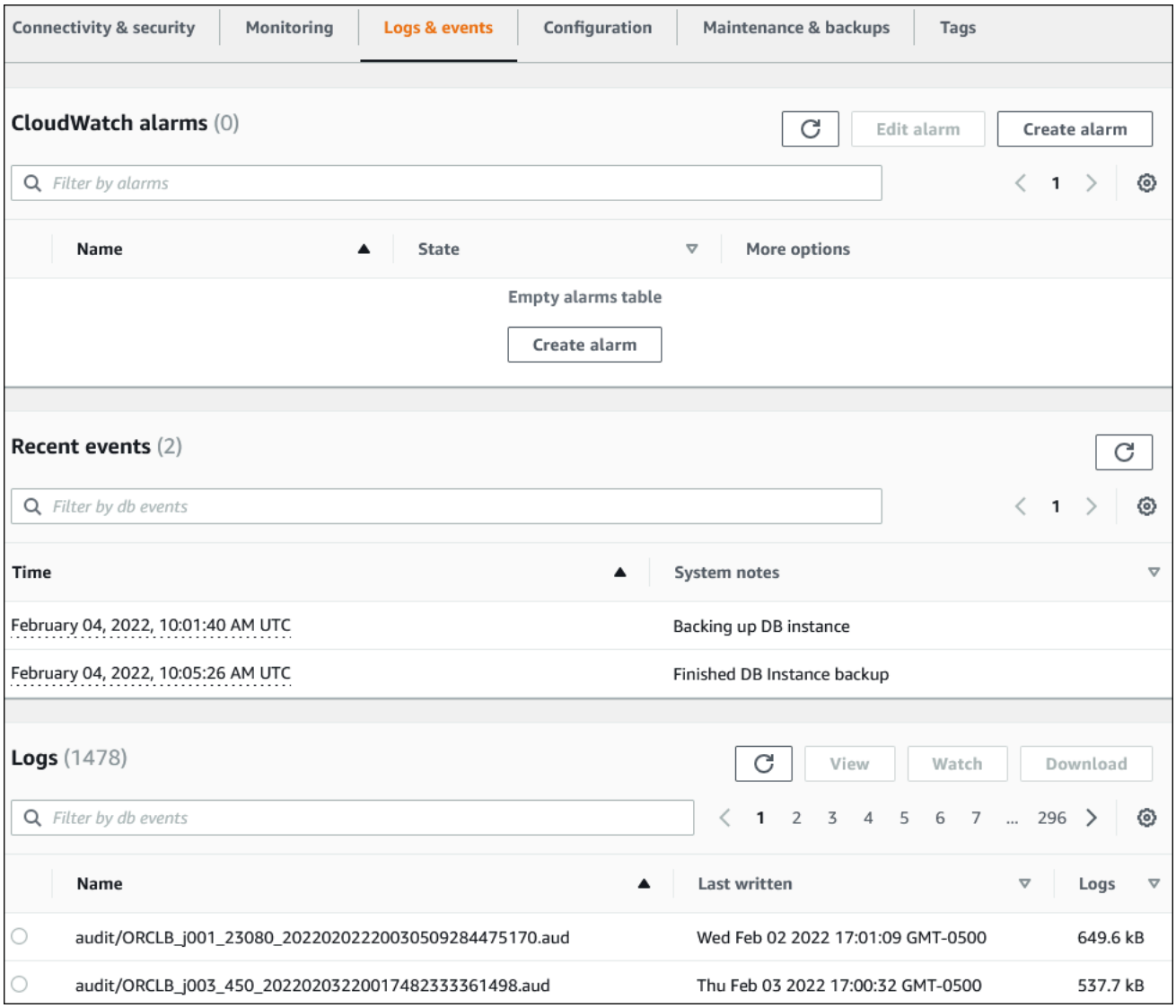
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that you want to monitor.

   The database page appears. The following example shows an Oracle database named `orclb`.



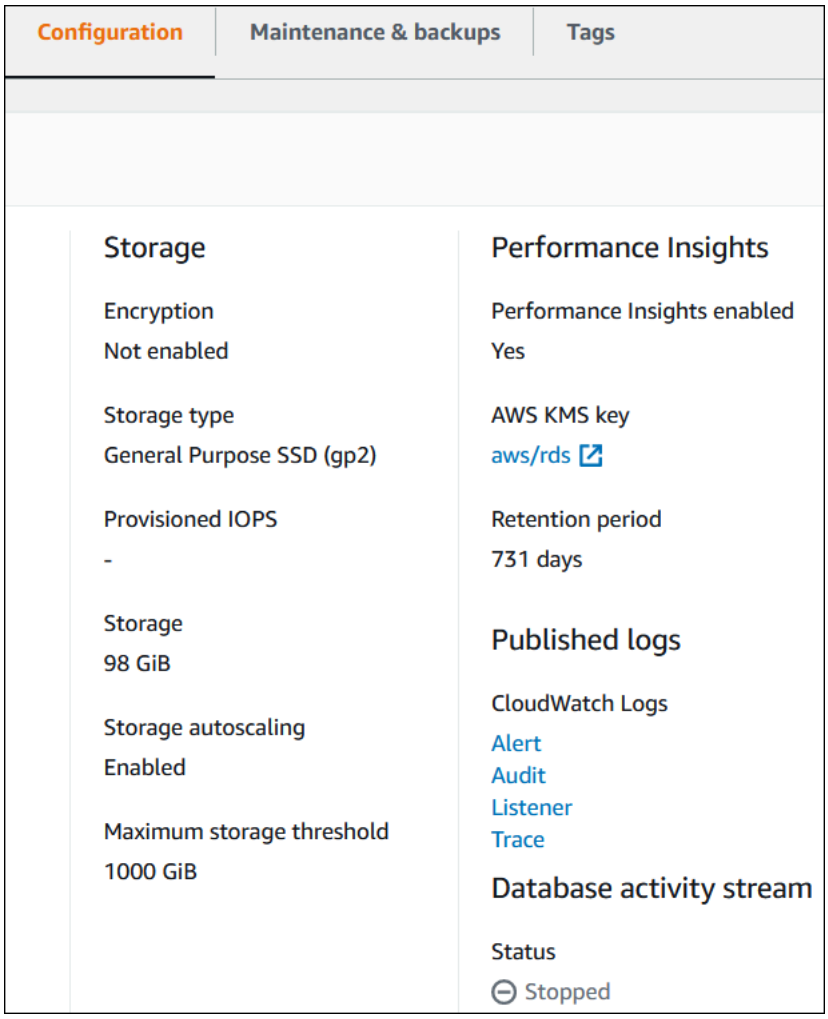
4. Choose **Logs & events**.

   The Logs & events section appears.



5. Choose **Configuration**.

   The following example shows the status of the database activity streams for your DB instance.



Monitoring Amazon RDS events

An *event* indicates a change in an environment. This can be an AWS environment, an SaaS partner service or application, or a custom application or service. For descriptions of the RDS events, see Amazon RDS event categories and event messages.

Topics

- Overview of events for Amazon RDS
- Viewing Amazon RDS events
- Working with Amazon RDS event notification
- Creating a rule that triggers on an Amazon RDS event
- Amazon RDS event categories and event messages

Overview of events for Amazon RDS

An *RDS event* indicates a change in the Amazon RDS environment. For example, Amazon RDS generates an event when the state of a DB instance changes from pending to running. Amazon RDS delivers events to EventBridge in near-real time.

> **Note**
>
> Amazon RDS emits events on a best effort basis. We recommend that you avoid writing programs that depend on the order or existence of notification events, because they might be out of sequence or missing.

Amazon RDS records events that relate to the following resources:

- DB instances

  For a list of DB instance events, see DB instance events.
- DB parameter groups

  For a list of DB parameter group events, see DB parameter group events.
- DB security groups

  For a list of DB security group events, see DB security group events.

- DB snapshots

  For a list of DB snapshot events, see DB snapshot events.
- RDS Proxy events

  For a list of RDS Proxy events, see RDS Proxy events.
- Blue/green deployment events

  For a list of blue/green deployment events, see Blue/green deployment events.

This information includes the following:

- The date and time of the event
- The source name and source type of the event
- A message associated with the event
- Event notifications include tags from when the message was sent and may not reflect tags at the time when the event occurred

Viewing Amazon RDS events

You can retrieve the following event information for your Amazon RDS resources:

- Resource name
- Resource type
- Time of the event
- Message summary of the event

You can access events in the following parts of the AWS Management Console:

- The **Events** tab, which shows events from the past 24 hours.
- The **Recent events** table in the **Logs & events** section in the **Databases** tab, which can show events for up to the past 2 weeks.

You can also retrieve events by using the describe-events AWS CLI command, or the DescribeEvents RDS API operation. If you use the AWS CLI or the RDS API to view events, you can retrieve events for up to the past 14 days.

> **Note**
>
> If you need to store events for longer periods of time, you can send Amazon RDS events to EventBridge. For more information, see Creating a rule that triggers on an Amazon RDS event

For descriptions of the Amazon RDS events, see Amazon RDS event categories and event messages.

To access detailed information about events using AWS CloudTrail, including request parameters, see CloudTrail events.

Console

To view all Amazon RDS events for the past 24 hours

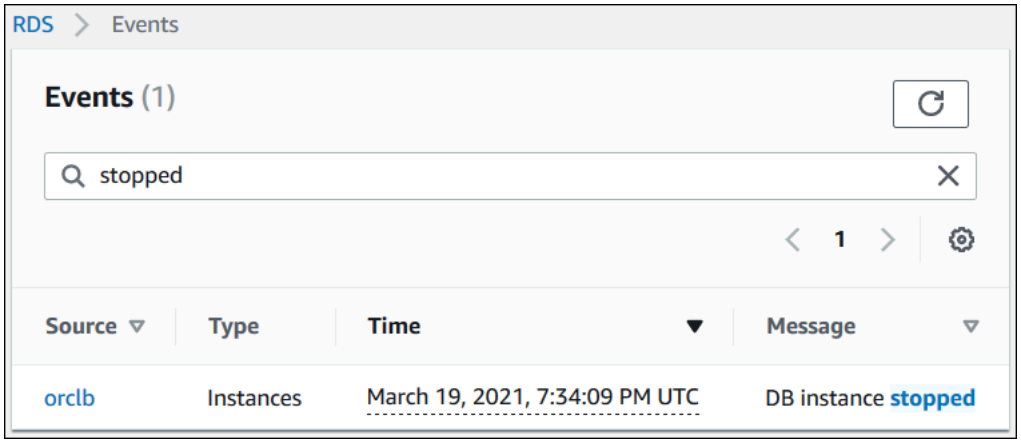
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Events**.

   The available events appear in a list.

3. (Optional) Enter a search term to filter your results.

   The following example shows a list of events filtered by the characters **`stopped`**.



AWS CLI

To view all events generated in the last hour, call describe-events with no parameters.

```
aws rds describe-events
```

The following sample output shows that a DB instance has been stopped.

```
{
    "Events": [
        {
            "EventCategories": [
                "notification"
            ],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:testinst",
            "Date": "2022-04-22T21:31:00.681Z",
            "Message": "DB instance stopped",
            "SourceIdentifier": "testinst"
        }
    ]
}
```

To view all Amazon RDS events for the past 10080 minutes (7 days), call the describe-events AWS CLI command and set the `--duration` parameter to `10080`.

```
aws rds describe-events --duration 10080
```

The following example shows the events in the specified time range for DB instance *test-instance*.

```
aws rds describe-events \
    --source-identifier test-instance \
    --source-type db-instance \
    --start-time 2022-03-13T22:00Z \
    --end-time 2022-03-13T23:59Z
```

The following sample output shows the status of a backup.

```
{
    "Events": [
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Backing up DB instance",
            "Date": "2022-03-13T23:09:23.983Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        },
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Finished DB Instance backup",
            "Date": "2022-03-13T23:15:13.049Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        }
    ]
}
```

API

You can view all Amazon RDS instance events for the past 14 days by calling the DescribeEvents RDS API operation and setting the `Duration` parameter to `20160`.

Working with Amazon RDS event notification

Amazon RDS uses the Amazon Simple Notification Service (Amazon SNS) to provide notification when an Amazon RDS event occurs. These notifications can be in any notification form supported by Amazon SNS for an AWS Region, such as an email, a text message, or a call to an HTTP endpoint.

Topics

- Overview of Amazon RDS event notification
- Granting permissions to publish notifications to an Amazon SNS topic
- Subscribing to Amazon RDS event notification
- Amazon RDS event notification tags and attributes
- Listing Amazon RDS event notification subscriptions
- Modifying an Amazon RDS event notification subscription
- Adding a source identifier to an Amazon RDS event notification subscription
- Removing a source identifier from an Amazon RDS event notification subscription
- Listing the Amazon RDS event notification categories
- Deleting an Amazon RDS event notification subscription

Overview of Amazon RDS event notification

Amazon RDS groups events into categories that you can subscribe to so that you can be notified when an event in that category occurs.

Topics

- RDS resources eligible for event subscription
- Basic process for subscribing to Amazon RDS event notifications
- Delivery of RDS event notifications
- Billing for Amazon RDS event notifications
- Examples of Amazon RDS events using Amazon EventBridge

RDS resources eligible for event subscription

You can subscribe to an event category for the following resources:

- DB instance
- DB snapshot
- DB parameter group
- DB security group
- RDS Proxy
- Custom engine version

For example, if you subscribe to the backup category for a given DB instance, you're notified whenever a backup-related event occurs that affects the DB instance. If you subscribe to a configuration change category for a DB instance, you're notified when the DB instance is changed. You also receive notification when an event notification subscription changes.

You might want to create several different subscriptions. For example, you might create one subscription that receives all event notifications for all DB instances and another subscription that includes only critical events for a subset of the DB instances. For the second subscription, specify one or more DB instances in the filter.

Basic process for subscribing to Amazon RDS event notifications

The process for subscribing to Amazon RDS event notification is as follows:

1. You create an Amazon RDS event notification subscription by using the Amazon RDS console, AWS CLI, or API.

   Amazon RDS uses the ARN of an Amazon SNS topic to identify each subscription. The Amazon RDS console creates the ARN for you when you create the subscription. Create the ARN by using the Amazon SNS console, the AWS CLI, or the Amazon SNS API.
2. Amazon RDS sends an approval email or SMS message to the addresses you submitted with your subscription.
3. You confirm your subscription by choosing the link in the notification you received.
4. The Amazon RDS console updates the **My Event Subscriptions** section with the status of your subscription.
5. Amazon RDS begins sending the notifications to the addresses that you provided when you created the subscription.

To learn about identity and access management when using Amazon SNS, see Identity and access management in Amazon SNS in the *Amazon Simple Notification Service Developer Guide*.

You can use AWS Lambda to process event notifications from a DB instance. For more information, see Using AWS Lambda with Amazon RDS in the *AWS Lambda Developer Guide*.

Delivery of RDS event notifications

Amazon RDS sends notifications to the addresses that you provide when you create the subscription. The notification can include message attributes which provide structured metadata about the message. For more information about message attributes, see Amazon RDS event categories and event messages.

Event notifications might take up to five minutes to be delivered.

> **Important**
>
> Amazon RDS doesn't guarantee the order of events sent in an event stream. The event order is subject to change.

When Amazon SNS sends a notification to a subscribed HTTP or HTTPS endpoint, the POST message sent to the endpoint has a message body that contains a JSON document. For more information, see Amazon SNS message and JSON formats in the *Amazon Simple Notification Service Developer Guide*.

You can configure SNS to notify you with text messages. For more information, see Mobile text messaging (SMS) in the *Amazon Simple Notification Service Developer Guide*.

To turn off notifications without deleting a subscription, choose **No** for **Enabled** in the Amazon RDS console. Or you can set the `Enabled` parameter to `false` using the AWS CLI or Amazon RDS API.

Billing for Amazon RDS event notifications

Billing for Amazon RDS event notification is through Amazon SNS. Amazon SNS fees apply when using event notification. For more information about Amazon SNS billing, see Amazon Simple Notification Service pricing.

Examples of Amazon RDS events using Amazon EventBridge

The following examples illustrate different types of Amazon RDS events in JSON format. For a tutorial that shows you how to capture and view events in JSON format, see Tutorial: Log DB instance state changes using Amazon EventBridge.

Topics

- Example of a DB instance event
- Example of a DB parameter group event
- Example of a DB snapshot event

Example of a DB instance event

The following is an example of a DB instance event in JSON format. The event shows that RDS performed a multi-AZ failover for the instance named `my-db-instance`. The event ID is RDS-EVENT-0049.

```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Instance Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:db:my-db-instance"
  ],
  "detail": {
    "EventCategories": [
      "failover"
    ],
    "SourceType": "DB_INSTANCE",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:my-db-instance",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "A Multi-AZ failover has completed.",
    "SourceIdentifier": "my-db-instance",
    "EventID": "RDS-EVENT-0049"
  }
}
```

Example of a DB parameter group event

The following is an example of a DB parameter group event in JSON format. The event shows that the parameter `time_zone` was updated in parameter group `my-db-param-group`. The event ID is RDS-EVENT-0037.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Parameter Group Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PARAM",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "my-db-param-group",
    "EventID": "RDS-EVENT-0037"
  }
}
```

Example of a DB snapshot event

The following is an example of a DB snapshot event in JSON format. The event shows the deletion of the snapshot named `my-db-snapshot`. The event ID is RDS-EVENT-0041.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Snapshot Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
```

```
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:snapshot:rds:my-db-snapshot"
  ],
  "detail": {
    "EventCategories": [
      "deletion"
    ],
    "SourceType": "SNAPSHOT",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:snapshot:rds:my-db-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Deleted manual snapshot",
    "SourceIdentifier": "my-db-snapshot",
    "EventID": "RDS-EVENT-0041"
  }
}
```

Granting permissions to publish notifications to an Amazon SNS topic

To grant Amazon RDS permissions to publish notifications to an Amazon Simple Notification Service (Amazon SNS) topic, attach an AWS Identity and Access Management (IAM) policy to the destination topic. For more information about permissions, see Example cases for Amazon Simple Notification Service access control in the *Amazon Simple Notification Service Developer Guide*.

By default, an Amazon SNS topic has a policy allowing all Amazon RDS resources within the same account to publish notifications to it. You can attach a custom policy to allow cross-account notifications, or to restrict access to certain resources.

The following is an example of an IAM policy that you attach to the destination Amazon SNS topic. It restricts the topic to DB instances with names that match the specified prefix. To use this policy, specify the following values:

- `Resource` – The Amazon Resource Name (ARN) for your Amazon SNS topic
- `SourceARN` – Your RDS resource ARN
- `SourceAccount` – Your AWS account ID

To see a list of resource types and their ARNs, see Resources Defined by Amazon RDS in the *Service Authorization Reference*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.rds.amazonaws.com"
      },
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"
        },
```

```
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Subscribing to Amazon RDS event notification

The simplest way to create a subscription is with the RDS console. If you choose to create event notification subscriptions using the CLI or API, you must create an Amazon Simple Notification Service topic and subscribe to that topic with the Amazon SNS console or Amazon SNS API. You will also need to retain the Amazon Resource Name (ARN) of the topic because it is used when submitting CLI commands or API operations. For information on creating an SNS topic and subscribing to it, see Getting started with Amazon SNS in the *Amazon Simple Notification Service Developer Guide*.

You can specify the type of source you want to be notified of and the Amazon RDS source that triggers the event:

Source type

The type of source. For example, **Source type** might be **Instances**. You must choose a source type.

Resources **to include**

The Amazon RDS resources that are generating the events. For example, you might choose **Select specific instances** and then **myDBInstance1**.

The following table explains the result when you specify or don't specify ***Resources*** **to include**.

Resources to include	Description	Example
Specified	RDS notifies you about all events for the specified resource only.	If your **Source type** is **Instances** and your resource is **myDBInstance1**, RDS notifies you about all events for myDBInstance1 only.
Not specified	RDS notifies you about the events for the specified source type for all your Amazon RDS resources.	If your **Source type** is **Instances**, RDS notifies you about all instance-related events in your account.

An Amazon SNS topic subscriber receives every message published to the topic by default. To receive only a subset of the messages, the subscriber must assign a filter policy to the topic subscription. For more information about SNS message filtering, see Amazon SNS message filtering in the *Amazon Simple Notification Service Developer Guide*

Console

To subscribe to RDS event notification

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In navigation pane, choose **Event subscriptions**.
3. In the **Event subscriptions** pane, choose **Create event subscription**.
4. Enter your subscription details as follows:

   a. For **Name**, enter a name for the event notification subscription.

   b. For **Send notifications to**, do one of the following:

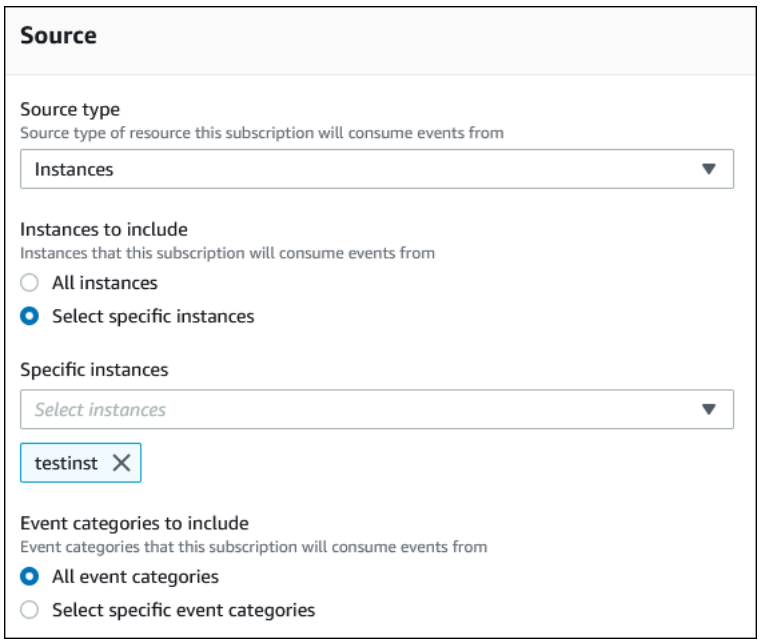
   - Choose **New email topic**. Enter a name for your email topic and a list of recipients. We recommend that you configure the events subscriptions to the same email address as your primary account contact. The recommendations, service events, and personal health messages are sent using different channels. The subscriptions to the same email address ensures that all the messages are consolidated in one location.
   - Choose **Amazon Resource Name (ARN)**. Then choose existing Amazon SNS ARN for an Amazon SNS topic.

     If you want to use a topic that has been enabled for server-side encryption (SSE), grant Amazon RDS the necessary permissions to access the AWS KMS key. For more information, see Enable compatibility between event sources from AWS services and encrypted topics in the *Amazon Simple Notification Service Developer Guide*.

   c. For **Source type**, choose a source type. For example, choose **Instances** or **Parameter groups**.

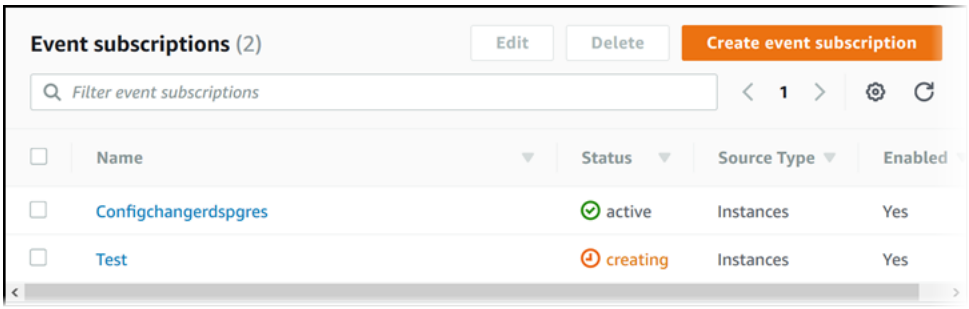
   d. Choose the event categories and resources that you want to receive event notifications for.

   The following example configures event notifications for the DB instance named `testinst`.



e. Choose **Create**.

The Amazon RDS console indicates that the subscription is being created.



AWS CLI

To subscribe to RDS event notification, use the AWS CLI `create-event-subscription` command. Include the following required parameters:

- `--subscription-name`
- `--sns-topic-arn`

Example

For Linux, macOS, or Unix:

```
aws rds create-event-subscription \
```

```
    --subscription-name myeventsubscription \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
    --enabled
```

For Windows:

```
aws rds create-event-subscription ^
    --subscription-name myeventsubscription ^
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS ^
    --enabled
```

API

To subscribe to Amazon RDS event notification, call the Amazon RDS API function `CreateEventSubscription`. Include the following required parameters:

- `SubscriptionName`
- `SnsTopicArn`

Amazon RDS event notification tags and attributes

When Amazon RDS sends an event notification to Amazon Simple Notification Service (SNS) or Amazon EventBridge, the notification contains message attributes and event tags. RDS sends the message attributes separately along with the message, while the event tags are in the body of the message. Use the message attributes and the Amazon RDS tags to add metadata to your resources. You can modify these tags with your own notations about the DB instances. For more information about tagging Amazon RDS resources, see Tagging Amazon RDS resources.

By default, the Amazon SNS and Amazon EventBridge receives every message sent to them. SNS and EventBridge can filter the message and send the notifications to the preferred communication mode, such as an email, a text message, or a call to an HTTP endpoint.

> **Note**
>
> The notification sent in an email or a text message will not have event tags.

The following table shows the message attributes for RDS events sent to the topic subscriber.

Amazon RDS event attribute	Description
EventID	Identifier for the RDS event message, for example, RDS-EVENT-0006.
Resource	The ARN identifier for the resource emitting the event, for example, `arn:aws:rds:ap-southeast-2:123456789012:db:database-1` .

The RDS tags provide data about the resource that was affected by the service event. RDS adds the current state of the tags in the message body when the notification is sent to SNS or EventBridge.

For more information about filtering message attributes for SNS, see Amazon SNS message filtering in the *Amazon Simple Notification Service Developer Guide*.

For more information about filtering event tags for EventBridge, see Comparison operators for use in event patterns in Amazon EventBridge in the *Amazon EventBridge User Guide*.

For more information about filtering payload-based tags for SNS, see Introducing payload-based message filtering for Amazon SNS

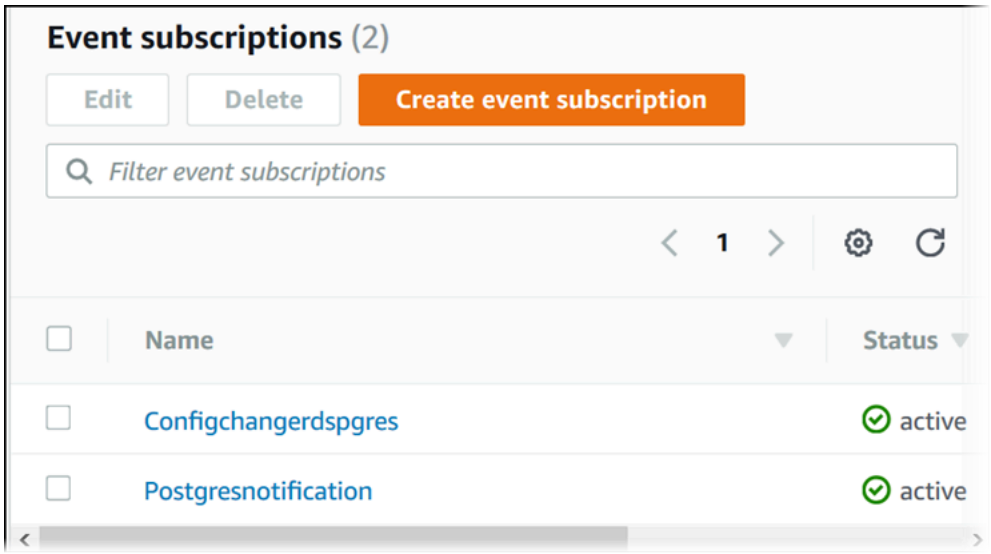
Listing Amazon RDS event notification subscriptions

You can list your current Amazon RDS event notification subscriptions.

Console

To list your current Amazon RDS event notification subscriptions

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Event subscriptions**. The **Event subscriptions** pane shows all your event notification subscriptions.



AWS CLI

To list your current Amazon RDS event notification subscriptions, use the AWS CLI `describe-event-subscriptions` command.

Example

The following example describes all event subscriptions.

```
aws rds describe-event-subscriptions
```

The following example describes the `myfirsteventsubscription`.

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

To list your current Amazon RDS event notification subscriptions, call the Amazon RDS API `DescribeEventSubscriptions` action.

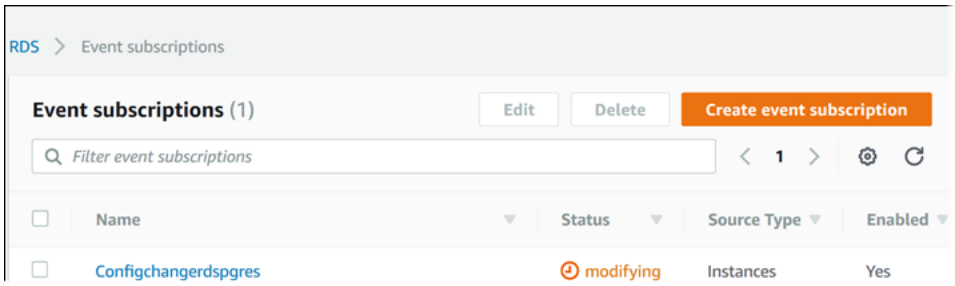
Modifying an Amazon RDS event notification subscription

After you have created a subscription, you can change the subscription name, source identifier, categories, or topic ARN.

Console

To modify an Amazon RDS event notification subscription

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Event subscriptions**.
3. In the **Event subscriptions** pane, choose the subscription that you want to modify and choose **Edit**.
4. Make your changes to the subscription in either the **Target** or **Source** section.
5. Choose **Edit**. The Amazon RDS console indicates that the subscription is being modified.



AWS CLI

To modify an Amazon RDS event notification subscription, use the AWS CLI `modify-event-subscription` command. Include the following required parameter:

- `--subscription-name`

Example

The following code enables `myeventsubscription`.

For Linux, macOS, or Unix:

```
aws rds modify-event-subscription \
```

```
    --subscription-name myeventsubscription \
    --enabled
```

For Windows:

```
aws rds modify-event-subscription ^
    --subscription-name myeventsubscription ^
    --enabled
```

API

To modify an Amazon RDS event, call the Amazon RDS API operation `ModifyEventSubscription`. Include the following required parameter:

- `SubscriptionName`

Adding a source identifier to an Amazon RDS event notification subscription

You can add a source identifier (the Amazon RDS source generating the event) to an existing subscription.

Console

You can easily add or remove source identifiers using the Amazon RDS console by selecting or deselecting them when modifying a subscription. For more information, see Modifying an Amazon RDS event notification subscription.

AWS CLI

To add a source identifier to an Amazon RDS event notification subscription, use the AWS CLI `add-source-identifier-to-subscription` command. Include the following required parameters:

- `--subscription-name`
- `--source-identifier`

Example

The following example adds the source identifier `mysqldb` to the `myrdseventsubscription` subscription.

For Linux, macOS, or Unix:

```
aws rds add-source-identifier-to-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

For Windows:

```
aws rds add-source-identifier-to-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

To add a source identifier to an Amazon RDS event notification subscription, call the Amazon RDS API `AddSourceIdentifierToSubscription`. Include the following required parameters:

- `SubscriptionName`
- `SourceIdentifier`

Removing a source identifier from an Amazon RDS event notification subscription

You can remove a source identifier (the Amazon RDS source generating the event) from a subscription if you no longer want to be notified of events for that source.

Console

You can easily add or remove source identifiers using the Amazon RDS console by selecting or deselecting them when modifying a subscription. For more information, see Modifying an Amazon RDS event notification subscription.

AWS CLI

To remove a source identifier from an Amazon RDS event notification subscription, use the AWS CLI `remove-source-identifier-from-subscription` command. Include the following required parameters:

- `--subscription-name`
- `--source-identifier`

Example

The following example removes the source identifier `mysqldb` from the `myrdseventsubscription` subscription.

For Linux, macOS, or Unix:

```
aws rds remove-source-identifier-from-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

For Windows:

```
aws rds remove-source-identifier-from-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

To remove a source identifier from an Amazon RDS event notification subscription, use the Amazon RDS API `RemoveSourceIdentifierFromSubscription` command. Include the following required parameters:

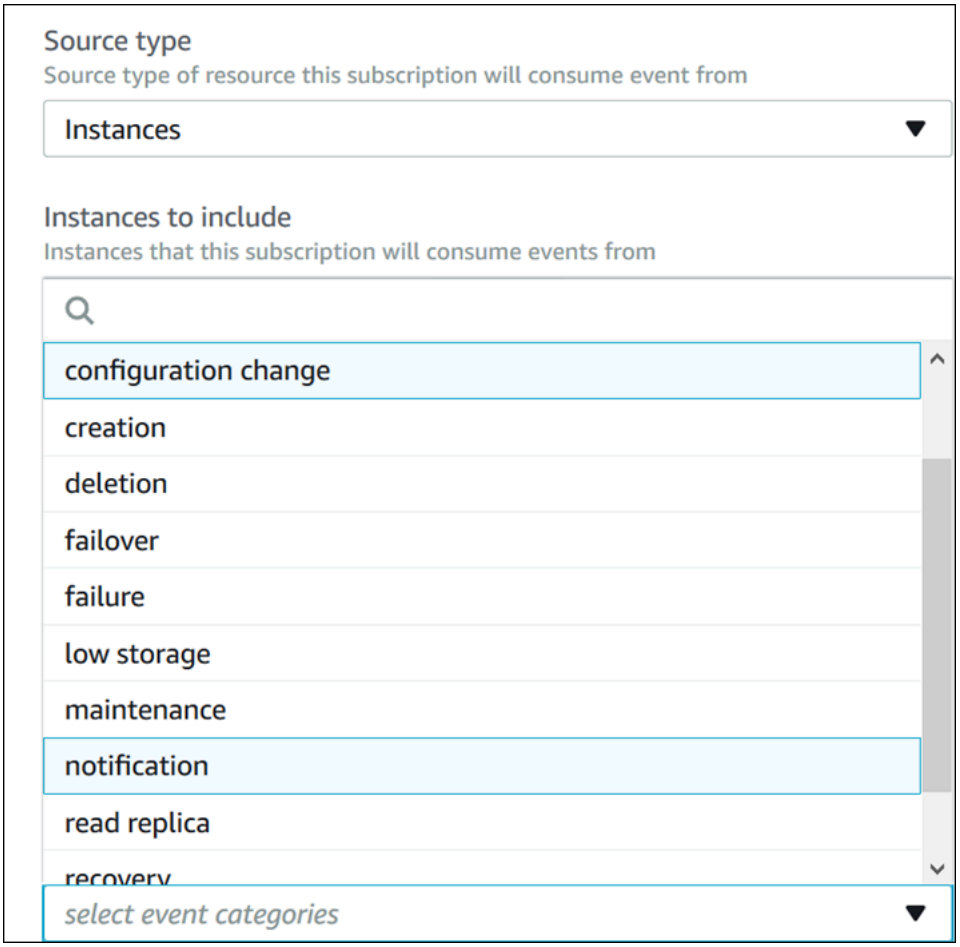
- `SubscriptionName`
- `SourceIdentifier`

Listing the Amazon RDS event notification categories

All events for a resource type are grouped into categories. To view the list of categories available, use the following procedures.

Console

When you create or modify an event notification subscription, the event categories are displayed in the Amazon RDS console. For more information, see Modifying an Amazon RDS event notification subscription.



AWS CLI

To list the Amazon RDS event notification categories, use the AWS CLI `describe-event-categories` command. This command has no required parameters.

Example

```
aws rds describe-event-categories
```

API

To list the Amazon RDS event notification categories, use the Amazon RDS API `DescribeEventCategories` command. This command has no required parameters.

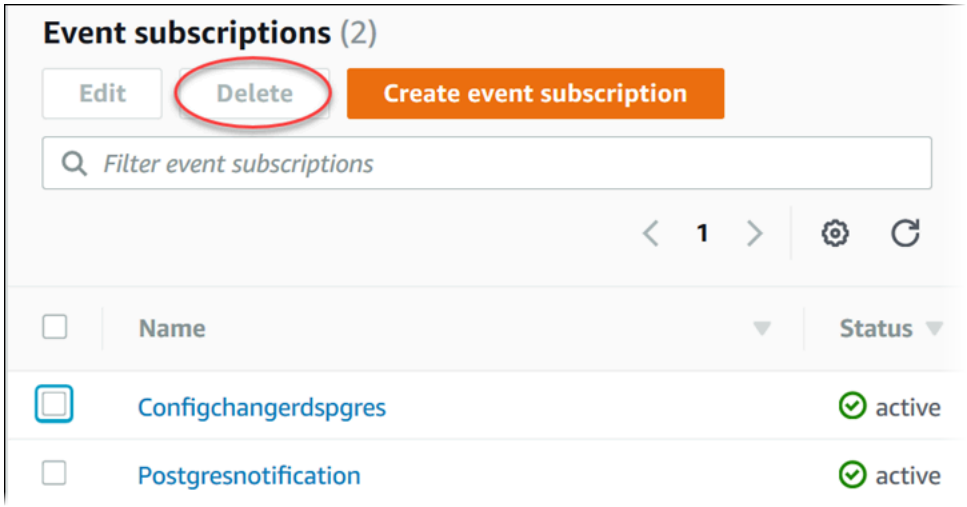
Deleting an Amazon RDS event notification subscription

You can delete a subscription when you no longer need it. All subscribers to the topic will no longer receive event notifications specified by the subscription.

Console

To delete an Amazon RDS event notification subscription

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **DB Event Subscriptions**.
3. In the **My DB Event Subscriptions** pane, choose the subscription that you want to delete.
4. Choose **Delete**.
5. The Amazon RDS console indicates that the subscription is being deleted.



AWS CLI

To delete an Amazon RDS event notification subscription, use the AWS CLI `delete-event-subscription` command. Include the following required parameter:

- `--subscription-name`

Example

The following example deletes the subscription `myrdssubscription`.

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

To delete an Amazon RDS event notification subscription, use the RDS API `DeleteEventSubscription` command. Include the following required parameter:

- `SubscriptionName`

Creating a rule that triggers on an Amazon RDS event

Using Amazon EventBridge, you can automate AWS services and respond to system events such as application availability issues or resource changes.

Topics

- Creating rules to send Amazon RDS events to Amazon EventBridge
- Tutorial: Log DB instance state changes using Amazon EventBridge

Creating rules to send Amazon RDS events to Amazon EventBridge

You can write simple rules to indicate which Amazon RDS events interest you and which automated actions to take when an event matches a rule. You can set a variety of targets, such as an AWS Lambda function or an Amazon SNS topic, which receive events in JSON format. For example, you can configure Amazon RDS to send events to Amazon EventBridge whenever a DB instance is created or deleted. For more information, see the Amazon CloudWatch Events User Guide and the Amazon EventBridge User Guide.

To create a rule that triggers on an RDS event:

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Under **Events** in the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. For **Event Source**, do the following:
    a. Choose **Event Pattern**.
    b. For **Service Name**, choose **Relational Database Service (RDS)**.
    c. For **Event Type**, choose the type of Amazon RDS resource that triggers the event. For example, if a DB instance triggers the event, choose **RDS DB Instance Event**.
5. For **Targets**, choose **Add Target** and choose the AWS service that is to act when an event of the selected type is detected.
6. In the other fields in this section, enter information specific to this target type, if any is needed.
7. For many target types, EventBridge needs permissions to send events to the target. In these cases, EventBridge can create the IAM role needed for your event to run:

- To create an IAM role automatically, choose **Create a new role for this specific resource**.
- To use an IAM role that you created before, choose **Use existing role**.

8. Optionally, repeat steps 5-7 to add another target for this rule.
9. Choose **Configure details**. For **Rule definition**, type a name and description for the rule.

   The rule name must be unique within this Region.
10. Choose **Create rule**.

For more information, see Creating an EventBridge Rule That Triggers on an Event in the *Amazon CloudWatch User Guide*.

Tutorial: Log DB instance state changes using Amazon EventBridge

In this tutorial, you create an AWS Lambda function that logs the state changes for an Amazon RDS instance. You then create a rule that runs the function whenever there is a state change of an existing RDS DB instance. The tutorial assumes that you have a small running test instance that you can shut down temporarily.

> **Important**
>
> Don't perform this tutorial on a running production DB instance.

Topics

- Step 1: Create an AWS Lambda function
- Step 2: Create a rule
- Step 3: Test the rule

Step 1: Create an AWS Lambda function

Create a Lambda function to log the state change events. You specify this function when you create your rule.

To create a Lambda function

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.

2. If you're new to Lambda, you see a welcome page. Choose **Get Started Now**. Otherwise, choose **Create function**.
3. Choose **Author from scratch**.
4. On the **Create function** page, do the following:

   a. Enter a name and description for the Lambda function. For example, name the function **`RDSInstanceStateChange`**.

   b. In **Runtime**, select **Node.js 16x**.

   c. For **Architecture**, choose **x86_64**.

   d. For **Execution role**, do either of the following:

      - Choose **Create a new role with basic Lambda permissions**.
      - For **Existing role**, choose **Use an existing role**. Choose the role that you want to use.

   e. Choose **Create function**.
5. On the **RDSInstanceStateChange** page, do the following:

   a. In **Code source**, select **index.js**.

   b. In the **index.js** pane, delete the existing code.

   c. Enter the following code:

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

   d. Choose **Deploy**.

Step 2: Create a rule

Create a rule to run your Lambda function whenever you launch an Amazon RDS instance.

To create the EventBridge rule

1. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.

4. Enter a name and description for the rule. For example, enter **`RDSInstanceStateChangeRule`**.
5. Choose **Rule with an event pattern**, and then choose **Next**.
6. For **Event source**, choose **AWS events or EventBridge partner events**.
7. Scroll down to the **Event pattern** section.
8. For **Event source**, choose **AWS services**.
9. For **AWS service**, choose **Relational Database Service (RDS)**.
10. For **Event type**, choose **RDS DB Instance Event**.
11. Leave the default event pattern. Then choose **Next**.
12. For **Target types**, choose **AWS service**.
13. For **Select a target**, choose **Lambda function**.
14. For **Function**, choose the Lambda function that you created. Then choose **Next**.
15. In **Configure tags**, choose **Next**.
16. Review the steps in your rule. Then choose **Create rule**.

Step 3: Test the rule

To test your rule, shut down an RDS DB instance. After waiting a few minutes for the instance to shut down, verify that your Lambda function was invoked.

To test your rule by stopping a DB instance

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Stop an RDS DB instance.
3. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
4. In the navigation pane, choose **Rules**, choose the name of the rule that you created.
5. In **Rule details**, choose **Monitoring**.

   You are redirected to the Amazon CloudWatch console. If you are not redirected, click **View the metrics in CloudWatch**.
6. In **All metrics**, choose the name of the rule that you created.

   The graph should indicate that the rule was invoked.
7. In the navigation pane, choose **Log groups**.

8. Choose the name of the log group for your Lambda function (**/aws/lambda/*function-name***).
9. Choose the name of the log stream to view the data provided by the function for the instance that you launched. You should see a received event similar to the following:

```
{
    "version": "0",
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "111111111111",
    "time": "2021-03-19T19:34:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:111111111111:db:testdb"
    ],
    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
        "Date": "2021-03-19T19:34:09.293Z",
        "Message": "DB instance stopped",
        "SourceIdentifier": "testdb",
        "EventID": "RDS-EVENT-0087"
    }
}
```

   For more examples of RDS events in JSON format, see Overview of events for Amazon RDS.
10. (Optional) When you're finished, you can open the Amazon RDS console and start the instance that you stopped.

Amazon RDS event categories and event messages

Amazon RDS generates a significant number of events in categories that you can subscribe to using the Amazon RDS Console, AWS CLI, or the API.

Topics

- DB cluster events
- DB cluster snapshot events
- DB instance events
- DB parameter group events
- DB security group events
- DB snapshot events
- RDS Proxy events
- Blue/green deployment events
- Custom engine version events

DB cluster events

The following table shows the event category and a list of events when a DB cluster is the source type.

For more information about Multi-AZ DB cluster deployments, see Multi-AZ DB cluster deployments for Amazon RDS.

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0016	Reset master credentials.	None
creation	RDS-EVENT-0170	DB cluster created.	None
failover	RDS-EVENT-0069	Cluster failover failed, check the health of your cluster instances and try again.	None

Category	RDS event ID	Message	Notes
failover	RDS-EVENT-0070	Promoting previous primary again: *name*.	None
failover	RDS-EVENT-0071	Completed failover to DB instance: *name*.	None
failover	RDS-EVENT-0072	Started same AZ failover to DB instance: *name*.	None
failover	RDS-EVENT-0073	Started cross AZ failover to DB instance: *name*.	None
failure	RDS-EVENT-0354	You can't create the DB cluster because of incompatible resources. *message*.	The *message* includes details about the failure.
failure	RDS-EVENT-0355	The DB cluster can't be created because of insufficient resource limits. *message*.	The *message* includes details about the failure.
maintenance	RDS-EVENT-0156	The DB cluster has a DB engine minor version upgrade available.	None
maintenance	RDS-EVENT-0173	Database cluster engine version has been upgraded.	Patching of the DB cluster has completed.
maintenance	RDS-EVENT-0174	Database cluster is in a state that cannot be upgraded.	None
maintenance	RDS-EVENT-0176	Database cluster engine major version has been upgraded.	None

Category	RDS event ID	Message	Notes
maintenance	RDS-EVENT-0177	Database cluster upgrade is in progress.	None
maintenance	RDS-EVENT-0286	Database cluster engine *version_number* version upgrade started. Cluster remains online.	None
maintenance	RDS-EVENT-0287	Operating system upgrade requirement detected.	None
maintenance	RDS-EVENT-0288	Cluster operating system upgrade starting.	None
maintenance	RDS-EVENT-0289	Cluster operating system upgrade completed.	None
maintenance	RDS-EVENT-0290	Database cluster has been patched: source version *version_number* => *new_version_number* .	None
maintenance	RDS-EVENT-0410	The pre-check started for the database cluster engine version upgrade.	None
maintenance	RDS-EVENT-0412	The pre-check for the database cluster engine version upgrade failed or timed out.	None
maintenance	RDS-EVENT-0413	The DB cluster pre-upgrade tasks are in progress.	None
maintenance	RDS-EVENT-0414	The DB cluster post-upgrade tasks are in progress.	None

Category	RDS event ID	Message	Notes
maintenance	RDS-EVENT-0417	Database cluster engine version upgrade started.	None
notification	RDS-EVENT-0172	Renamed cluster from *name* to *name*.	None
read replica	RDS-EVENT-0411	The pre-check finished for the database cluster engine version upgrade.	None

DB cluster snapshot events

The following table shows the event category and a list of events when a DB cluster snapshot is the source type.

Category	RDS event ID	Message	Notes
backup	RDS-EVENT-0074	Creating manual cluster snapshot.	None
backup	RDS-EVENT-0075	Manual cluster snapshot created.	None
notification	RDS-EVENT-0162	The cluster snapshot export task failed.	None
notification	RDS-EVENT-0163	The cluster snapshot export task was canceled.	None
notification	RDS-EVENT-0164	The cluster snapshot export task completed.	None
backup	RDS-EVENT-0168	Creating automated cluster snapshot.	None

Category	RDS event ID	Message	Notes
backup	RDS-EVENT-0169	Automated cluster snapshot created.	None
failure	RDS-EVENT-0489	Export task *name*: Table extraction failed for table *name* due to *message*.	This event is not supported by faster export.
notification	RDS-EVENT-0491	Export task *name*: Table *name* will be exported in a slower single-threaded mode. Please expect a delay in export completio n.	This event is not supported by faster export.
notification	RDS-EVENT-0492	Export task *name*: Table *name* is in progress, and the exported size is *number* GB.	This event will only be sent for tables taking more than 1 hour to complete. This event is not supported by faster export.
notification	RDS-EVENT-0493	Export task *name* is at *name* stage.	None
		Export task *name*: Export progress is at *number*% with *number* table(s) in-progress and *number* table(s) finished. The total size of finished tables is *number* GB.	This event is not supported by faster export.

DB instance events

The following table shows the event category and a list of events when a DB instance is the source type.

Category	RDS event ID	Message	Notes
availability	RDS-EVENT-0004	DB instance shutdown.	None
availability	RDS-EVENT-0006	DB instance restarted.	None
availability	RDS-EVENT-0022	Error restarting mysql: *message*.	An error has occurred while restarting MySQL.
availability	RDS-EVENT-0221	DB instance has reached the storage-full threshold , and the database has been shut down. You can increase the allocated storage to address this issue.	None
availability	RDS-EVENT-0222	Free storage capacity for DB instance *name* is low at *percentage* of the allocated storage [Allocated storage: *amount*, Free storage: *amount*]. The database will be shut down to prevent corruption if free storage is lower than *amount*. You can increase the allocated storage to address this issue.	Applies only to RDS for MySQL when a DB instance consumes more than 90% of the allocated storage. Monitor the storage space for a DB instance using the Free Storage Space metric. For more information, see Amazon RDS DB instance storage.
availability	RDS-EVENT-0330	The free storage capacity of the dedicated transaction log volume is too low for DB instance *name*. The log volume free storage is *percentage* of the allocated storage.	For more information, see Dedicated log volume (DLV).

Category	RDS event ID	Message	Notes
		[Allocated storage: *amount*, Free storage: *amount*] The database will be shut down to prevent corruption if the free storage is lower than *amount*. You can disable the dedicated transaction log volume to resolve this issue.	
availability	RDS-EVENT-0331	The free storage capacity of the dedicated transacti on log volume is too low for DB instance *name*. The log volume free storage is *percentage* of the provisioned storage. [Provisioned Storage: *amount*, Free Storage: *amount*] You can disable the dedicated transaction log volume to resolve this issue.	For more information, see Dedicated log volume (DLV).
availability	RDS-EVENT-0396	Amazon RDS has scheduled a reboot for this read replica in this instance's next maintenan ce window after internal user password rotation.	None

Category	RDS event ID	Message	Notes
availability	RDS-EVENT-0419	Amazon RDS has been unable to access the KMS encryption key for database instance *name*. This database will be placed into an inaccessi ble state. Please refer to the troubleshooting section in the Amazon RDS documentation for further details.	None
backup	RDS-EVENT-0001	Backing up DB instance.	None
backup	RDS-EVENT-0002	Finished DB instance backup.	None
backup	RDS-EVENT-0086	We are unable to associate the option group *name* with the database instance *name*. Confirm that option group *name* is supported on your DB instance class and configuration. If so, verify all option group settings and retry.	For more information see Working with option groups.
configuration change	RDS-EVENT-0011	Updated to use DBParamet erGroup *name*.	None
configuration change	RDS-EVENT-0012	Applying modification to database instance class.	None
configuration change	RDS-EVENT-0014	Finished applying modification to DB instance class.	None

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0016	Reset master credentials.	None
configuration change	RDS-EVENT-0017	Finished applying modification to allocated storage.	None
configuration change	RDS-EVENT-0018	Applying modification to allocated storage.	None
configuration change	RDS-EVENT-0024	Applying modification to convert to a Multi-AZ DB instance.	None
configuration change	RDS-EVENT-0025	Finished applying modification to convert to a Multi-AZ DB instance.	None
configuration change	RDS-EVENT-0028	Disabled automated backups.	None
configuration change	RDS-EVENT-0029	Finished applying modification to convert to a standard (Single-AZ) DB instance.	None
configuration change	RDS-EVENT-0030	Applying modification to convert to a standard (Single-AZ) DB instance.	None
configuration change	RDS-EVENT-0032	Enabled automated backups.	None

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0033	There are *number* users matching the master username; only resetting the one not tied to a specific host.	None
configuration change	RDS-EVENT-0067	Unable to reset your password. Error informati on: *message*.	None
configuration change	RDS-EVENT-0078	Monitoring Interval changed to *number*.	The Enhanced Monitorin g configuration has been changed.
configuration change	RDS-EVENT-0092	Finished updating DB parameter group.	None
configuration change	RDS-EVENT-0217	Applying autoscaling-initiated modification to allocated storage.	None
configuration change	RDS-EVENT-0218	Finished applying autoscali ng-initiated modification to allocated storage.	None
configuration change	RDS-EVENT-0295	Storage configuration upgrade started.	None
configuration change	RDS-EVENT-0296	Storage configuration upgrade completed.	None
configuration change	RDS-EVENT-0332	The dedicated log volume is disabled.	For more information, see Dedicated log volume (DLV).

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0333	Disabling the dedicated log volume has started.	For more information, see Dedicated log volume (DLV).
configuration change	RDS-EVENT-0334	Enabling the dedicated log volume has started.	For more information, see Dedicated log volume (DLV).
configuration change	RDS-EVENT-0335	The dedicated log volume is enabled.	For more information, see Dedicated log volume (DLV).
configuration change	RDS-EVENT-0383	*engine version* doesn't support the memcached plugin. RDS will continue upgrading your DB instance and remove this plugin.	Starting with MySQL 8.3.0, the memcached plugin isn't supported. For more information, see Changes in MySQL 8.3.0 (2024-01-16, Innovation Release).
creation	RDS-EVENT-0005	DB instance created.	None
deletion	RDS-EVENT-0003	DB instance deleted.	None
failover	RDS-EVENT-0013	Multi-AZ instance failover started.	A Multi-AZ failover that resulted in the promotion of a standby DB instance has started.
failover	RDS-EVENT-0015	Multi-AZ failover to standby complete - DNS propagation may take a few minutes.	A Multi-AZ failover that resulted in the promotion of a standby DB instance is complete. It may take several minutes for the DNS to transfer to the new primary DB instance.

Category	RDS event ID	Message	Notes
failover	RDS-EVENT-0034	Abandoning user requested failover since a failover recently occurred on the database instance.	Amazon RDS isn't attempting a requested failover because a failover recently occurred on the DB instance.
failover	RDS-EVENT-0049	Multi-AZ instance failover completed.	None
failover	RDS-EVENT-0050	Multi-AZ instance activation started.	A Multi-AZ activation has started after a successful DB instance recovery. This event occurs if Amazon RDS promotes the primary DB instance to the same AZ as the previous primary DB instance.
failover	RDS-EVENT-0051	Multi-AZ instance activation completed.	A Multi-AZ activation is complete. Your database should be accessible now.
failover	RDS-EVENT-0065	Recovered from partial failover.	None
failure	RDS-EVENT-0031	DB instance put into *name* state. RDS recommends that you initiate a point-in-time-restore.	The DB instance has failed due to an incompatible configuration or an underlying storage issue. Begin a point-in-time-restore for the DB instance.

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0035	Database instance put into *state*. *message*.	The DB instance has invalid parameters. For example, if the DB instance could not start because a memory-related parameter is set too high for this instance class, your action would be to modify the memory parameter and reboot the DB instance.
failure	RDS-EVENT-0036	Database instance in *state*. *message*.	The DB instance is in an incompatible network. Some of the specified subnet IDs are invalid or do not exist.
failure	RDS-EVENT-0058	The Statspack installation failed. *message*.	Error while creating Oracle Statspack user account PERFSTAT. Drop the account before you add the STATSPACK option.

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0079	Amazon RDS has been unable to create credentials for enhanced monitoring and this feature has been disabled. This is likely due to the rds-monitoring-role not being present and configured correctly in your account. Please refer to the troubleshooting section in the Amazon RDS documentation for further details.	Enhanced Monitoring can't be enabled without the Enhanced Monitoring IAM role. For information about creating the IAM role, see To create an IAM role for Amazon RDS enhanced monitoring.
failure	RDS-EVENT-0080	Amazon RDS has been unable to configure enhanced monitoring on your instance: *name* and this feature has been disabled. This is likely due to the rds-monitoring-role not being present and configured correctly in your account. Please refer to the troubleshooting section in the Amazon RDS documentation for further details.	Enhanced Monitoring was disabled because an error occurred during the configuration change. It is likely that the Enhanced Monitoring IAM role is configured incorrectly. For information about creating the enhanced monitoring IAM role, see To create an IAM role for Amazon RDS enhanced monitoring.

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0081	Amazon RDS has been unable to create credentials for *name* option. This is due to the *name* IAM role not being configured correctly in your account. Please refer to the troubleshooting section in the Amazon RDS documentation for further details.	The IAM role that you use to access your Amazon S3 bucket for SQL Server native backup and restore is configured incorrectly. For more information, see Setting up for native backup and restore.
failure	RDS-EVENT-0165	The RDS Custom DB instance is outside the support perimeter.	It's your responsibility to fix configuration issues that put your RDS Custom DB instance into the `unsupported-configuration` state. If the issue is with the AWS infrastructure, you can use the console or the AWS CLI to fix it. If the issue is with the operating system or the database configuration, you can log in to the host to fix it. For more information, see RDS Custom support perimeter.

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0188	The DB instance is in a state that can't be upgraded. *message*	Amazon RDS was unable to upgrade a MySQL DB instance because of incompatibilities related to the data dictionary. The DB instance was rolled back to MySQL version 5.7 because an attempted upgrade to version 8.0 failed, or rolled back to MySQL version 8.0 because an attempted upgrade to version 8.4 failed. For more informati on, see Rollback after failure to upgrade.
failure	RDS-EVENT-0219	DB instance is in an invalid state. No actions are necessary. Autoscaling will retry later.	None
failure	RDS-EVENT-0220	DB instance is in the cooling-off period for a previous scale storage operation. We're optimizin g your DB instance. This takes at least 6 hours. No actions are necessary. Autoscaling will retry after the cooling-off period.	None
failure	RDS-EVENT-0223	Storage autoscaling is unable to scale the storage for the reason: *reason*.	None

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0224	Storage autoscaling has triggered a pending scale storage task that will reach or exceed the maximum storage threshold. Increase the maximum storage threshold.	None
failure	RDS-EVENT-0237	DB instance has a storage type that's currently unavailable in the Availabil ity Zone. Autoscaling will retry later.	None
failure	RDS-EVENT-0254	Underlying storage quota for this customer account has exceeded the limit. Please increase the allowed storage quota to let the scaling go through on the instance.	None
failure	RDS-EVENT-0278	The DB instance creation failed. *message*	The *message* includes details about the failure.
failure	RDS-EVENT-0279	The promotion of the RDS Custom read replica failed. *message*	The *message* includes details about the failure.
failure	RDS-EVENT-0280	RDS Custom couldn't upgrade the DB instance because the pre-check failed. *message*	The *message* includes details about the failure.

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0281	RDS Custom couldn't modify the DB instance because the pre-check failed. *message*	The *message* includes details about the failure.
failure	RDS-EVENT-0282	RDS Custom couldn't modify the DB instance because the Elastic IP permissions aren't correct. Please confirm the Elastic IP address is tagged with `AWSRDSCustom` .	None
failure	RDS-EVENT-0283	RDS Custom couldn't modify the DB instance because the Elastic IP limit has been reached in your account. Release unused Elastic IPs or request a quota increase for your Elastic IP address limit.	None
failure	RDS-EVENT-0284	RDS Custom couldn't convert the instance to high availability because the pre-check failed. *message*	The *message* includes details about the failure.
failure	RDS-EVENT-0285	RDS Custom couldn't create a final snapshot for the DB instance because *message*.	The *message* includes details about the failure.

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0421	RDS Custom couldn't convert the DB instance to a Multi-AZ deployment: *message*. The instance will remain a Single-AZ deployment. See the RDS User Guide for information about Multi-AZ deployments for RDS Custom for Oracle.	The *message* includes details about the failure.
failure	RDS-EVENT-0306	Storage configuration upgrade failed. Please retry the upgrade.	None
failure	RDS-EVENT-0315	Unable to move incompatible-network database, *name*, to the available status: *message*	The database networking configuration is invalid. The database could not be moved from incompatible-network to available.
failure	RDS-EVENT-0328	Failed to join a host to a domain. Domain membership status for instance *instancename* has been set to Failed.	None

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0329	Failed to join a host to your domain. During the domain join process, Microsoft Windows returned the error code *message*. Verify your network and permission configurations and issue a `modify-db-instance` request to re-attempt the domain join.	When using a self-managed Active Directory, see Troubleshooting self-managed Active Directory.
failure	RDS-EVENT-0353	The DB instance can't be created because of insufficient resource limits. *message*.	The *message* includes details about the failure.
failure	RDS-EVENT-0356	RDS was unable to configure the Kerberos endpoint in your domain. This might prevent Kerberos authentication for your DB instance. Verify the network configuration between your DB instance and domain controllers.	None

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0418	Amazon RDS is unable to access the KMS encryption key for database instance *name*. This is likely due to the key being disabled or Amazon RDS being unable to access it. If this continues the database will be placed into an inaccessible state. Please refer to the troublesh ooting section in the Amazon RDS documenta tion for further details.	None
failure	RDS-EVENT-0420	Amazon RDS can now successfully access the KMS encryption key for database instance *name*.	None
low storage	RDS-EVENT-0007	Allocated storage has been exhausted. Allocate additional storage to resolve.	The allocated storage for the DB instance has been consumed. To resolve this issue, allocate additional storage for the DB instance. For more information, see the RDS FAQ. You can monitor the storage space for a DB instance using the **Free Storage Space** metric.

Category	RDS event ID	Message	Notes
low storage	RDS-EVENT-0089	The free storage capacity for DB instance: *name* is low at *percentage* of the provisioned storage [Provisioned Storage: *size*, Free Storage: *size*]. You may want to increase the provisioned storage to address this issue.	The DB instance has consumed more than 90% of its allocated storage. You can monitor the storage space for a DB instance using the **Free Storage Space** metric.
low storage	RDS-EVENT-0227	Your Aurora cluster's storage is dangerously low with only *amount* terabytes remaining. Please take measures to reduce the storage load on your cluster.	The Aurora storage subsystem is running low on space.
maintenance	RDS-EVENT-0026	Applying off-line patches to DB instance.	Offline maintenance of the DB instance is taking place. The DB instance is currently unavailable.
maintenance	RDS-EVENT-0027	Finished applying off-line patches to DB instance.	Offline maintenance of the DB instance is complete. The DB instance is now available.
maintenance	RDS-EVENT-0047	Database instance patched.	None
maintenance	RDS-EVENT-0155	The DB instance has a DB engine minor version upgrade available.	None

Category	RDS event ID	Message	Notes
maintenance	RDS-EVENT-0178	Database instance upgrade is in progress.	None
maintenance	RDS-EVENT-0264	The pre-check started for the DB engine version upgrade.	None
maintenance	RDS-EVENT-0265	The pre-check finished for the DB engine version upgrade.	None
maintenance	RDS-EVENT-0266	The downtime started for the DB instance.	None
maintenance	RDS-EVENT-0267	The engine version upgrade started.	None
maintenance	RDS-EVENT-0268	The engine version upgrade finished.	None
maintenance	RDS-EVENT-0269	The post-upgrade tasks are in progress.	None
maintenance	RDS-EVENT-0270	The DB engine version upgrade failed. The engine version upgrade rollback succeeded.	None
maintenance	RDS-EVENT-0398	Waiting for the DB engine version upgrade to finish on the primary DB instance.	Emitted on a read replica during a major engine version upgrade.
maintenance	RDS-EVENT-0399	Waiting for the DB engine version upgrade to finish on the read replicas.	Emitted on source DB engine during a major engine version upgrade.

Category	RDS event ID	Message	Notes
maintenance	RDS-EVENT-0422	RDS will replace the host of DB instance *name* due to a pending maintenance action.	None
maintenance, failure	RDS-EVENT-0195	*message*	The update of the Oracle time zone file failed. For more information, see Oracle time zone file autoupgrade.
maintenance, notification	RDS-EVENT-0191	A new version of the time zone file is available for update.	If you update your RDS for Oracle DB engine, Amazon RDS generates this event if you haven't chosen a time zone file upgrade and the database doesn’t use the latest DST time zone file available on the instance. For more information, see Oracle time zone file autoupgrade.
maintenance, notification	RDS-EVENT-0192	The update of your time zone file has started.	The upgrade of your Oracle time zone file has begun. For more informati on, see Oracle time zone file autoupgrade.

Category	RDS event ID	Message	Notes
maintenance, notification	RDS-EVENT-0193	No update is available for the current time zone file version.	Your Oracle DB instance is using latest time zone file version, and either of the following statements is true: • You recently added the `TIMEZONE_FILE_AUTOUPGRADE` option. • Your Oracle DB engine is being upgraded. For more information, see Oracle time zone file autoupgrade.
maintenance, notification	RDS-EVENT-0194	The update of your time zone file has finished.	The update of your Oracle time zone file has completed. For more information, see Oracle time zone file autoupgrade.
notification	RDS-EVENT-0044	*message*	This is an operator-issued notification. For more information, see the event message.
notification	RDS-EVENT-0048	Delaying database engine upgrade since this instance has read replicas that need to be upgraded first.	Patching of the DB instance has been delayed.

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0054	*message*	The MySQL storage engine you are using is not InnoDB, which is the recommended MySQL storage engine for Amazon RDS. For information about MySQL storage engines, see Supported storage engines for RDS for MySQL.
notification	RDS-EVENT-0055	*message*	The number of tables you have for your DB instance exceeds the recommended best practices for Amazon RDS. Reduce the number of tables on your DB instance. For information about recommended best practices, see Amazon RDS basic operational guidelines.
notification	RDS-EVENT-0056	*message*	The number of databases you have for your DB instance exceeds the recommended best practices for Amazon RDS. Reduce the number of databases on your DB instance. For information about recommended best practices, see Amazon RDS basic operational guidelines.

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0064	The TDE encryption key was rotated successfully.	For information about recommended best practices, see Amazon RDS basic operational guidelines.
notification	RDS-EVENT-0084	Unable to convert the DB instance to Multi-AZ: *message*.	You attempted to convert a DB instance to Multi-AZ, but it contains in-memory file groups that are not supported for Multi-AZ. For more information, see Multi-AZ deployments for Amazon RDS for Microsoft SQL Server.
notification	RDS-EVENT-0087	DB instance stopped.	None
notification	RDS-EVENT-0088	DB instance started.	None
notification	RDS-EVENT-0154	DB instance is being started due to it exceeding the maximum allowed time being stopped.	None

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0157	Unable to modify the DB instance class. *message*.	RDS can't modify the DB instance class because the target instance class can't support the number of databases that exist on the source DB instance. The error message appears as: "The instance has *N* databases, but after conversion it would only support *N*". For more information, see Limitations for Microsoft SQL Server DB instances.
notification	RDS-EVENT-0158	Database instance is in a state that cannot be upgraded: *message*.	None
notification	RDS-EVENT-0167	*message*	The RDS Custom support perimeter configuration has changed.

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0189	The gp2 burst balance credits for the RDS database instance are low. To resolve this issue, reduce IOPS usage or modify your storage settings to enable higher performance.	The gp2 burst balance credits for the RDS database instance are low. To resolve this issue, reduce IOPS usage or modify your storage settings to enable higher performance. For more information, see I/O credits and burst performance in the *Amazon Elastic Compute Cloud User Guide*.
notification	RDS-EVENT-0225	Allocated storage size *amount* GB is approaching the maximum storage threshold *amount* GB. Increase the maximum storage threshold.	This event is invoked when the allocated storage reaches 80% of the maximum storage threshold. To avoid the event, increase the maximum storage threshold.

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0231	Your DB instance's storage modification encounter ed an internal error. The modification request is pending and will be retried later.	An error has occurred in the read replicati on process. For more information, see the event message. In addition, see the troubleshooting section for read replicas for your DB engine. • Troubleshooting a MariaDB read replica problem • Troubleshooting a SQL Server read replica problem • Troubleshooting a MySQL read replica problem • Troubleshooting RDS for Oracle replicas

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0253	The database is using the doublewrite buffer. *message*. For more information see the RDS Optimized Writes for *name* documentation.	RDS Optimized Writes is incompatible with the instance storage configuration. For more information, see Improving write performance with RDS Optimized Writes for MySQL and Improving write performance with Amazon RDS Optimized Writes for MariaDB. You can perform storage configuration upgrade to enable Optimized Writes by Creating a blue/green deployment.
notification	RDS-EVENT-0297	The storage configuration for DB instance *name* supports a maximum size of 16384 GiB. Perform a storage configuration upgrade to support storage sizes greater than 16384 GiB.	You cannot increase the allocated storage size of the DB instance beyond 16384 GiB. To overcome this limitation, perform a storage configuration upgrade. For more information, see Upgrading the storage file system for a DB instance.

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0298	The storage configuration for DB instance *name* supports a maximum table size of 2048 GiB. Perform a storage configuration upgrade to support table sizes greater than 2048 GiB.	RDS MySQL and MariaDB instances with this limitation cannot have a table size exceeding 2048 GiB. To overcome this limitation, perform a storage configuration upgrade. For more information, see Upgrading the storage file system for a DB instance.
notification	RDS-EVENT-0327	Amazon RDS could not find the secret *SECRET ARN*. *message.*	None
notification	RDS-EVENT-0365	Timezone files were updated. Restart your RDS instance for the changes to take effect.	None
notification	RDS-EVENT-0385	Cluster topology is updated.	There are DNS changes to the DB cluster for the DB instance. This includes when new DB instances are added or deleted, or there's a failover.
notification	RDS-EVENT-0403	A database workload is causing the system to run critically low on memory. To help mitigate the issue, RDS automatically set the value of innodb_buffer_pool_size to *amount*.	Applies only to RDS for MySQL and RDS for MariaDB DB instances.

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0404	A database workload is causing the system to run critically low on memory. To help mitigate the issue, RDS automatically set the value of shared_buffers to *amount*.	Applies only to RDS for PostgreSQL DB instances.
read replica	RDS-EVENT-0045	Replication has stopped.	This message appears when there is an error during replication. To determine the type of error, see Troubleshooting a MySQL read replica problem.
read replica	RDS-EVENT-0046	Replication for the Read Replica resumed.	This message appears when you first create a read replica, or as a monitoring message confirming that replication is functioning properly. If this message follows an RDS-EVENT-0045 notification, then replication has resumed following an error or after replication was stopped.
read replica	RDS-EVENT-0057	Replication streaming has been terminated.	None
read replica	RDS-EVENT-0062	Replication for the Read Replica has been manually stopped.	None

Category	RDS event ID	Message	Notes
read replica	RDS-EVENT-0063	Replication from Non RDS instance has been reset.	None
read replica	RDS-EVENT-0202	Read replica creation failed.	None
read replica	RDS-EVENT-0233	The Switchover to the read replica started.	None
read replica	RDS-EVENT-0357	Replication channel *name* started.	For information about replication channels, see the section called "Configuring multi-source replication".
read replica	RDS-EVENT-0358	Replication channel *name* stopped.	For information about replication channels, see the section called "Configuring multi-source replication".
read replica	RDS-EVENT-0359	Replication channel *name* was manually stopped.	For information about replication channels, see the section called "Configuring multi-source replication".
read replica	RDS-EVENT-0360	Replication channel *name* was reset.	For information about replication channels, see the section called "Configuring multi-source replication".
read replica	RDS-EVENT-0415	The upgrade process resumed replication on the read replica.	None

Category	RDS event ID	Message	Notes
read replica	RDS-EVENT-0416	The upgrade process stopped replication on the read replica.	None
recovery	RDS-EVENT-0020	Recovery of the DB instance has started. Recovery time will vary with the amount of data to be recovered.	None
recovery	RDS-EVENT-0021	Recovery of the DB instance is complete.	None
recovery	RDS-EVENT-0023	Emergent Snapshot Request: *message*.	A manual backup has been requested but Amazon RDS is currently in the process of creating a DB snapshot. Submit the request again after Amazon RDS has completed the DB snapshot.
recovery	RDS-EVENT-0052	Multi-AZ instance recovery started.	Recovery time will vary with the amount of data to be recovered.
recovery	RDS-EVENT-0053	Multi-AZ instance recovery completed. Pending failover or activation.	This message indicates that Amazon RDS has prepared your DB instance to initiate a failover to the secondary instance if necessary.

Category	RDS event ID	Message	Notes
recovery	RDS-EVENT-0066	Instance will be degraded while mirroring is reestabli shed: *message*.	The SQL Server DB instance is re-establishing its mirror. Performance will be degraded until the mirror is reestablished. A database was found with non-FULL recovery model. The recovery model was changed back to FULL and mirroring recovery was started. (<dbname>: <recovery model found>[,. ..])"
recovery	RDS-EVENT-0166	*message*	The RDS Custom DB instance is inside the support perimeter.
recovery	RDS-EVENT-0361	Recovery of standby DB instance has started.	The standby DB instance is rebuilt during the recovery process. Database performance is impacted during the recovery process.
recovery	RDS-EVENT-0362	Recovery of standby DB instance has completed.	The standby DB instance is rebuilt during the recovery process. Database performance is impacted during the recovery process.
restoration	RDS-EVENT-0019	Restored from DB instance *name* to *name*.	The DB instance has been restored from a point-in-time backup.

Category	RDS event ID	Message	Notes
security	RDS-EVENT-0068	Decrypting hsm partition password to update instance.	RDS is decrypting the AWS CloudHSM partition password to make updates to the DB instance. For more information see Oracle Database Transparent Data Encryption (TDE) with AWS CloudHSM in the *AWS CloudHSM User Guide*.
security patching	RDS-EVENT-0230	A system update is available for your DB instance. For information about applying updates, see 'Maintaining a DB instance' in the RDS User Guide.	A new Operating System update is available. A new, minor version, operating system update is available for your DB instance. For information about applying updates, see Operating system updates for RDS DB instances.
maintenance	RDS-EVENT-0425	Amazon RDS can't perform the OS upgrade because there are no available IP addresses in the specified subnets. Choose subnets with available IP addresses and try again.	None

Category	RDS event ID	Message	Notes
maintenance	RDS-EVENT-0429	Amazon RDS can't perform the OS upgrade because of insufficient capacity available for the *type* instance type in the *zone* Availability Zone	None
maintenance	RDS-EVENT-0501	Amazon RDS DB instance's server certificate requires rotation through a pending maintenance action.	DB instance's server certificate requires rotation through a pending maintenance action. Amazon RDS reboots your database during this maintenance to complete the certificate rotation. To schedule this maintenance, go to the **Maintenance & backups** tab and choose **Apply now** or **Schedule for next maintenance window**. If the change is not scheduled, Amazon RDS automatically applies it in your mainteance window on the auto apply date shown in your maintenance action.

Category	RDS event ID	Message	Notes
maintenance	RDS-EVENT-0502	Amazon RDS has scheduled a server certificate rotation for DB instance during the next maintenance window. This maintenance will require a database reboot.	None

DB parameter group events

The following table shows the event category and a list of events when a DB parameter group is the source type.

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0037	Updated parameter *name* to *value* with apply method *method*.	None

DB security group events

The following table shows the event category and a list of events when a DB security group is the source type.

> **Note**
>
> DB security groups are resources for EC2-Classic. EC2-Classic was retired on August 15, 2022. If you haven't migrated from EC2-Classic to a VPC, we recommend that you migrate as soon as possible. For more information, see Migrate from EC2-Classic to a VPC in the *Amazon EC2 User Guide* and the blog EC2-Classic Networking is Retiring – Here's How to Prepare.

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0038	Applied change to security group.	None
failure	RDS-EVENT-0039	Revoking authorization as *user*.	The security group owned by *user* doesn't exist. The authorization for the security group has been revoked because it is invalid.

DB snapshot events

The following table shows the event category and a list of events when a DB snapshot is the source type.

Category	RDS event ID	Message	Notes
creation	RDS-EVENT-0040	Creating manual snapshot.	None
creation	RDS-EVENT-0042	Manual snapshot created.	None
creation	RDS-EVENT-0090	Creating automated snapshot.	None
creation	RDS-EVENT-0091	Automated snapshot created.	None
deletion	RDS-EVENT-0041	Deleted user snapshot.	None
notification	RDS-EVENT-0059	Started copy of snapshot*name* from region *name*.	This is a cross-Region snapshot copy.
notification	RDS-EVENT-0060	Finished copy of snapshot *name* from region *name* in *number* minutes.	This is a cross-Region snapshot copy.

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0061	Canceled snapshot copy request of *name* from region *name*.	This is a cross-Region snapshot copy.
notification	RDS-EVENT-0159	The snapshot export task failed.	None
notification	RDS-EVENT-0160	The snapshot export task was canceled.	None
notification	RDS-EVENT-0161	The snapshot export task completed.	None
notification	RDS-EVENT-0196	Started copy of snapshot *name* in region *name*.	This is a local snapshot copy.
notification	RDS-EVENT-0197	Finished copy of snapshot *name* in region *name*.	This is a local snapshot copy.
notification	RDS-EVENT-0190	Canceled snapshot copy request of *name* in region *name*.	This is a local snapshot copy.
restoration	RDS-EVENT-0043	Restored from snapshot *name*.	A DB instance is being restored from a DB snapshot.
failure	RDS-EVENT-0484	Export task *name*: Table extraction failed for table *name* due to *name*.	This event is not supported by faster export.
notification	RDS-EVENT-0486	Export task *name*: Table *name* will be exported in a slower single-threaded mode. Please expect a delay in export completio n.	This event is not supported by faster export.

Category	RDS event ID	Message	Notes
notification	RDS-EVENT-0487	Export task *name*: Table *name* is in progress, and the exported size is *number* GB.	This event will only be sent for tables taking more than 1 hour to complete. This event is not supported by faster export.
notification	RDS-EVENT-0488	Export task *name* is at *name* stage.	None
		Export task *name*: Export progress is at *number*% with *number* table(s) in-progress and *number* table(s) finished. The total size of finished tables is *number* GB.	This event is not supported by faster export.

RDS Proxy events

The following table shows the event category and a list of events when an RDS Proxy is the source type.

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0204	RDS modified DB proxy *name*.	None
configuration change	RDS-EVENT-0207	RDS modified the end point of the DB proxy *name*.	None
configuration change	RDS-EVENT-0213	RDS detected the addition of the DB instance and automatically added it to	None

Category	RDS event ID	Message	Notes
		the target group of the DB proxy *name*.	
configuration change	RDS-EVENT-0214	RDS detected deletion of DB instance *name* and automatically removed it from target group *name* of DB proxy *name*.	None
configuration change	RDS-EVENT-0215	RDS detected deletion of DB cluster *name* and automatically removed it from target group *name* of DB proxy *name*.	None
creation	RDS-EVENT-0203	RDS created DB proxy *name*.	None
creation	RDS-EVENT-0206	RDS created endpoint *name* for DB proxy *name*.	None
deletion	RDS-EVENT-0205	RDS deleted DB proxy *name*.	None
deletion	RDS-EVENT-0208	RDS deleted endpoint *name* for DB proxy *name*.	None

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0243	RDS failed to provision capacity for proxy *name* because there aren't enough IP addresses available in your subnets: *name*. To fix the issue, make sure that your subnets have the minimum number of unused IP addresses as recommend ed in the RDS Proxy documentation.	To determine the recommended number for your instance class, see Planning for IP address capacity.
failure	RDS-EVENT-0275	RDS throttled some connections to DB proxy *name*. The number of simultaneous connection requests from the client to the proxy has exceeded the limit.	None

Blue/green deployment events

The following table shows the event category and a list of events when a blue/green deployment is the source type.

For more information about blue/green deployments, see Using Amazon RDS Blue/Green Deployments for database updates.

Category	Amazon RDS event ID	Message	Notes
creation	RDS-EVENT-0244	Blue/green deployment tasks completed. You can make more modifications	None

Category	Amazon RDS event ID	Message	Notes
		to the green environment databases or switch over the deployment.	
failure	RDS-EVENT-0245	Creation of blue/green deployment failed because *reason*.	None
deletion	RDS-EVENT-0246	Blue/green deployment deleted.	None
notification	RDS-EVENT-0247	Switchover from *blue* to *green* started.	None
notification	RDS-EVENT-0248	Switchover completed on blue/green deployment.	None
failure	RDS-EVENT-0249	Switchover canceled on blue/green deployment.	None
notification	RDS-EVENT-0250	Switchover from primary/read replica *blue* to *green* started.	None
notification	RDS-EVENT-0251	Switchover from primary/read replica *blue* to *green* completed. Renamed *blue* to *blue-old* and *green* to *blue*.	None
failure	RDS-EVENT-0252	Switchover from primary/read replica *blue* to *green* was canceled due to *reason*.	None

Category	Amazon RDS event ID	Message	Notes
notification	RDS-EVENT-0307	Sequence sync for switchover of *blue* to *green* has initiated. Switchover when using sequences may lead to extended downtime.	None
notification	RDS-EVENT-0308	Sequence sync for switchover of *blue* to *green* has completed.	None
failure	RDS-EVENT-0310	Sequence sync for switchover of *blue* to *green* was cancelled because sequences failed to sync.	None
notification	RDS-EVENT-0405	Your storage volumes are being initialized.	None
notification	RDS-EVENT-0406	Your storage volumes have been initialized.	None
notification	RDS-EVENT-0409	*message*	None

Custom engine version events

The following table shows the event category and a list of events when a custom engine version is the source type.

Category	Amazon RDS event ID	Message	Notes
creation	RDS-EVENT-0316	Preparing to create custom engine version *name*. The	None

Category	Amazon RDS event ID	Message	Notes
		entire creation process may take up to four hours to complete.	
creation	RDS-EVENT-0317	Creating custom engine version *name*.	None
creation	RDS-EVENT-0318	Validating custom engine version *name*.	None
creation	RDS-EVENT-0319	Custom engine version *name* has been created successfully.	None
creation	RDS-EVENT-0320	RDS can't create custom engine version *name* because of an internal issue. We are addressin g the problem and will contact you if necessary . For further assistance, contact AWS Premium Support/.	None
failure	RDS-EVENT-0198	Creation failed for custom engine version *name*. *message*	The *message* includes details about the failure, such as missing files.
failure	RDS-EVENT-0277	Failure during deletion of custom engine version *name*. *message*	The *message* includes details about the failure.

Category	Amazon RDS event ID	Message	Notes
restoring	RDS-EVENT-0352	The maximum database count supported for point-in-time restore has changed.	The *message* includes details about the event.

Monitoring Amazon RDS log files

Every RDS database engine generates logs that you can access for auditing and troubleshooting. The type of logs depends on your database engine.

You can access database logs for DB instances using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the Amazon RDS API. You can't view, watch, or download transaction logs.

Topics

- Viewing and listing database log files
- Downloading a database log file
- Watching a database log file
- Publishing database logs to Amazon CloudWatch Logs
- Reading log file contents using REST
- Amazon RDS for Db2 database log files
- MariaDB database log files
- Amazon RDS for Microsoft SQL Server database log files
- MySQL database log files
- Amazon RDS for Oracle database log files
- RDS for PostgreSQL database log files

Viewing and listing database log files

You can view database log files for your Amazon RDS DB engine by using the AWS Management Console. You can list what log files are available for download or monitoring by using the AWS CLI or Amazon RDS API.

> **Note**
>
> If you can't view the list of log files for an existing RDS for Oracle DB instance, reboot the instance to view the list.

Console

To view a database log file

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.
5. Scroll down to the **Logs** section.
6. (Optional) Enter a search term to filter your results.
7. Choose the log that you want to view, and then choose **View**.

AWS CLI

To list the available database log files for a DB instance, use the AWS CLI `describe-db-log-files` command.

The following example returns a list of log files for a DB instance named `my-db-instance`.

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

To list the available database log files for a DB instance, use the Amazon RDS API `DescribeDBLogFiles` action.

Downloading a database log file

You can use the AWS Management Console, AWS CLI, or API to download a database log file.

Console

To download a database log file

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.

4. Choose the **Logs & events** tab.
5. Scroll down to the **Logs** section.
6. In the **Logs** section, choose the button next to the log that you want to download, and then choose **Download**.
7. Open the context (right-click) menu for the link provided, and then choose **Save Link As**. Enter the location where you want the log file to be saved, and then choose **Save**.



AWS CLI

To download a database log file, use the AWS CLI command `download-db-log-file-portion`. By default, this command downloads only the latest portion of a log file. However, you can download an entire file by specifying the parameter `--starting-token 0`.

The following example shows how to download the entire contents of a log file called *log/ERROR.4* and store it in a local file called *errorlog.txt*.

Example

For Linux, macOS, or Unix:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

For Windows:

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

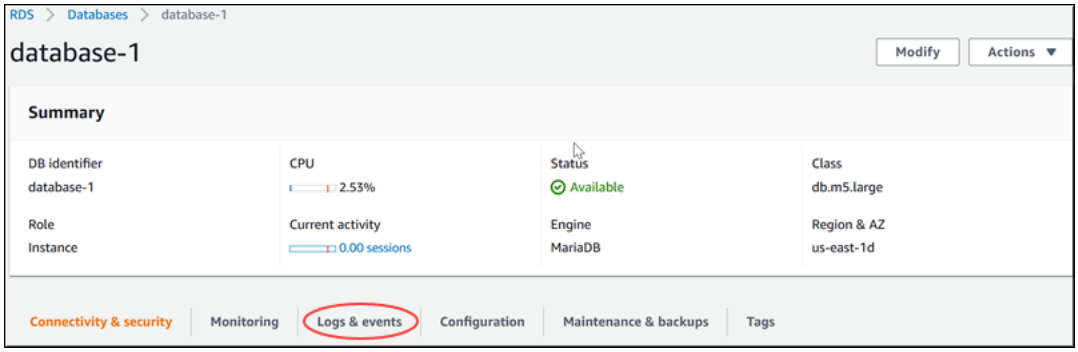
To download a database log file, use the Amazon RDS API `DownloadDBLogFilePortion` action.

Watching a database log file

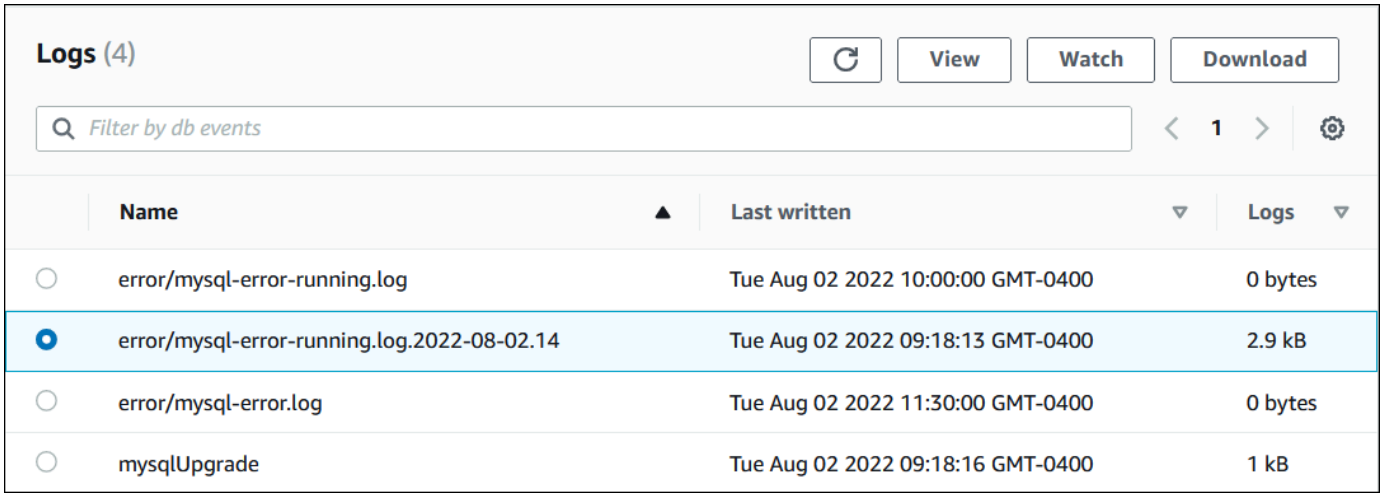
Watching a database log file is equivalent to tailing the file on a UNIX or Linux system. You can watch a log file by using the AWS Management Console. RDS refreshes the tail of the log every 5 seconds.

To watch a database log file

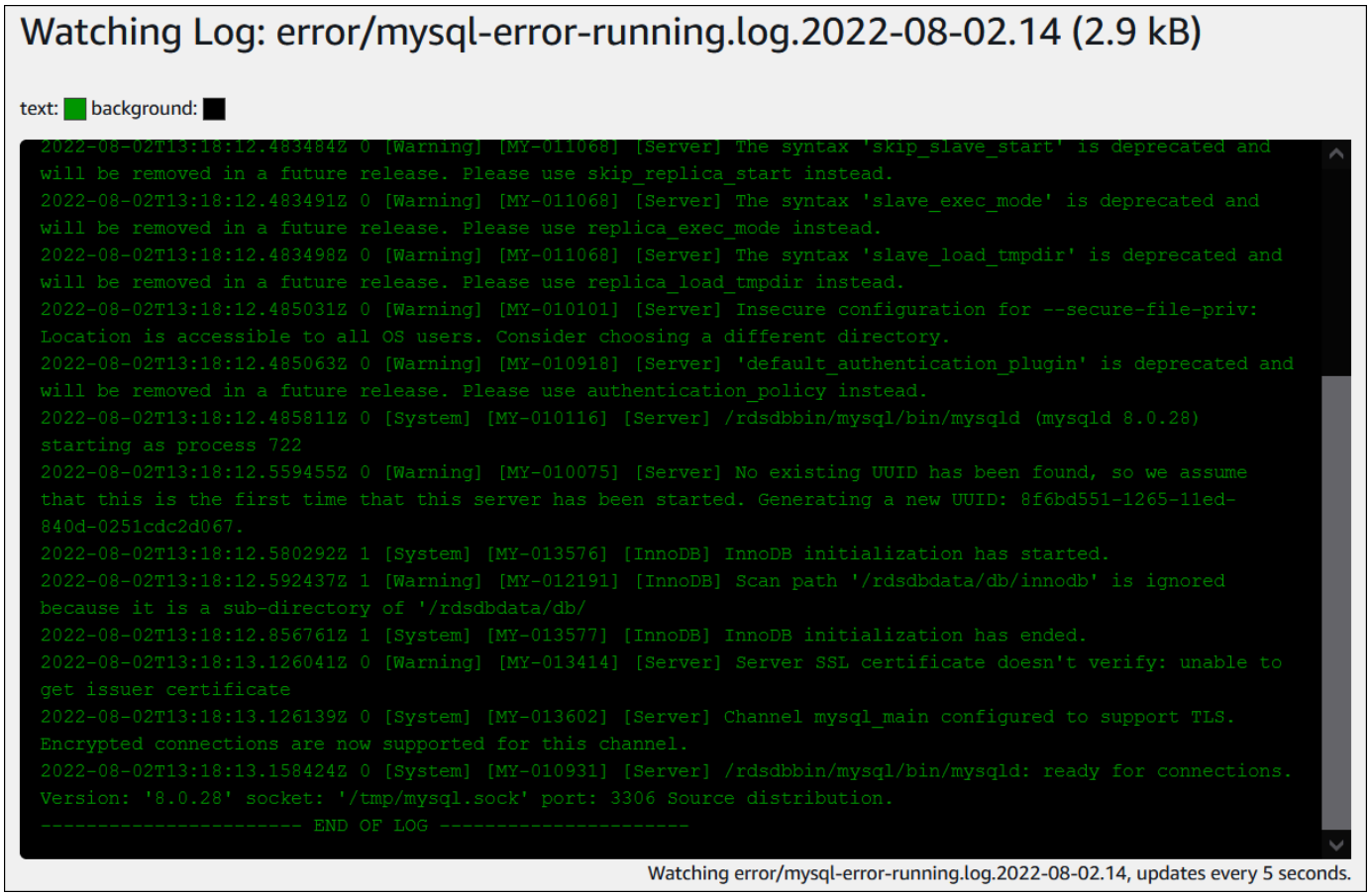
1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.



5. In the **Logs** section, choose a log file, and then choose **Watch**.



RDS shows the tail of the log, as in the following MySQL example.



Publishing database logs to Amazon CloudWatch Logs

In an on-premises database, the database logs reside on the file system. Amazon RDS doesn't provide host access to the database logs on the file system of your DB instance. For this reason,

Amazon RDS lets you export database logs to Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data. You can also store the data in highly durable storage and manage the data with the CloudWatch Logs Agent.

Topics

- Overview of RDS integration with CloudWatch Logs
- Deciding which logs to publish to CloudWatch Logs
- Specifying the logs to publish to CloudWatch Logs
- Searching and filtering your logs in CloudWatch Logs

Overview of RDS integration with CloudWatch Logs

In CloudWatch Logs, a *log stream* is a sequence of log events that share the same source. Each separate source of logs in CloudWatch Logs makes up a separate log stream. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

Amazon RDS continuously streams your DB instance log records to a log group. For example, you have a log group `/aws/rds/instance/`*`instance_name`*`/`*`log_type`* for each type of log that you publish. This log group is in the same AWS Region as the database instance that generates the log.

AWS retains log data published to CloudWatch Logs for an indefinite time period unless you specify a retention period. For more information, see Change log data retention in CloudWatch Logs.

Deciding which logs to publish to CloudWatch Logs

Each RDS database engine supports its own set of logs. To learn about the options for your database engine, review the following topics:

- the section called “Publishing Db2 logs to Amazon CloudWatch Logs”
- the section called “Publishing MariaDB logs to Amazon CloudWatch Logs”
- the section called “Publishing MySQL logs to Amazon CloudWatch Logs”
- the section called “Publishing Oracle logs to Amazon CloudWatch Logs”
- the section called “Publishing PostgreSQL logs to Amazon CloudWatch Logs”
- the section called “Publishing SQL Server logs to Amazon CloudWatch Logs”

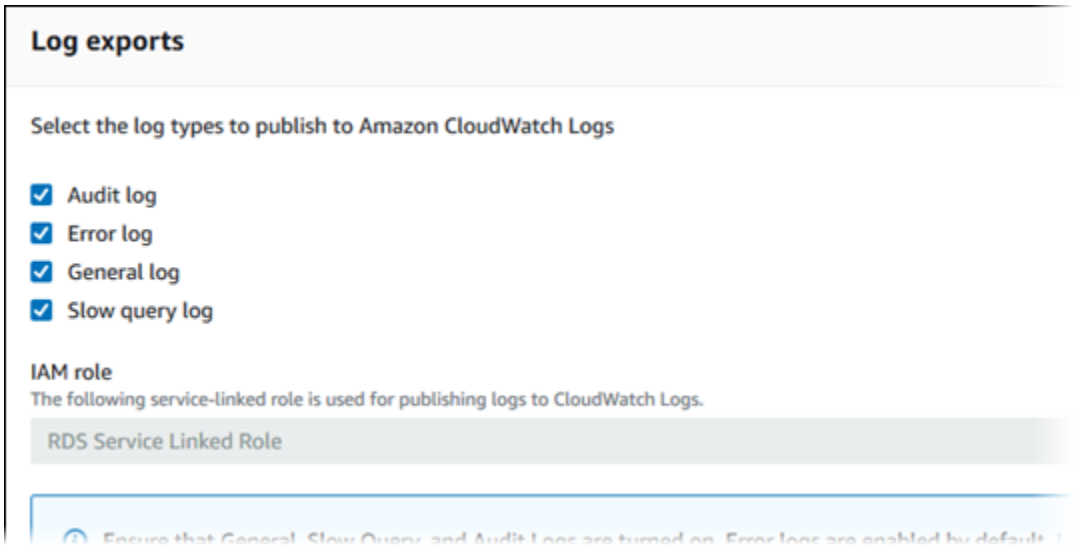
Specifying the logs to publish to CloudWatch Logs

You specify which logs to publish in the console. Make sure that you have a service-linked role in AWS Identity and Access Management (IAM). For more information about service-linked roles, see Using service-linked roles for Amazon RDS.

To specify the logs to publish

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Do either of the following:
   - Choose **Create database**.
   - Choose a database from the list, and then choose **Modify**.
4. In **Logs exports**, choose which logs to publish.

   The following example specifies the audit log, error logs, general log, and slow query log for an RDS for MySQL DB instance.



Searching and filtering your logs in CloudWatch Logs

You can search for log entries that meet a specified criteria using the CloudWatch Logs console. You can access the logs either through the RDS console, which leads you to the CloudWatch Logs console, or from the CloudWatch Logs console directly.

To search your RDS logs using the RDS console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose a DB instance.
4. Choose **Configuration**.
5. Under **Published logs**, choose the database log that you want to view.

To search your RDS logs using the CloudWatch Logs console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Log groups**.
3. In the filter box, enter **`/aws/rds`**.
4. For **Log Groups**, choose the name of the log group containing the log stream to search.
5. For **Log Streams**, choose the name of the log stream to search.
6. Under **Log events**, enter the filter syntax to use.

For more information, see Searching and filtering log data in the *Amazon CloudWatch Logs User Guide*. For a blog tutorial explaining how to monitor RDS logs, see Build proactive database monitoring for Amazon RDS with Amazon CloudWatch Logs, AWS Lambda, and Amazon SNS.

Reading log file contents using REST

Amazon RDS provides a REST endpoint that allows access to DB instance log files. This is useful if you need to write an application to stream Amazon RDS log file contents.

The syntax is:

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
Content-type: application/json
host: rds.region.amazonaws.com
```

The following parameters are required:

- *`DBInstanceIdentifier`*—the name of the DB instance that contains the log file you want to download.

- *LogFileName*—the name of the log file to be downloaded.

The response contains the contents of the requested log file, as a stream.

The following example downloads the log file named *log/ERROR.6* for the DB instance named *sample-sql* in the *us-west-2* region.

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

If you specify a nonexistent DB instance, the response consists of the following error:

- DBInstanceNotFound—*DBInstanceIdentifier* does not refer to an existing DB instance. (HTTP status code: 404)

Amazon RDS for Db2 database log files

You can access RDS for Db2 diagnostic logs and notify logs by using the Amazon RDS console, AWS CLI, or RDS API. For more information about viewing, downloading, and watching file-based database logs, see Monitoring Amazon RDS log files.

Topics

- Retention schedule
- Publishing Db2 logs to Amazon CloudWatch Logs

Retention schedule

Log files are rotated each day and whenever your DB instance is restarted. The following is the retention schedule for RDS for Db2 logs on Amazon RDS.

Log type	Retention schedule
Diagnostic logs	Db2 deletes logs outside of the retention settings in the instance-level configuration. Amazon RDS sets the `diagsize` parameter to 1000.
Notify logs	Db2 deletes logs outside of the retention settings in the instance-level configuration. Amazon RDS sets the `diagsize` parameter to 1000.

Publishing Db2 logs to Amazon CloudWatch Logs

With RDS for Db2, you can publish diagnostic and notify log events directly to Amazon CloudWatch Logs. Analyze the log data with CloudWatch Logs, then use CloudWatch to create alarms and view metrics.

With CloudWatch Logs, you can do the following:

- Store logs in highly durable storage space with a retention period that you define.
- Search and filter log data.
- Share log data between accounts.
- Export logs to Amazon S3.
- Stream data to Amazon OpenSearch Service.

- Process log data in real time with Amazon Kinesis Data Streams. For more information, see Working with Amazon CloudWatch Logs in the *Amazon Managed Service for Apache Flink for SQL Applications Developer Guide*.

Amazon RDS publishes each RDS for Db2 database log as a separate database stream in the log group. For example, if you publish the diagnostic logs and notify logs, diagnostic data is stored in a diagnostic log stream in the `/aws/rds/instance/my_instance/diagnostic` log group, and notify log data is stored in the `/aws/rds/instance/my_instance/notify` log group.

> **Note**
>
> Publishing RDS for Db2 logs to CloudWatch Logs isn't enabled by default. Publishing self-tuning memory manager (STMM) and optimizer statistics logs isn't supported. Publishing RDS for Db2 logs to CloudWatch Logs is supported in all Regions, except for Asia Pacific (Hong Kong).

Console

To publish RDS for Db2 logs to CloudWatch Logs from the AWS Management Console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**.
4. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.

   You can choose **diag.log**, **notify.log**, or both.
5. Choose **Continue**, and then choose **Modify DB Instance** on the summary page.

AWS CLI

To publish RDS for Db2 logs, you can use the `modify-db-instance` command with the following parameters:

- `--db-instance-identifier`

- `--cloudwatch-logs-export-configuration`

> **Note**
>
> A change to the `--cloudwatch-logs-export-configuration` option is always applied to the DB instance immediately. Therefore, the `--apply-immediately` and `--no-apply-immediately` options have no effect.

You can also publish RDS for Db2 logs using the following commands:

- `create-db-instance`
- `restore-db-instance-from-db-snapshot`
- `restore-db-instance-to-point-in-time`

Example

The following example creates an RDS for Db2 DB instance with CloudWatch Logs publishing enabled. The `--enable-cloudwatch-logs-exports` value is a JSON array of strings that can include `diag.log`, `notify.log`, or both.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --enable-cloudwatch-logs-exports '["diag.log","notify.log"]' \
    --db-instance-class db.m4.large \
    --engine db2-se
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --enable-cloudwatch-logs-exports "[\"diag.log\",\"notify.log\"]" ^
    --db-instance-class db.m4.large ^
    --engine db2-se
```

> **Note**
>
> When using the Windows command prompt, you must escape double quotes (") in JSON code by prefixing them with a backslash (\).

Example

The following example modifies an existing RDS for Db2 DB instance to publish log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `EnableLogTypes`, and its value is an array of strings that can include `diag.log`, `notify.log`, or both.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["diag.log","notify.log"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"EnableLogTypes\":[\"diag.log\",
\"notify.log\"]}"
```

> **Note**
>
> When using the Windows command prompt, you must escape double quotes (") in JSON code by prefixing them with a backslash (\).

Example

The following example modifies an existing RDS for Db2 DB instance to disable publishing diagnostic log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `DisableLogTypes`, and its value is an array of strings that can include `diag.log`, `notify.log`, or both.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":["diag.log"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"diag.log\"]}"
```

> **Note**
>
> When using the Windows command prompt, you must escape double quotes (") in JSON code by prefixing them with a backslash (\).

MariaDB database log files

You can monitor the MariaDB error log, slow query log, the IAM database authentication error log, and the general log. The MariaDB error log is generated by default; you can generate the slow query and general logs by setting parameters in your DB parameter group. Amazon RDS rotates all of the MariaDB log files; the intervals for each type are given following.

You can monitor the MariaDB logs directly through the Amazon RDS console, Amazon RDS API, Amazon RDS CLI, or AWS SDKs. You can also access MariaDB logs by directing the logs to a database table in the main database and querying that table. You can use the mysqlbinlog utility to download a binary log.

For more information about viewing, downloading, and watching file-based database logs, see Monitoring Amazon RDS log files.

Topics

- Accessing MariaDB error logs
- Accessing the MariaDB slow query and general logs
- Publishing MariaDB logs to Amazon CloudWatch Logs
- Log rotation and retention for MariaDB
- Managing table-based MariaDB logs
- Configuring MariaDB binary logging
- Accessing MariaDB binary logs
- Enabling MariaDB binary log annotation

Accessing MariaDB error logs

The MariaDB error log is written to the `<host-name>.err` file. You can view this file by using the Amazon RDS console, You can also retrieve the log using the Amazon RDS API, Amazon RDS CLI, or AWS SDKs. The `<host-name>.err` file is flushed every 5 minutes, and its contents are appended to `mysql-error-running.log`. The `mysql-error-running.log` file is then rotated every hour and the hourly files generated during the last 24 hours are retained. Each log file has the hour it was generated (in UTC) appended to its name. The log files also have a timestamp that helps you determine when the log entries were written.

MariaDB writes to the error log only on startup, shutdown, and when it encounters errors. A DB instance can go hours or days without new entries being written to the error log. If you see no recent entries, it's because the server did not encounter an error that resulted in a log entry.

Accessing the MariaDB slow query and general logs

You can write the MariaDB slow query log and general log to a file or database table by setting parameters in your DB parameter group. For information about creating and modifying a DB parameter group, see Parameter groups for Amazon RDS. You must set these parameters before you can view the slow query log or general log in the Amazon RDS console or by using the Amazon RDS API, AWS CLI, or AWS SDKs.

You can control MariaDB logging by using the parameters in this list:

- `slow_query_log` or `log_slow_query`: To create the slow query log, set to 1. The default is 0.
- `general_log`: To create the general log, set to 1. The default is 0.
- `long_query_time` or `log_slow_query_time`: To prevent fast-running queries from being logged in the slow query log, specify a value for the shortest query run time to be logged, in seconds. The default is 10 seconds; the minimum is 0. If log_output = FILE, you can specify a floating point value that goes to microsecond resolution. If log_output = TABLE, you must specify an integer value with second resolution. Only queries whose run time exceeds the `long_query_time` or `log_slow_query_time` value are logged. For example, setting `long_query_time` or `log_slow_query_time` to 0.1 prevents any query that runs for less than 100 milliseconds from being logged.
- `log_queries_not_using_indexes`: To log all queries that do not use an index to the slow query log, set this parameter to 1. The default is 0. Queries that do not use an index are logged even if their run time is less than the value of the `long_query_time` parameter.
- `log_output` *`option`*: You can specify one of the following options for the `log_output` parameter:
  - **TABLE** (default)– Write general queries to the `mysql.general_log` table, and slow queries to the `mysql.slow_log` table.
  - **FILE**– Write both general and slow query logs to the file system. Log files are rotated hourly.
  - **NONE**– Disable logging.

When logging is enabled, Amazon RDS rotates table logs or deletes log files at regular intervals. This measure is a precaution to reduce the possibility of a large log file either blocking database use or affecting performance. FILE and TABLE logging approach rotation and deletion as follows:

- When FILE logging is enabled, log files are examined every hour and log files older than 24 hours are deleted. In some cases, the remaining combined log file size after the deletion might exceed the threshold of 2 percent of a DB instance's allocated space. In these cases, the largest log files are deleted until the log file size no longer exceeds the threshold.
- When TABLE logging is enabled, in some cases log tables are rotated every 24 hours. This rotation occurs if the space used by the table logs is more than 20 percent of the allocated storage space. It also occurs if the size of all logs combined is greater than 10 GB. If the amount of space used for a DB instance is greater than 90 percent of the DB instance's allocated storage space, the thresholds for log rotation are reduced. Log tables are then rotated if the space used by the table logs is more than 10 percent of the allocated storage space. They're also rotated if the size of all logs combined is greater than 5 GB.

  When log tables are rotated, the current log table is copied to a backup log table and the entries in the current log table are removed. If the backup log table already exists, then it is deleted before the current log table is copied to the backup. You can query the backup log table if needed. The backup log table for the `mysql.general_log` table is named `mysql.general_log_backup`. The backup log table for the `mysql.slow_log` table is named `mysql.slow_log_backup`.

  You can rotate the `mysql.general_log` table by calling the `mysql.rds_rotate_general_log` procedure. You can rotate the `mysql.slow_log` table by calling the `mysql.rds_rotate_slow_log` procedure.

  Table logs are rotated during a database version upgrade.

Amazon RDS records both TABLE and FILE log rotation in an Amazon RDS event and sends you a notification.

To work with the logs from the Amazon RDS console, Amazon RDS API, Amazon RDS CLI, or AWS SDKs, set the `log_output` parameter to FILE. Like the MariaDB error log, these log files are rotated hourly. The log files that were generated during the previous 24 hours are retained.

For more information about the slow query and general logs, go to the following topics in the MariaDB documentation:

- Slow query log
- General query log

Publishing MariaDB logs to Amazon CloudWatch Logs

You can configure your MariaDB DB instance to publish log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage.

Amazon RDS publishes each MariaDB database log as a separate database stream in the log group. For example, suppose that you configure the export function to include the slow query log. Then slow query data is stored in a slow query log stream in the `/aws/rds/instance/my_instance/slowquery` log group.

The error log is enabled by default. The following table summarizes the requirements for the other MariaDB logs.

Log	Requirement
Audit log	The DB instance must use a custom option group with the `MARIADB_AUDIT_PLUGIN` option.
General log	The DB instance must use a custom parameter group with the parameter setting `general_log = 1` to enable the general log.
Slow query log	The DB instance must use a custom parameter group with the parameter setting `slow_query_log = 1` or `log_slow_query = 1` to enable the slow query log.
IAM database authentication error log	You must enable the log type `iam-db-auth-error` for a DB instance by creating or modifying a DB instance.

Log	Requirement
Log output	The DB instance must use a custom parameter group with the parameter setting `log_output = FILE` to write logs to the file system and publish them to CloudWatch Logs.

Console

To publish MariaDB logs to CloudWatch Logs from the console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**.
4. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.
5. Choose **Continue**, and then choose **Modify DB Instance** on the summary page.

AWS CLI

You can publish a MariaDB logs with the AWS CLI. You can call the `modify-db-instance` command with the following parameters:

- `--db-instance-identifier`
- `--cloudwatch-logs-export-configuration`

> **Note**
>
> A change to the `--cloudwatch-logs-export-configuration` option is always applied to the DB instance immediately. Therefore, the `--apply-immediately` and `--no-apply-immediately` options have no effect.

You can also publish MariaDB logs by calling the following AWS CLI commands:

- `create-db-instance`
- `restore-db-instance-from-db-snapshot`
- `restore-db-instance-from-s3`
- `restore-db-instance-to-point-in-time`

Run one of these AWS CLI commands with the following options:

- `--db-instance-identifier`
- `--enable-cloudwatch-logs-exports`
- `--db-instance-class`
- `--engine`

Other options might be required depending on the AWS CLI command you run.

Example

The following example modifies an existing MariaDB DB instance to publish log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `EnableLogTypes`, and its value is an array of strings with any combination of `audit`, `error`, `general`, and `slowquery`.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["audit","error","general","slowquery"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["audit","error","general","slowquery"]}'
```

Example

The following command creates a MariaDB DB instance and publishes log files to CloudWatch Logs. The `--enable-cloudwatch-logs-exports` value is a JSON array of strings. The strings can be any combination of `audit`, `error`, `general`, and `slowquery`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --enable-cloudwatch-logs-exports '["audit","error","general","slowquery"]' \
    --db-instance-class db.m4.large \
    --engine mariadb
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --enable-cloudwatch-logs-exports '["audit","error","general","slowquery"]' ^
    --db-instance-class db.m4.large ^
    --engine mariadb
```

RDS API

You can publish MariaDB logs with the RDS API. You can call the `ModifyDBInstance` operation with the following parameters:

- `DBInstanceIdentifier`
- `CloudwatchLogsExportConfiguration`

> **Note**
>
> A change to the `CloudwatchLogsExportConfiguration` parameter is always applied to the DB instance immediately. Therefore, the `ApplyImmediately` parameter has no effect.

You can also publish MariaDB logs by calling the following RDS API operations:

- `CreateDBInstance`
- `RestoreDBInstanceFromDBSnapshot`
- `RestoreDBInstanceFromS3`
- `RestoreDBInstanceToPointInTime`

Run one of these RDS API operations with the following parameters:

- `DBInstanceIdentifier`
- `EnableCloudwatchLogsExports`
- `Engine`
- `DBInstanceClass`

Other parameters might be required depending on the AWS CLI command you run.

Log rotation and retention for MariaDB

When logging is enabled, Amazon RDS rotates table logs or deletes log files at regular intervals. This measure is a precaution to reduce the possibility of a large log file either blocking database use or affecting performance.

The MariaDB slow query log, error log, and the general log file sizes are constrained to no more than 2 percent of the allocated storage space for a DB instance. To maintain this threshold, logs are automatically rotated every hour and log files older than 24 hours are removed. If the combined log file size exceeds the threshold after removing old log files, then the largest log files are deleted until the log file size no longer exceeds the threshold.

Amazon RDS rotates IAM database authentication error log files larger than 10 MB. Amazon RDS removes IAM database authentication error log files that are older than five days or larger than 100 MB.

Managing table-based MariaDB logs

You can direct the general and slow query logs to tables on the DB instance. To do so, create a DB parameter group and set the `log_output` server parameter to `TABLE`. General queries are then logged to the `mysql.general_log` table, and slow queries are logged to the `mysql.slow_log` table. You can query the tables to access the log information. Enabling this logging increases the amount of data written to the database, which can degrade performance.

Both the general log and the slow query logs are disabled by default. To enable logging to tables, you must also set the following server parameters to 1:

- `general_log`
- `slow_query_log` or `log_slow_query`

Log tables keep growing until the respective logging activities are turned off by resetting the appropriate parameter to `0`. A large amount of data often accumulates over time, which can use up a considerable percentage of your allocated storage space. Amazon RDS doesn't allow you to truncate the log tables, but you can move their contents. Rotating a table saves its contents to a backup table and then creates a new empty log table. You can manually rotate the log tables with the following command line procedures, where the command prompt is indicated by `PROMPT>`:

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

To completely remove the old data and reclaim the disk space, call the appropriate procedure twice in succession.

Configuring MariaDB binary logging

The *binary log* is a set of log files that contain information about data modifications made to a MariaDB server instance. The binary log contains information such as the following:

- Events that describe database changes such as table creation or row modifications
- Information about the duration of each statement that updated data
- Events for statements that could have updated data but didn't

The binary log records statements that are sent during replication. It is also required for some recovery operations. For more information, see Binary Log in the MariaDB documentation.

The automated backups feature determines whether binary logging is turned on or off for MariaDB. You have the following options:

Turn binary logging on

Set the backup retention period to a positive nonzero value.

Turn binary logging off

Set the backup retention period to zero.

For more information, see Enabling automated backups.

MariaDB on Amazon RDS supports the *row-based*, *statement-based*, and *mixed* binary logging formats. The default binary logging format is *mixed*. For details on the different MariaDB binary log formats, see Binary Log Formats in the MariaDB documentation.

If you plan to use replication, the binary logging format is important. This is because it determines the record of data changes that is recorded in the source and sent to the replication targets. For information about the advantages and disadvantages of different binary logging formats for replication, see Advantages and Disadvantages of Statement-Based and Row-Based Replication in the MySQL documentation.

Important

Setting the binary logging format to row-based can result in very large binary log files. Large binary log files reduce the amount of storage available for a DB instance. They also can increase the amount of time to perform a restore operation of a DB instance.
Statement-based replication can cause inconsistencies between the source DB instance and a read replica. For more information, see Unsafe Statements for Statement-based Replication in the MariaDB documentation.
Enabling binary logging increases the number of write disk I/O operations to the DB instance. You can monitor IOPS usage with the `WriteIOPS` CloudWatch metric.

To set the MariaDB binary logging format

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose the parameter group that is used by the DB instance that you want to modify.

   You can't modify a default parameter group. If the DB instance is using a default parameter group, create a new parameter group and associate it with the DB instance.

   For more information on DB parameter groups, see Parameter groups for Amazon RDS.

4. For **Parameter group actions**, choose **Edit**.
5. Set the `binlog_format` parameter to the binary logging format of your choice (**ROW**, **STATEMENT**, or **MIXED**).

   You can turn off binary logging by setting the backup retention period of a DB instance to zero, but this disables daily automated backups. Disabling automated backups turns off or disables the `log_bin` session variable. This disables binary logging on the RDS for MariaDB DB instance, which in turn resets the `binlog_format` session variable to the default value of `ROW` in the database. We recommend that you don't disable backups. For more information about the **Backup retention period** setting, see Settings for DB instances.
6. Choose **Save changes** to save the updates to the DB parameter group.

Because the `binlog_format` parameter is dynamic in RDS for MariaDB, you don't need to reboot the DB instance for the changes to apply.

> **Important**
>
> Changing a DB parameter group affects all DB instances that use that parameter group. If you want to specify different binary logging formats for different MariaDB DB instances in an AWS Region, the DB instances must use different DB parameter groups. These parameter groups identify different logging formats. Assign the appropriate DB parameter group to the each DB instance.

Accessing MariaDB binary logs

You can use the mysqlbinlog utility to download binary logs in text format from MariaDB DB instances. The binary log is downloaded to your local computer. For more information about using the mysqlbinlog utility, go to Using mysqlbinlog in the MariaDB documentation.

To run the mysqlbinlog utility against an Amazon RDS instance, use the following options:

- Specify the `--read-from-remote-server` option.
- `--host`: Specify the DNS name from the endpoint of the instance.
- `--port`: Specify the port used by the instance.
- `--user`: Specify a MariaDB user that has been granted the replication slave permission.

- `--password`: Specify the password for the user, or omit a password value so the utility prompts you for a password.
- `--result-file`: Specify the local file that receives the output.
- Specify the names of one or more binary log files. To get a list of the available logs, use the SQL command SHOW BINARY LOGS.

For more information about mysqlbinlog options, go to mysqlbinlog options in the MariaDB documentation.

The following is an example:

For Linux, macOS, or Unix:

```
mysqlbinlog \
    --read-from-remote-server \
    --host=mariadbinstance1.1234abcd.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password <password> \
    --result-file=/tmp/binlog.txt
```

For Windows:

```
mysqlbinlog ^
    --read-from-remote-server ^
    --host=mariadbinstance1.1234abcd.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password <password> ^
    --result-file=/tmp/binlog.txt
```

Amazon RDS normally purges a binary log as soon as possible. However, the binary log must still be available on the instance to be accessed by mysqlbinlog. To specify the number of hours for RDS to retain binary logs, use the `mysql.rds_set_configuration` stored procedure. Specify a period with enough time for you to download the logs. After you set the retention period, monitor storage usage for the DB instance to ensure that the retained binary logs don't take up too much storage.

The following example sets the retention period to 1 day.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

To display the current setting, use the `mysql.rds_show_configuration` stored procedure.

```
call mysql.rds_show_configuration;
```

Enabling MariaDB binary log annotation

In a MariaDB DB instance, you can use the `Annotate_rows` event to annotate a row event with a copy of the SQL query that caused the row event. This approach provides similar functionality to enabling the `binlog_rows_query_log_events` parameter on an RDS for MySQL DB instance.

You can enable binary log annotations globally by creating a custom parameter group and setting the `binlog_annotate_row_events` parameter to **1**. You can also enable annotations at the session level, by calling `SET SESSION binlog_annotate_row_events = 1`. Use the `replicate_annotate_row_events` to replicate binary log annotations to the replica instance if binary logging is enabled on it. No special privileges are required to use these settings.

The following is an example of a row-based transaction in MariaDB. The use of row-based logging is triggered by setting the transaction isolation level to read-committed.

```
CREATE DATABASE IF NOT EXISTS test;
USE test;
CREATE TABLE square(x INT PRIMARY KEY, y INT NOT NULL) ENGINE = InnoDB;
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
INSERT INTO square(x, y) VALUES(5, 5 * 5);
COMMIT;
```

Without annotations, the binary log entries for the transaction look like the following:

```
BEGIN
/*!*/;
# at 1163
# at 1209
#150922  7:55:57 server id 1855786460  end_log_pos 1209         Table_map:
 `test`.`square` mapped to number 76
#150922  7:55:57 server id 1855786460  end_log_pos 1247         Write_rows: table id 76
 flags: STMT_END_F
### INSERT INTO `test`.`square`
```

```
### SET
###   @1=5
###   @2=25
# at 1247
#150922  7:56:01 server id 1855786460  end_log_pos 1274         Xid = 62
COMMIT/*!*/;
```

The following statement enables session-level annotations for this same transaction, and disables them after committing the transaction:

```
CREATE DATABASE IF NOT EXISTS test;
USE test;
CREATE TABLE square(x INT PRIMARY KEY, y INT NOT NULL) ENGINE = InnoDB;
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET SESSION binlog_annotate_row_events = 1;
BEGIN;
INSERT INTO square(x, y) VALUES(5, 5 * 5);
COMMIT;
SET SESSION binlog_annotate_row_events = 0;
```

With annotations, the binary log entries for the transaction look like the following:

```
BEGIN
/*!*/;
# at 423
# at 483
# at 529
#150922  8:04:24 server id 1855786460  end_log_pos 483  Annotate_rows:
#Q> INSERT INTO square(x, y) VALUES(5, 5 * 5)
#150922  8:04:24 server id 1855786460  end_log_pos 529  Table_map: `test`.`square`
 mapped to number 76
#150922  8:04:24 server id 1855786460  end_log_pos 567  Write_rows: table id 76 flags:
 STMT_END_F
### INSERT INTO `test`.`square`
### SET
###   @1=5
###   @2=25
# at 567
#150922  8:04:26 server id 1855786460  end_log_pos 594  Xid = 88
COMMIT/*!*/;
```

Amazon RDS for Microsoft SQL Server database log files

You can access Microsoft SQL Server error logs, agent logs, trace files, and dump files by using the Amazon RDS console, AWS CLI, or RDS API. For more information about viewing, downloading, and watching file-based database logs, see Monitoring Amazon RDS log files.

Retention schedule

Log files are rotated each day and whenever your DB instance is restarted. The following is the retention schedule for Microsoft SQL Server logs on Amazon RDS.

Log type	Retention schedule
Error logs	A maximum of 30 error logs are retained. Amazon RDS might delete error logs older than 7 days.
Agent logs	A maximum of 10 agent logs are retained. Amazon RDS might delete agent logs older than 7 days.
Trace files	Trace files are retained according to the trace file retention period of your DB instance. The default trace file retention period is 7 days. To modify the trace file retention period for your DB instance, see Setting the retention period for trace and dump files.
Dump files	Dump files are retained according to the dump file retention period of your DB instance. The default dump file retention period is 7 days. To modify the dump file retention period for your DB instance, see Setting the retention period for trace and dump files.

Viewing the SQL Server error log by using the rds_read_error_log procedure

You can use the Amazon RDS stored procedure `rds_read_error_log` to view error logs and agent logs. For more information, see Viewing error and agent logs.

Publishing SQL Server logs to Amazon CloudWatch Logs

With Amazon RDS for SQL Server, you can publish error and agent log events directly to Amazon CloudWatch Logs. Analyze the log data with CloudWatch Logs, then use CloudWatch to create alarms and view metrics.

With CloudWatch Logs, you can do the following:

- Store logs in highly durable storage space with a retention period that you define.
- Search and filter log data.
- Share log data between accounts.
- Export logs to Amazon S3.
- Stream data to Amazon OpenSearch Service.
- Process log data in real time with Amazon Kinesis Data Streams. For more information, see Working with Amazon CloudWatch Logs in the *Amazon Managed Service for Apache Flink for SQL Applications Developer Guide*.

Amazon RDS publishes each SQL Server database log as a separate database stream in the log group. For example, if you publish the agent logs and error logs, error data is stored in an error log stream in the `/aws/rds/instance/my_instance.node1/error` log group, and agent log data is stored in the `/aws/rds/instance/my_instance.node1/agent` log group.

For Multi-AZ DB instances, Amazon RDS publishes the database log as two separate streams in the log group. For example, if you publish the error logs, the error data is stored in the error log streams `/aws/rds/instance/my_instance.node1/error` and `/aws/rds/instance/my_instance.node2/error` respectively. The log streams don't change during a failover and the error log stream of each node can contain error logs from primary or secondary instance. With Multi-AZ, a log stream is automatically created for `/aws/rds/instance/my_instance/rds-events` to store event data such as DB instance failovers.

> **Note**
>
> Publishing SQL Server logs to CloudWatch Logs isn't enabled by default. Publishing trace and dump files isn't supported. Publishing SQL Server logs to CloudWatch Logs is supported in all regions.

Console

To publish SQL Server DB logs to CloudWatch Logs from the AWS Management Console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**.
4. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.

   You can choose **Agent log**, **Error log**, or both.
5. Choose **Continue**, and then choose **Modify DB Instance** on the summary page.

AWS CLI

To publish SQL Server logs, you can use the `modify-db-instance` command with the following parameters:

- `--db-instance-identifier`
- `--cloudwatch-logs-export-configuration`

> **Note**
>
> A change to the `--cloudwatch-logs-export-configuration` option is always applied to the DB instance immediately. Therefore, the `--apply-immediately` and `--no-apply-immediately` options have no effect.

You can also publish SQL Server logs using the following commands:

- `create-db-instance`
- `restore-db-instance-from-db-snapshot`
- `restore-db-instance-to-point-in-time`

Example

The following example creates an SQL Server DB instance with CloudWatch Logs publishing enabled. The `--enable-cloudwatch-logs-exports` value is a JSON array of strings that can include `error`, `agent`, or both.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --enable-cloudwatch-logs-exports '["error","agent"]' \
    --db-instance-class db.m4.large \
    --engine sqlserver-se
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --enable-cloudwatch-logs-exports "[\"error\",\"agent\"]" ^
    --db-instance-class db.m4.large ^
    --engine sqlserver-se
```

> **Note**
>
> When using the Windows command prompt, you must escape double quotes (") in JSON code by prefixing them with a backslash (\).

Example

The following example modifies an existing SQL Server DB instance to publish log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `EnableLogTypes`, and its value is an array of strings that can include `error`, `agent`, or both.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":["error","agent"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"EnableLogTypes\":[\"error\",\"agent\"]}"
```

> **Note**
>
> When using the Windows command prompt, you must escape double quotes (") in JSON code by prefixing them with a backslash (\).

Example

The following example modifies an existing SQL Server DB instance to disable publishing agent log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `DisableLogTypes`, and its value is an array of strings that can include `error`, `agent`, or both.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":["agent"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"agent\"]}"
```

> **Note**
>
> When using the Windows command prompt, you must escape double quotes (") in JSON code by prefixing them with a backslash (\).

MySQL database log files

You can monitor the MySQL logs directly through the Amazon RDS console, Amazon RDS API, AWS CLI, or AWS SDKs. You can also access MySQL logs by directing the logs to a database table in the main database and querying that table. You can use the mysqlbinlog utility to download a binary log.

For more information about viewing, downloading, and watching file-based database logs, see Monitoring Amazon RDS log files.

Topics

- Overview of RDS for MySQL database logs
- Publishing MySQL logs to Amazon CloudWatch Logs
- Sending MySQL log output to tables
- Configuring RDS for MySQL binary logging for Single-AZ databases
- Configuring MySQL binary logging for Multi-AZ DB clusters
- Accessing MySQL binary logs

Overview of RDS for MySQL database logs

You can monitor the following types of RDS for MySQL log files:

- Error log
- Slow query log
- General log
- Audit log
- Instance log
- IAM database authentication error log

The RDS for MySQL error log is generated by default. You can generate the slow query and general logs by setting parameters in your DB parameter group.

Topics

- RDS for MySQL error logs

- RDS for MySQL slow query and general logs
- MySQL audit log
- Log rotation and retention for RDS for MySQL
- Size limits on redo logs

RDS for MySQL error logs

RDS for MySQL writes errors in the `mysql-error.log` file. Each log file has the hour it was generated (in UTC) appended to its name. The log files also have a timestamp that helps you determine when the log entries were written.

RDS for MySQL writes to the error log only on startup, shutdown, and when it encounters errors. A DB instance can go hours or days without new entries being written to the error log. If you see no recent entries, it's because the server didn't encounter an error that would result in a log entry.

By design, the error logs are filtered so that only unexpected events such as errors are shown. However, the error logs also contain some additional database information, for example query progress, which isn't shown. Therefore, even without any actual errors the size of the error logs might increase because of ongoing database activities. And while you might see a certain size in bytes or kilobytes for the error logs in the AWS Management Console, they might have 0 bytes when you download them.

RDS for MySQL writes `mysql-error.log` to disk every 5 minutes. It appends the contents of the log to `mysql-error-running.log`.

RDS for MySQL rotates the `mysql-error-running.log` file every hour. It retains the logs generated during the last two weeks.

> **Note**
>
> The log retention period is different between Amazon RDS and Aurora.

RDS for MySQL slow query and general logs

You can write the RDS for MySQL slow query log and the general log to a file or a database table. To do so, set parameters in your DB parameter group. For information about creating and modifying a DB parameter group, see Parameter groups for Amazon RDS. You must set these

parameters before you can view the slow query log or general log in the Amazon RDS console or by using the Amazon RDS API, Amazon RDS CLI, or AWS SDKs.

You can control RDS for MySQL logging by using the parameters in this list:

- `slow_query_log`: To create the slow query log, set to 1. The default is 0.
- `general_log`: To create the general log, set to 1. The default is 0.
- `long_query_time`: To prevent fast-running queries from being logged in the slow query log, specify a value for the shortest query runtime to be logged, in seconds. The default is 10 seconds; the minimum is 0. If log_output = FILE, you can specify a floating point value that goes to microsecond resolution. If log_output = TABLE, you must specify an integer value with second resolution. Only queries whose runtime exceeds the `long_query_time` value are logged. For example, setting `long_query_time` to 0.1 prevents any query that runs for less than 100 milliseconds from being logged.
- `log_queries_not_using_indexes`: To log all queries that do not use an index to the slow query log, set to 1. Queries that don't use an index are logged even if their runtime is less than the value of the `long_query_time` parameter. The default is 0.
- `log_output` *`option`*: You can specify one of the following options for the `log_output` parameter.
  - **TABLE** (default) – Write general queries to the `mysql.general_log` table, and slow queries to the `mysql.slow_log` table.
  - **FILE** – Write both general and slow query logs to the file system.
  - **NONE** – Disable logging.

For slow query data to appear in Amazon CloudWatch Logs, the following conditions must be met:

- CloudWatch Logs must be configured to include slow query logs.
- `slow_query_log` must be enabled.
- `log_output` must be set to `FILE`.
- The query must take longer than the time configured for `long_query_time`.

For more information about the slow query and general logs, go to the following topics in the MySQL documentation:

- The slow query log

- The general query log

MySQL audit log

To access the audit log, the DB instance must use a custom option group with the `MARIADB_AUDIT_PLUGIN` option. For more information, see MariaDB Audit Plugin support for MySQL.

Log rotation and retention for RDS for MySQL

When logging is enabled, Amazon RDS rotates table logs or deletes log files at regular intervals. This measure is a precaution to reduce the possibility of a large log file either blocking database use or affecting performance. RDS for MySQL handles rotation and deletion as follows:

- The MySQL slow query log, error log, and the general log file sizes are constrained to no more than 2 percent of the allocated storage space for a DB instance. To maintain this threshold, logs are automatically rotated every hour. MySQL removes log files more than two weeks old. If the combined log file size exceeds the threshold after removing old log files, then the oldest log files are deleted until the log file size no longer exceeds the threshold.
- When `FILE` logging is enabled, log files are examined every hour and log files more than two weeks old are deleted. In some cases, the remaining combined log file size after the deletion might exceed the threshold of 2 percent of a DB instance's allocated space. In these cases, the oldest log files are deleted until the log file size no longer exceeds the threshold.
- When `TABLE` logging is enabled, in some cases log tables are rotated every 24 hours. This rotation occurs if the space used by the table logs is more than 20 percent of the allocated storage space. It also occurs if the size of all logs combined is greater than 10 GB. If the amount of space used for a DB instance is greater than 90 percent of the DB instance's allocated storage space, then the thresholds for log rotation are reduced. Log tables are then rotated if the space used by the table logs is more than 10 percent of the allocated storage space. They're also rotated if the size of all logs combined is greater than 5 GB. You can subscribe to the `low storage` event category to be notified when log tables are rotated to free up space. For more information, see Working with Amazon RDS event notification.

  When log tables are rotated, the current log table is first copied to a backup log table. Then the entries in the current log table are removed. If the backup log table already exists, then it is deleted before the current log table is copied to the backup. You can query the backup log table if needed. The backup log table for the `mysql.general_log` table is named

`mysql.general_log_backup`. The backup log table for the `mysql.slow_log` table is named `mysql.slow_log_backup`.

You can rotate the `mysql.general_log` table by calling the `mysql.rds_rotate_general_log` procedure. You can rotate the `mysql.slow_log` table by calling the `mysql.rds_rotate_slow_log` procedure.

Table logs are rotated during a database version upgrade.

To work with the logs from the Amazon RDS console, Amazon RDS API, Amazon RDS CLI, or AWS SDKs, set the `log_output` parameter to FILE. Like the MySQL error log, these log files are rotated hourly. The log files that were generated during the previous two weeks are retained. Note that the retention period is different between Amazon RDS and Aurora.

Size limits on redo logs

For RDS for MySQL version 8.0.32 and lower, the default value of this parameter is 256 MB. This amount is derived by multiplying the default value of the `innodb_log_file_size` parameter (128 MB) by the default value of the `innodb_log_files_in_group` parameter (2). For more information, see Best practices for configuring parameters for Amazon RDS for MySQL, part 1: Parameters related to performance.

For RDS for MySQL version 8.0.33 and higher minor versions, Amazon RDS uses the `innodb_redo_log_capacity` parameter instead of the `innodb_log_file_size` parameter. The Amazon RDS default value of the `innodb_redo_log_capacity` parameter is 2 GB. For more information, see Changes in MySQL 8.0.30 in the MySQL documentation.

Starting with MySQL 8.4, Amazon RDS enables the `innodb_dedicated_server` parameter by default. With the `innodb_dedicated_server` parameter, the database engine calculates the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters. For more information, see Configuring buffer pool size and redo log capacity in MySQL 8.4.

Publishing MySQL logs to Amazon CloudWatch Logs

You can configure your MySQL DB instance to publish log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage.

Amazon RDS publishes each MySQL database log as a separate database stream in the log group. For example, if you configure the export function to include the slow query log, slow query data is stored in a slow query log stream in the `/aws/rds/instance/my_instance/slowquery` log group.

The error log is enabled by default. The following table summarizes the requirements for the other MySQL logs.

Log	Requirement
Audit log	The DB instance must use a custom option group with the `MARIADB_AUDIT_PLUGIN` option.
General log	The DB instance must use a custom parameter group with the parameter setting `general_log = 1` to enable the general log.
Slow query log	The DB instance must use a custom parameter group with the parameter setting `slow_query_log = 1` to enable the slow query log.
IAM database authentication error log	You must enable the log type `iam-db-auth-error` for a DB instance by creating or modifying a DB instance.
Log output	The DB instance must use a custom parameter group with the parameter setting `log_output = FILE` to write logs to the file system and publish them to CloudWatch Logs.

Console

To publish MySQL logs to CloudWatch Logs using the console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.

3. Choose **Modify**.
4. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.
5. Choose **Continue**, and then choose **Modify DB Instance** on the summary page.

AWS CLI

You can publish MySQL logs with the AWS CLI. You can call the `modify-db-instance` command with the following parameters:

- `--db-instance-identifier`
- `--cloudwatch-logs-export-configuration`

> **Note**
>
> A change to the `--cloudwatch-logs-export-configuration` option is always applied to the DB instance immediately. Therefore, the `--apply-immediately` and `--no-apply-immediately` options have no effect.

You can also publish MySQL logs by calling the following AWS CLI commands:

- `create-db-instance`
- `restore-db-instance-from-db-snapshot`
- `restore-db-instance-from-s3`
- `restore-db-instance-to-point-in-time`

Run one of these AWS CLI commands with the following options:

- `--db-instance-identifier`
- `--enable-cloudwatch-logs-exports`
- `--db-instance-class`
- `--engine`

Other options might be required depending on the AWS CLI command you run.

Example

The following example modifies an existing MySQL DB instance to publish log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `EnableLogTypes`, and its value is an array of strings with any combination of `audit`, `error`, `general`, and `slowquery`.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["audit","error","general","slowquery"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["audit","error","general","slowquery"]}'
```

Example

The following example creates a MySQL DB instance and publishes log files to CloudWatch Logs. The `--enable-cloudwatch-logs-exports` value is a JSON array of strings. The strings can be any combination of `audit`, `error`, `general`, and `slowquery`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --enable-cloudwatch-logs-exports '["audit","error","general","slowquery"]' \
    --db-instance-class db.m4.large \
    --engine MySQL
```

For Windows:

```
aws rds create-db-instance ^
```

```
    --db-instance-identifier mydbinstance ^
    --enable-cloudwatch-logs-exports '["audit","error","general","slowquery"]' ^
    --db-instance-class db.m4.large ^
    --engine MySQL
```

RDS API

You can publish MySQL logs with the RDS API. You can call the ModifyDBInstance action with the following parameters:

- `DBInstanceIdentifier`
- `CloudwatchLogsExportConfiguration`

> **Note**
>
> A change to the `CloudwatchLogsExportConfiguration` parameter is always applied to the DB instance immediately. Therefore, the `ApplyImmediately` parameter has no effect.

You can also publish MySQL logs by calling the following RDS API operations:

- CreateDBInstance
- RestoreDBInstanceFromDBSnapshot
- RestoreDBInstanceFromS3
- RestoreDBInstanceToPointInTime

Run one of these RDS API operations with the following parameters:

- `DBInstanceIdentifier`
- `EnableCloudwatchLogsExports`
- `Engine`
- `DBInstanceClass`

Other parameters might be required depending on the AWS CLI command you run.

Sending MySQL log output to tables

You can direct the general and slow query logs to tables on the DB instance by creating a DB parameter group and setting the `log_output` server parameter to `TABLE`. General queries are then logged to the `mysql.general_log` table, and slow queries are logged to the `mysql.slow_log` table. You can query the tables to access the log information. Enabling this logging increases the amount of data written to the database, which can degrade performance.

Both the general log and the slow query logs are disabled by default. In order to enable logging to tables, you must also set the `general_log` and `slow_query_log` server parameters to `1`.

Log tables keep growing until the respective logging activities are turned off by resetting the appropriate parameter to `0`. A large amount of data often accumulates over time, which can use up a considerable percentage of your allocated storage space. Amazon RDS doesn't allow you to truncate the log tables, but you can move their contents. Rotating a table saves its contents to a backup table and then creates a new empty log table. You can manually rotate the log tables with the following command line procedures, where the command prompt is indicated by `PROMPT>`:

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

To completely remove the old data and reclaim the disk space, call the appropriate procedure twice in succession.

Configuring RDS for MySQL binary logging for Single-AZ databases

The *binary log* is a set of log files that contain information about data modifications made to an MySQL server instance. The binary log contains information such as the following:

- Events that describe database changes such as table creation or row modifications
- Information about the duration of each statement that updated data
- Events for statements that could have updated data but didn't

The binary log records statements that are sent during replication. It is also required for some recovery operations. For more information, see The Binary Log in the MySQL documentation.

The automated backups feature determines whether binary logging is turned on or off for MySQL. You have the following options:

Turn binary logging on

Set the backup retention period to a positive nonzero value.

Turn binary logging off

Set the backup retention period to zero.

For more information, see Enabling automated backups.

MySQL on Amazon RDS supports the *row-based*, *statement-based*, and *mixed* binary logging formats. We recommend mixed unless you need a specific binlog format. For details on the different MySQL binary log formats, see Binary Logging Formats in the MySQL documentation.

If you plan to use replication, the binary logging format is important because it determines the record of data changes that is recorded in the source and sent to the replication targets. For information about the advantages and disadvantages of different binary logging formats for replication, see Advantages and Disadvantages of Statement-Based and Row-Based Replication in the MySQL documentation.

Important

With MySQL 8.0.34, MySQL deprecated the `binlog_format` parameter. In later MySQL versions, MySQL plans to remove the parameter and only support row-based replication. As a result, we recommend using row-based logging for new MySQL replication setups. For more information, see binlog_format in the MySQL documentation.
MySQL versions 8.0 and 8.4 accept the parameter `binlog_format`. When using this parameter, MySQL issues a deprecation warning. In a future major release, MySQL will remove the parameter `binlog_format`.
Statement-based replication can cause inconsistencies between the source DB instance and a read replica. For more information, see Determination of Safe and Unsafe Statements in Binary Logging in the MySQL documentation.
Enabling binary logging increases the number of write disk I/O operations to the DB instance. You can monitor IOPS usage with the `WriteIOPS` CloudWatch metric.

To set the MySQL binary logging format

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Parameter groups**.
3. Choose the DB parameter group, associated with the DB instance, that you want to modify.

   You can't modify a default parameter group. If the DB instance is using a default parameter group, create a new parameter group and associate it with the DB instance.

   For more information on parameter groups, see Parameter groups for Amazon RDS.
4. From **Actions**, choose **Edit**.
5. Set the `binlog_format` parameter to the binary logging format of your choice (`ROW`, `STATEMENT`, or `MIXED`).

   You can turn off binary logging by setting the backup retention period of a DB instance to zero, but this disables daily automated backups. Disabling automated backups turns off or disables the `log_bin` session variable. This disables binary logging on the RDS for MySQL DB instance, which in turn resets the `binlog_format` session variable to the default value of `ROW` in the database. We recommend that you don't disable backups. For more information about the **Backup retention period** setting, see Settings for DB instances.
6. Choose **Save changes** to save the updates to the DB parameter group.

Because the `binlog_format` parameter is dynamic in RDS for MySQL, you don't need to reboot the DB instance for the changes to apply. (Note that in Aurora MySQL, this parameter is static. For more information, see Configuring Aurora MySQL binary logging.)

> **Important**
>
> Changing a DB parameter group affects all DB instances that use that parameter group. If you want to specify different binary logging formats for different MySQL DB instances in an AWS Region, the DB instances must use different DB parameter groups. These parameter groups identify different logging formats. Assign the appropriate DB parameter group to the each DB instance.

Configuring MySQL binary logging for Multi-AZ DB clusters

Binary logging in Amazon RDS for MySQL Multi-AZ DB clusters records all database changes to support replication, point-in-time recovery, and auditing. In Multi-AZ DB clusters, binary logs

synchronize secondary nodes with the primary node, ensuring data consistency across Availability Zones and enabling seamless failovers.

To optimize binary logging, Amazon RDS supports binary log transaction compression, which reduces the storage requirements for binary logs and improves replication efficiency.

Topics

- Binary log transaction compression for Multi-AZ DB clusters
- Configuring binary log transaction compression for Multi-AZ DB clusters

Binary log transaction compression for Multi-AZ DB clusters

Binary log transaction compression uses the zstd algorithm to reduce the size of transaction data stored in binary logs. When enabled, the MySQL database engine compresses transaction payloads into a single event, minimizing I/O and storage overhead. This feature improves database performance, reduces binary log size, and optimizes resource use for managing and replicating logs in Multi-AZ DB clusters.

Amazon RDS provides binary log transaction compression for RDS for MySQL Multi-AZ DB clusters through the following parameters:

- `binlog_transaction_compression` – When enabled (1), the database engine compresses transaction payloads and writes them to the binary log as a single event. This reduces storage usage and I/O overhead. The parameter is disabled by default.
- `binlog_transaction_compression_level_zstd` – Configures the zstd compression level for binary log transactions. Higher values increase the compression ratio, reducing storage requirements further but increasing CPU and memory usage for compression. The default value is 3, with a range of 1-22.

These parameters let you fine-tune binary log compression based on workload characteristics and resource availability. For more information, see Binary Log Transaction Compression in the MySQL documentation.

Binary log transaction compression has the following main benefits:

- Compression decreases the size of binary logs, particularly for workloads with large transactions or high write volumes.
- Smaller binary logs reduce network and I/O overhead, enhancing replication performance.

- The `binlog_transaction_compression_level_zstd` parameter provides control over the trade-off between compression ratio and resource consumption.

Configuring binary log transaction compression for Multi-AZ DB clusters

To configure binary log transaction compression for an RDS for MySQL Multi-AZ DB cluster, modify the relevant cluster parameter settings to match your workload requirements.

Console

To enable binary log transaction compression

1. Modify the DB cluster parameter group to set the `binlog_transaction_compression` parameter to 1.
2. (Optional) Adjust the value of the `binlog_transaction_compression_level_zstd` parameter based on your workload requirements and resource availability.

For more information, see the section called “Modifying parameters in a DB cluster parameter group”.

AWS CLI

To configure binary log transaction compression using the AWS CLI, use the modify-db-cluster-parameter-group command.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name your-cluster-parameter-group \
  --parameters
 "ParameterName=binlog_transaction_compression,ParameterValue=1,ApplyMethod=pending-
reboot"
```

For Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name your-cluster-parameter-group ^
```

```
  --parameters
 "ParameterName=binlog_transaction_compression,ParameterValue=1,ApplyMethod=pending-
reboot"
```

RDS API

To configure binary log transaction compression using the Amazon RDS API, use the `ModifyDBClusterParameterGroup` operation.

Accessing MySQL binary logs

You can use the mysqlbinlog utility to download or stream binary logs from RDS for MySQL DB instances. The binary log is downloaded to your local computer, where you can perform actions such as replaying the log using the mysql utility. For more information about using the mysqlbinlog utility, see Using mysqlbinlog to back up binary log files in the MySQL documentation.

To run the mysqlbinlog utility against an Amazon RDS instance, use the following options:

- `--read-from-remote-server` – Required.
- `--host` – The DNS name from the endpoint of the instance.
- `--port` – The port used by the instance.
- `--user` – A MySQL user that has been granted the `REPLICATION SLAVE` permission.
- `--password` – The password for the MySQL user, or omit a password value so that the utility prompts you for a password.
- `--raw` – Download the file in binary format.
- `--result-file` – The local file to receive the raw output.
- `--stop-never` – Stream the binary log files.
- `--verbose` – When you use the `ROW` binlog format, include this option to see the row events as pseudo-SQL statements. For more information on the `--verbose` option, see mysqlbinlog row event display in the MySQL documentation.
- Specify the names of one or more binary log files. To get a list of the available logs, use the SQL command `SHOW BINARY LOGS`.

For more information about mysqlbinlog options, see mysqlbinlog — Utility for processing binary log files in the MySQL documentation.

The following examples show how to use the mysqlbinlog utility.

For Linux, macOS, or Unix:

```
mysqlbinlog \
    --read-from-remote-server \
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password \
    --raw \
    --verbose \
    --result-file=/tmp/ \
    binlog.00098
```

For Windows:

```
mysqlbinlog ^
    --read-from-remote-server ^
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password ^
    --raw ^
    --verbose ^
    --result-file=/tmp/ ^
    binlog.00098
```

Binary logs must remain available on the DB instance for the mysqlbinlog utility to access them. To ensure their availability, use the mysql.rds_set_configuration stored procedure and specify a period with enough time for you to download the logs. If this configuration isn't set, Amazon RDS purges the binary logs as soon as possible, leading to gaps in the binary logs that the mysqlbinlog utility retrieves.

The following example sets the retention period to 1 day.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

To display the current setting, use the mysql.rds_show_configuration stored procedure.

```
call mysql.rds_show_configuration;
```

Amazon RDS for Oracle database log files

You can access Oracle alert logs, audit files, and trace files by using the Amazon RDS console or API. For more information about viewing, downloading, and watching file-based database logs, see Monitoring Amazon RDS log files.

The Oracle audit files provided are the standard Oracle auditing files. Amazon RDS supports the Oracle fine-grained auditing (FGA) feature. However, log access doesn't provide access to FGA events that are stored in the `SYS.FGA_LOG$` table and that are accessible through the `DBA_FGA_AUDIT_TRAIL` view.

The `DescribeDBLogFiles` API operation that lists the Oracle log files that are available for a DB instance ignores the `MaxRecords` parameter and returns up to 1,000 records. The call returns `LastWritten` as a POSIX date in milliseconds.

Topics

- Retention schedule
- Working with Oracle trace files
- Publishing Oracle logs to Amazon CloudWatch Logs
- Accessing alert logs and listener logs

Retention schedule

The Oracle database engine might rotate log files if they get very large. To retain audit or trace files, download them. If you store the files locally, you reduce your Amazon RDS storage costs and make more space available for your data.

The following table shows the retention schedule for Oracle alert logs, audit files, and trace files on Amazon RDS.

Log type	Retention schedule
Alert logs	The text alert log is rotated daily with 30-day retention managed by Amazon RDS. The XML alert log is retained for at least seven days. You can access this log by using the `ALERTLOG` view.
Audit files	The default retention period for audit files is seven days. Amazon RDS might delete audit files older than seven days.

Log type	Retention schedule
Trace files	The default retention period for trace files is seven days. Amazon RDS might delete trace files older than seven days.
Listener logs	The default retention period for the listener logs is seven days. Amazon RDS might delete listener logs older than seven days.

> **Note**
>
> Audit files and trace files share the same retention configuration.

Working with Oracle trace files

Following, you can find descriptions of Amazon RDS procedures to create, refresh, access, and delete trace files.

Topics

- Listing files
- Generating trace files and tracing a session
- Retrieving trace files
- Purging trace files

Listing files

You can use either of two procedures to allow access to any file in the `background_dump_dest` path. The first procedure refreshes a view containing a listing of all files currently in `background_dump_dest`.

```
EXEC rdsadmin.manage_tracefiles.refresh_tracefile_listing;
```

After the view is refreshed, query the following view to access the results.

```
SELECT * FROM rdsadmin.tracefile_listing;
```

An alternative to the previous process is to use `FROM table` to stream nonrelational data in a table-like format to list database directory contents.

```
SELECT * FROM TABLE(rdsadmin.rds_file_util.listdir('BDUMP'));
```

The following query shows the text of a log file.

```
SELECT text FROM
 TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP','alert_dbname.log.date'));
```

On a read replica, get the name of the BDUMP directory by querying `V$DATABASE.DB_UNIQUE_NAME`. If the unique name is `DATABASE_B`, then the BDUMP directory is `BDUMP_B`. The following example queries the BDUMP name on a replica and then uses this name to query the contents of `alert_DATABASE.log.2020-06-23`.

```
SELECT 'BDUMP' || (SELECT regexp_replace(DB_UNIQUE_NAME,'.*(_[A-Z])', '\1') FROM V
$DATABASE) AS BDUMP_VARIABLE FROM DUAL;

BDUMP_VARIABLE
--------------
BDUMP_B

SELECT TEXT FROM
 table(rdsadmin.rds_file_util.read_text_file('BDUMP_B','alert_DATABASE.log.2020-06-23'));
```

Generating trace files and tracing a session

Because there are no restrictions on `ALTER SESSION`, many standard methods to generate trace files in Oracle remain available to an Amazon RDS DB instance. The following procedures are provided for trace files that require greater access.

Oracle method	Amazon RDS method
`oradebug hanganalyze 3`	`EXEC rdsadmin.manage_tracefiles.hanganalyze;`
`oradebug dump systemstate 266`	`EXEC rdsadmin.manage_tracefiles.dump_systemstate;`

You can use many standard methods to trace individual sessions connected to an Oracle DB instance in Amazon RDS. To enable tracing for a session, you can run subprograms in PL/SQL packages supplied by Oracle, such as `DBMS_SESSION` and `DBMS_MONITOR`. For more information, see Enabling tracing for a session in the Oracle documentation.

Retrieving trace files

You can retrieve any trace file in `background_dump_dest` using a standard SQL query on an Amazon RDS–managed external table. To use this method, you must execute the procedure to set the location for this table to the specific trace file.

For example, you can use the `rdsadmin.tracefile_listing` view mentioned preceding to list all of the trace files on the system. You can then set the `tracefile_table` view to point to the intended trace file using the following procedure.

```
EXEC
 rdsadmin.manage_tracefiles.set_tracefile_table_location('CUST01_ora_3260_SYSTEMSTATE.trc');
```

The following example creates an external table in the current schema with the location set to the file provided. You can retrieve the contents into a local file using a SQL query.

```
SPOOL /tmp/tracefile.txt
SELECT * FROM tracefile_table;
SPOOL OFF;
```

Purging trace files

Trace files can accumulate and consume disk space. Amazon RDS purges trace files by default and log files that are older than seven days. You can view and set the trace file retention period using the `show_configuration` procedure. You should run the command `SET SERVEROUTPUT ON` so that you can view the configuration results.

The following example shows the current trace file retention period, and then sets a new trace file retention period.

```
# Show the current tracefile retention
SQL> EXEC rdsadmin.rdsadmin_util.show_configuration;
NAME:tracefile retention
VALUE:10080
```

```
DESCRIPTION:tracefile expiration specifies the duration in minutes before tracefiles in
 bdump are automatically deleted.

# Set the tracefile retention to 24 hours:
SQL> EXEC rdsadmin.rdsadmin_util.set_configuration('tracefile retention',1440);
SQL> commit;

#show the new tracefile retention
SQL> EXEC rdsadmin.rdsadmin_util.show_configuration;
NAME:tracefile retention
VALUE:1440
DESCRIPTION:tracefile expiration specifies the duration in minutes before tracefiles in
 bdump are automatically deleted.
```

In addition to the periodic purge process, you can manually remove files from the `background_dump_dest`. The following example shows how to purge all files older than five minutes.

```
EXEC rdsadmin.manage_tracefiles.purge_tracefiles(5);
```

You can also purge all files that match a specific pattern (if you do, don't include the file extension, such as .trc). The following example shows how to purge all files that start with `SCHPOC1_ora_5935`.

```
EXEC rdsadmin.manage_tracefiles.purge_tracefiles('SCHPOC1_ora_5935');
```

Publishing Oracle logs to Amazon CloudWatch Logs

You can configure your RDS for Oracle DB instance to publish log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can analyze the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage.

Amazon RDS publishes each Oracle database log as a separate database stream in the log group. For example, if you configure the export function to include the audit log, audit data is stored in an audit log stream in the `/aws/rds/instance/my_instance/audit` log group. The following table summarizes the requirements for RDS for Oracle to publish logs to Amazon CloudWatch Logs.

Log name	Requirement	Default
Alert log	None. You can't disable this log.	Enabled
Trace log	Set the `trace_enabled` parameter to TRUE or leave it set at the default.	TRUE
Audit log	Set the `audit_trail` parameter to any of the following allowed values: `{ none | os | db [, extended] | xml [, extended] }`	none
Listener log	None. You can't disable this log.	Enabled
Oracle Management Agent log	None. You can't disable this log.	Enabled

This Oracle Management Agent log consists of the log groups shown in the following table.

Log name	CloudWatch log group
emctl.log	oemagent-emctl
emdctlj.log	oemagent-emdctlj
gcagent.log	oemagent-gcagent
gcagent_errors.log	oemagent-gcagent-errors
emagent.nohup	oemagent-emagent-nohup
secure.log	oemagent-secure

For more information, see Locating Management Agent Log and Trace Files in the Oracle documentation.

Console

To publish Oracle DB logs to CloudWatch Logs from the AWS Management Console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**.
4. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.
5. Choose **Continue**, and then choose **Modify DB Instance** on the summary page.

AWS CLI

To publish Oracle logs, you can use the `modify-db-instance` command with the following parameters:

- `--db-instance-identifier`
- `--cloudwatch-logs-export-configuration`

> **Note**
>
> A change to the `--cloudwatch-logs-export-configuration` option is always applied to the DB instance immediately. Therefore, the `--apply-immediately` and `--no-apply-immediately` options have no effect.

You can also publish Oracle logs using the following commands:

- `create-db-instance`
- `restore-db-instance-from-db-snapshot`
- `restore-db-instance-from-s3`
- `restore-db-instance-to-point-in-time`

Example

The following example creates an Oracle DB instance with CloudWatch Logs publishing enabled. The `--cloudwatch-logs-export-configuration` value is a JSON array of strings. The strings can be any combination of `alert`, `audit`, `listener`, and `trace`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration
 '["trace","audit","alert","listener","oemagent"]' \
    --db-instance-class db.m5.large \
    --allocated-storage 20 \
    --engine oracle-ee \
    --engine-version 19.0.0.0.ru-2024-04.rur-2024-04.r1 \
    --license-model bring-your-own-license \
    --master-username myadmin \
    --manage-master-user-password
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration trace alert audit listener oemagent ^
    --db-instance-class db.m5.large ^
    --allocated-storage 20 ^
    --engine oracle-ee ^
    --engine-version 19.0.0.0.ru-2024-04.rur-2024-04.r1 ^
    --license-model bring-your-own-license ^
    --master-username myadmin ^
    --manage-master-user-password
```

Example

The following example modifies an existing Oracle DB instance to publish log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `EnableLogTypes`, and its value is an array of strings with any combination of `alert`, `audit`, `listener`, and `trace`.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["trace","alert","audit","listener","oemagent"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration EnableLogTypes=\"trace\",\"alert\",\"audit
\",\"listener\",\"oemagent\"
```

Example

The following example modifies an existing Oracle DB instance to disable publishing audit and listener log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `DisableLogTypes`, and its value is an array of strings with any combination of `alert`, `audit`, `listener`, and `trace`.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":["audit","listener"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration DisableLogTypes=\"audit\",\"listener\"
```

RDS API

You can publish Oracle DB logs with the RDS API. You can call the `ModifyDBInstance` action with the following parameters:

- `DBInstanceIdentifier`
- `CloudwatchLogsExportConfiguration`

> **Note**
>
> A change to the `CloudwatchLogsExportConfiguration` parameter is always applied to the DB instance immediately. Therefore, the `ApplyImmediately` parameter has no effect.

You can also publish Oracle logs by calling the following RDS API operations:

- `CreateDBInstance`
- `RestoreDBInstanceFromDBSnapshot`
- `RestoreDBInstanceFromS3`
- `RestoreDBInstanceToPointInTime`

Run one of these RDS API operations with the following parameters:

- `DBInstanceIdentifier`
- `EnableCloudwatchLogsExports`
- `Engine`
- `DBInstanceClass`

Other parameters might be required depending on the RDS operation that you run.

Accessing alert logs and listener logs

You can view the alert log using the Amazon RDS console. You can also use the following SQL statement.

```
SELECT message_text FROM alertlog;
```

Access the listener log using Amazon CloudWatch Logs.

> **Note**
>
> Oracle rotates the alert and listener logs when they exceed 10 MB, at which point they are unavailable from Amazon RDS views.

RDS for PostgreSQL database log files

You can monitor the following types of log files:

- PostgreSQL log
- Upgrade log
- IAM database authentication error log

> **Note**
>
> To enable IAM database authentication error logs, you must first enable IAM database authentication for your RDS for PostgreSQL DB instance. For more information about enabling IAM database authentication, see Enabling and disabling IAM database authentication.

RDS for PostgreSQL logs database activities to the default PostgreSQL log file. For an on-premises PostgreSQL DB instance, these messages are stored locally in `log/postgresql.log`. For an RDS for PostgreSQL DB instance, the log file is available on the Amazon RDS instance. These logs are also accessible via the AWS Management Console, where you can view or download them. The default logging level captures login failures, fatal server errors, deadlocks, and query failures.

For more information about how you can view, download, and watch file-based database logs, see Monitoring Amazon RDS log files. To learn more about PostgreSQL logs, see Working with Amazon RDS and Aurora PostgreSQL logs: Part 1 and Working with Amazon RDS and Aurora PostgreSQL logs: Part 2.

In addition to the standard PostgreSQL logs discussed in this topic, RDS for PostgreSQL also supports the PostgreSQL Audit extension (`pgAudit`). Most regulated industries and government agencies need to maintain an audit log or audit trail of changes made to data to comply with legal requirements. For information about installing and using pgAudit, see Using pgAudit to log database activity.

Topics

- Parameters for logging in RDS for PostgreSQL
- Turning on query logging for your RDS for PostgreSQL DB instance
- Publishing PostgreSQL logs to Amazon CloudWatch Logs

Parameters for logging in RDS for PostgreSQL

You can customize the logging behavior for your RDS for PostgreSQL DB instance by modifying various parameters. In the following table you can find the parameters that affect how long the logs are stored, when to rotate the log, and whether to output the log as a CSV (comma-separated value) format. You can also find the text output sent to STDERR, among other settings. To change settings for the parameters that are modifiable, use a custom DB parameter group for your RDS for PostgreSQL instance. For more information, see DB parameter groups for Amazon RDS DB instances.

Parameter	Default	Description
log_destination	stderr	Sets the output format for the log. The default is `stderr` but you can also specify comma-separated value (CSV) by adding `csvlog` to the setting. For more information, see Setting the log destination (`stderr`, `csvlog`).
log_filename	postgresql.log.%Y-%m-%d-%H	Specifies the pattern for the log file name. In addition to the default, this parameter supports `postgresql.log.%Y-%m-%d` and `postgresql.log.%Y-%m-%d-%H%M` for the filename pattern.
log_line_prefix	%t:%r:%u@%d:[%p]:	Defines the prefix for each log line that gets written to `stderr`, to note the time (%t), remote host (%r), user (%u), database (%d), and process ID (%p).
log_rotation_age	60	Minutes after which log file is automatically rotated. You can change this value within the range of 1 and 1440 minutes. For more infor mation, see Setting log file rotation.
log_rotation_size	–	The size (kB) at which the log is automatic ally rotated. By default, this parameter isn't used because logs are rotated based on the

Parameter	Default	Description
		`log_rotation_age` parameter. To learn more, see Setting log file rotation.
rds.log_retention_period	4320	PostgreSQL logs that are older than the specified number of minutes are deleted. The default value of 4320 minutes deletes log files after 3 days. For more information, see Setting the log retention period.

To identify application issues, you can look for query failures, login failures, deadlocks, and fatal server errors in the log. For example, suppose that you converted a legacy application from Oracle to Amazon RDS PostgreSQL, but not all queries converted correctly. These incorrectly formatted queries generate error messages that you can find in the logs to help identify problems. For more information about logging queries, see Turning on query logging for your RDS for PostgreSQL DB instance.

In the following topics, you can find information about how to set various parameters that control the basic details for your PostgreSQL logs.

Topics

- Setting the log retention period
- Setting log file rotation
- Setting the log destination (stderr, csvlog)
- Understanding the log_line_prefix parameter

Setting the log retention period

The `rds.log_retention_period` parameter specifies how long your RDS for PostgreSQL DB instance keeps its log files. The default setting is 3 days (4,320 minutes), but you can set this value to anywhere from 1 day (1,440 minutes) to 7 days (10,080 minutes). Be sure that your RDS for PostgreSQL DB instance has sufficient storage to hold the log files for the period of time.

We recommend that you have your logs routinely published to Amazon CloudWatch Logs so that you can view and analyze system data long after the logs have been removed from your RDS

for PostgreSQL DB instance. For more information, see Publishing PostgreSQL logs to Amazon CloudWatch Logs.

Setting log file rotation

Amazon RDS creates new log files every hour by default. The timing is controlled by the `log_rotation_age` parameter. This parameter has a default value of 60 (minutes), but you can set it to anywhere from 1 minute to 24 hours (1,440 minutes). When it's time for rotation, a new distinct log file is created. The file is named according to the pattern specified by the `log_filename` parameter.

Log files can also be rotated according to their size, as specified in the `log_rotation_size` parameter. This parameter specifies that the log should be rotated when it reaches the specified size (in kilobytes). For an RDS for PostgreSQL DB instance, `log_rotation_size` is unset, that is, there is no value specified. However, you can set the parameter from 0-2097151 kB (kilobytes).

The log file names are based on the file name pattern specified in the `log_filename` parameter. The available settings for this parameter are as follows:

- `postgresql.log.%Y-%m-%d` – Default format for the log file name. Includes the year, month, and date in the name of the log file.
- `postgresql.log.%Y-%m-%d-%H` – Includes the hour in the log file name format.

For more information, see `log_rotation_age` and `log_rotation_size` in the PostgreSQL documentation.

Setting the log destination (`stderr`, `csvlog`)

By default, Amazon RDS PostgreSQL generates logs in standard error (stderr) format. This format is the default setting for the `log_destination` parameter. Each message is prefixed using the pattern specified in the `log_line_prefix` parameter. For more information, see Understanding the log_line_prefix parameter.

RDS for PostgreSQL can also generate the logs in `csvlog` format. The `csvlog` is useful for analyzing the log data as comma-separated values (CSV) data. For example, suppose that you use the `log_fdw` extension to work with your logs as foreign tables. The foreign table created on `stderr` log files contains a single column with log event data. By adding `csvlog` to the `log_destination` parameter, you get the log file in the CSV format with demarcations for the multiple columns of the foreign table. You can now sort and analyze your logs more easily. To learn

how to use the `log_fdw` with `csvlog`, see Using the log_fdw extension to access the DB log using SQL.

If you specify `csvlog` for this parameter, be aware that both `stderr` and `csvlog` files are generated. Be sure to monitor the storage consumed by the logs, taking into account the `rds.log_retention_period` and other settings that affect log storage and turnover. Using `stderr` and `csvlog` more than doubles the storage consumed by the logs.

If you add `csvlog` to `log_destination` and you want to revert to the `stderr` alone, you need to reset the parameter. To do so, open the Amazon RDS Console and then open the custom DB parameter group for your instance. Choose the `log_destination` parameter, choose **Edit parameter**, and then choose **Reset**.

For more information about configuring logging, see Working with Amazon RDS and Aurora PostgreSQL logs: Part 1.

Understanding the log_line_prefix parameter

The `stderr` log format prefixes each log message with the details specified by the `log_line_prefix` parameter. The default value is:

```
%t:%r:%u@%d:[%p]:t
```

Starting from Aurora PostgreSQL version 16, you can also choose:

```
%m:%r:%u@%d:[%p]:%l:%e:%s:%v:%x:%c:%q%a
```

Each log entry sent to stderr includes the following information based on the selected value:

- `%t` – Time of log entry without milliseconds
- `%m` – Time of log entry with milliseconds
- `%r` – Remote host address
- `%u@%d` – User name @ database name
- `[%p]` – Process ID if available
- `%l` – Log line number per session
- `%e` – SQL error code
- `%s` – Process start timestamp

- %v – Virtual transaction id
- %x – Transaction ID
- %c – Session ID
- %q – Non-session terminator
- %a – Application name

Turning on query logging for your RDS for PostgreSQL DB instance

You can collect more detailed information about your database activities, including queries, queries waiting for locks, checkpoints, and many other details by setting some of the parameters listed in the following table. This topic focuses on logging queries.

Parameter	Default	Description
log_connections	–	Logs each successful connection.
log_disconnections	–	Logs the end of each session and its duration.
log_checkpoints	1	Logs each checkpoint.
log_lock_waits	–	Logs long lock waits. By default, this parameter isn't set.
log_min_duration_sample	–	(ms) Sets the minimum execution time above which a sample of statements is logged. Sample size is set using the `log_statement_sample_rate` parameter.
log_min_duration_statement	–	Any SQL statement that runs atleast for the specified amount of time or longer gets logged. By default, this parameter isn't set. Turning on this parameter can help you find unoptimized queries.
log_statement	–	Sets the type of statements logged. By default, this parameter isn't set, but you can change it to `all`, `ddl`, or mod to specify

Parameter	Default	Description
		the types of SQL statements that you want logged. If you specify anything other than none for this parameter, you should also take additional steps to prevent the exposure of passwords in the log files. For more informati on, see Mitigating risk of password exposure when using query logging.
log_statement_samp le_rate	–	The percentage of statements exceeding the time specified in `log_min_duration_s ample` to be logged, expressed as a floating point value between 0.0 and 1.0.
log_statement_stats	–	Writes cumulative performance statistics to the server log.

Using logging to find slow performing queries

You can log SQL statements and queries to help find slow performing queries. You turn on this capability by modifying the settings in the `log_statement` and `log_min_duration` parameters as outlined in this section. Before turning on query logging for your RDS for PostgreSQL DB instance, you should be aware of possible password exposure in the logs and how to mitigate the risks. For more information, see Mitigating risk of password exposure when using query logging.

Following, you can find reference information about the `log_statement` and `log_min_duration` parameters.

log_statement

This parameter specifies the type of SQL statements that should get sent to the log. The default value is none. If you change this parameter to `all`, `ddl`, or `mod`, be sure to apply recommended actions to mitigate the risk of exposing passwords in the logs. For more information, see Mitigating risk of password exposure when using query logging.

all

Logs all statements. This setting is recommended for debugging purposes.

ddl

Logs all data definition language (DDL) statements, such as CREATE, ALTER, DROP, and so on.

mod

Logs all DDL statements and data manipulation language (DML) statements, such as INSERT, UPDATE, and DELETE, which modify the data.

none

No SQL statements get logged. We recommend this setting to avoid the risk of exposing passwords in the logs.

log_min_duration_statement

Any SQL statement that runs atleast for the specified amount of time or longer gets logged. By default, this parameter isn't set. Turning on this parameter can help you find unoptimized queries.

–1–2147483647

The number of milliseconds (ms) of runtime over which a statement gets logged.

To set up query logging

These steps assume that your RDS for PostgreSQL DB instance uses a custom DB parameter group.

1. Set the `log_statement` parameter to `all`. The following example shows the information that is written to the `postgresql.log` file with this parameter setting.

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: statement:
 SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: QUERY
 STATISTICS
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:DETAIL: ! system
 usage stats:
! 0.017355 s user, 0.000000 s system, 0.168593 s elapsed
! [0.025146 s user, 0.000000 s system total]
! 36644 kB max resident size
! 0/8 [0/8] filesystem blocks in/out
```

```
! 0/733 [0/1364] page faults/reclaims, 0 [0] swaps
! 0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
! 19/0 [27/0] voluntary/involuntary context switches
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: SELECT
 feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:ERROR: syntax error
 at or near "ORDER" at character 1
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: ORDER BY
 s.confidence DESC;
----------------------- END OF LOG ----------------------
```

2. Set the `log_min_duration_statement` parameter. The following example shows the information that is written to the `postgresql.log` file when the parameter is set to 1.

   Queries that exceed the duration specified in the `log_min_duration_statement` parameter are logged. The following shows an example. You can view the log file for your RDS for PostgreSQL DB instance in the Amazon RDS Console.

```
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: statement: DROP
 table comments;
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: duration:
 167.754 ms
2022-10-05 19:08:07 UTC::@:[355]:LOG: checkpoint starting: time
2022-10-05 19:08:08 UTC::@:[355]:LOG: checkpoint complete: wrote 11 buffers
 (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=1.013 s, sync=0.006 s,
 total=1.033 s; sync files=8, longest=0.004 s, average=0.001 s; distance=131028 kB,
 estimate=131028 kB
----------------------- END OF LOG ----------------------
```

Mitigating risk of password exposure when using query logging

We recommend that you keep `log_statement` set to none to avoid exposing passwords. If you set `log_statement` to `all`, `ddl`, or `mod`, we recommend that you take one or more of the following steps.

- For the client, encrypt sensitive information. For more information, see Encryption Options in the PostgreSQL documentation. Use the ENCRYPTED (and UNENCRYPTED) options of the CREATE and ALTER statements. For more information, see CREATE USER in the PostgreSQL documentation.

- For your RDS for PostgreSQL DB instance, set up and use the PostgreSQL Auditing (pgAudit) extension. This extension redacts sensitive information in CREATE and ALTER statements sent to the log. For more information, see Using pgAudit to log database activity.
- Restrict access to the CloudWatch logs.
- Use stronger authentication mechanisms such as IAM.

Publishing PostgreSQL logs to Amazon CloudWatch Logs

To store your PostgreSQL log records in highly durable storage, you can use Amazon CloudWatch Logs. With CloudWatch Logs, you can also perform real-time analysis of log data and use CloudWatch to view metrics and create alarms. For example, if you set `log_statement` to `ddl`, you can set up an alarm to alert you whenever a DDL statement is executed. You can choose to have your PostgreSQL logs uploaded to CloudWatch Logs during the process of creating your RDS for PostgreSQL DB instance. If you chose not to upload logs at that time, you can later modify your instance to start uploading logs from that point forward. In other words, existing logs aren't uploaded. Only new logs are uploaded as they're created on your modified RDS for PostgreSQL DB instance.

All currently available RDS for PostgreSQL versions support publishing log files to CloudWatch Logs. For more information, see Amazon RDS for PostgreSQL updates in the *Amazon RDS for PostgreSQL Release Notes.*.

To work with CloudWatch Logs, configure your RDS for PostgreSQL DB instance to publish log data to a log group.

You can publish the following log types to CloudWatch Logs for RDS for PostgreSQL:

- PostgreSQL log
- Upgrade log
- IAM database authentication error log

After you complete the configuration, Amazon RDS publishes the log events to log streams within a CloudWatch log group. For example, the PostgreSQL log data is stored within the log group `/aws/rds/instance/my_instance/postgresql`. To view your logs, open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

Console

To publish PostgreSQL logs to CloudWatch Logs using the console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify, and then choose **Modify**.
4. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.

   The **Log exports** section is available only for PostgreSQL versions that support publishing to CloudWatch Logs.
5. Choose **Continue**, and then choose **Modify DB Instance** on the summary page.

AWS CLI

You can publish PostgreSQL logs with the AWS CLI. You can call the `modify-db-instance` command with the following parameters.

- `--db-instance-identifier`
- `--cloudwatch-logs-export-configuration`

> **Note**
>
> A change to the `--cloudwatch-logs-export-configuration` option is always applied to the DB instance immediately. Therefore, the `--apply-immediately` and `--no-apply-immediately` options have no effect.

You can also publish PostgreSQL logs by calling the following CLI commands:

- `create-db-instance`
- `restore-db-instance-from-db-snapshot`
- `restore-db-instance-to-point-in-time`

Run one of these CLI commands with the following options:

- `--db-instance-identifier`
- `--enable-cloudwatch-logs-exports`
- `--db-instance-class`
- `--engine`

Other options might be required depending on the CLI command you run.

Example Modify an instance to publish logs to CloudWatch Logs

The following example modifies an existing PostgreSQL DB instance to publish log files to CloudWatch Logs. The `--cloudwatch-logs-export-configuration` value is a JSON object. The key for this object is `EnableLogTypes`, and its value is an array of strings with any combination of `postgresql` and `upgrade`.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql",
 "upgrade"]}'
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["postgresql","upgrade"]}'
```

Example Create an instance to publish logs to CloudWatch Logs

The following example creates a PostgreSQL DB instance and publishes log files to CloudWatch Logs. The `--enable-cloudwatch-logs-exports` value is a JSON array of strings. The strings can be any combination of `postgresql` and `upgrade`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
```

```
    --db-instance-identifier mydbinstance \
    --enable-cloudwatch-logs-exports '["postgresql","upgrade"]' \
    --db-instance-class db.m4.large \
    --engine postgres
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --enable-cloudwatch-logs-exports '["postgresql","upgrade"]' ^
    --db-instance-class db.m4.large ^
    --engine postgres
```

RDS API

You can publish PostgreSQL logs with the RDS API. You can call the `ModifyDBInstance` action with the following parameters:

- `DBInstanceIdentifier`
- `CloudwatchLogsExportConfiguration`

> **Note**
>
> A change to the `CloudwatchLogsExportConfiguration` parameter is always applied to the DB instance immediately. Therefore, the `ApplyImmediately` parameter has no effect.

You can also publish PostgreSQL logs by calling the following RDS API operations:

- `CreateDBInstance`
- `RestoreDBInstanceFromDBSnapshot`
- `RestoreDBInstanceToPointInTime`

Run one of these RDS API operations with the following parameters:

- `DBInstanceIdentifier`

- `EnableCloudwatchLogsExports`
- `Engine`
- `DBInstanceClass`

Other parameters might be required depending on the operation that you run.

Monitoring Amazon RDS API calls in AWS CloudTrail

AWS CloudTrail is an AWS service that helps you audit your AWS account. AWS CloudTrail is turned on for your AWS account when you create it. For more information about CloudTrail, see the AWS CloudTrail User Guide.

Topics

- CloudTrail integration with Amazon RDS
- Amazon RDS log file entries

CloudTrail integration with Amazon RDS

All Amazon RDS actions are logged by CloudTrail. CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon RDS.

CloudTrail events

CloudTrail captures API calls for Amazon RDS as events. An *event* represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. Events include calls from the Amazon RDS console and from code calls to the Amazon RDS API operations.

Amazon RDS activity is recorded in a CloudTrail event in **Event history**. You can use the CloudTrail console to view the last 90 days of recorded API activity and events in an AWS Region. For more information, see Viewing events with CloudTrail event history.

CloudTrail trails

For an ongoing record of events in your AWS account, including events for Amazon RDS, create a *trail*. A trail is a configuration that enables delivery of events to a specified Amazon S3 bucket. CloudTrail typically delivers log files within 15 minutes of account activity.

> **Note**
>
> If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**.

You can create two types of trails for an AWS account: a trail that applies to all Regions, or a trail that applies to one Region. By default, when you create a trail in the console, the trail applies to all Regions.

Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions and Receiving CloudTrail log files from multiple accounts

Amazon RDS log file entries

CloudTrail log files contain one or more log entries. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateDBInstance` action.

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
```

```
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "8.0.28",
        "dbInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-8-0"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql8.0",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
```

```
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1e"},
                    "subnetIdentifier": "subnet-d7c825e8",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1f"},
                    "subnetIdentifier": "subnet-6746046b",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1c"},
                    "subnetIdentifier": "subnet-bac383e0",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1d"},
                    "subnetIdentifier": "subnet-42599426",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1a"},
                    "subnetIdentifier": "subnet-da327bf6",
                    "subnetStatus": "Active"
                }
            ],
            "vpcId": "vpc-136a4c6a",
            "subnetGroupStatus": "Complete"
        },
        "masterUsername": "myawsuser",
        "multiAZ": false,
        "autoMinorVersionUpgrade": true,
        "engine": "mysql",
        "cACertificateIdentifier": "rds-ca-2015",
        "dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
        "dBSecurityGroups": [],
        "pendingModifiedValues": {
            "masterUserPassword": "****",
            "pendingCloudwatchLogsExports": {
                "logTypesToEnable": [
                    "audit",
                    "error",
```

```
                    "general",
                    "slowquery"
                ]
            }
        },
        "dBInstanceStatus": "creating",
        "publiclyAccessible": true,
        "domainMemberships": [],
        "copyTagsToSnapshot": false,
        "dBInstanceIdentifier": "test-instance",
        "licenseModel": "general-public-license",
        "iAMDatabaseAuthenticationEnabled": false,
        "performanceInsightsEnabled": false,
        "vpcSecurityGroups": [
            {
                "status": "active",
                "vpcSecurityGroupId": "sg-f839b688"
            }
        ]
    },
    "requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
    "eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

As shown in the `userIdentity` element in the preceding example, every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information about the `userIdentity`, see the CloudTrail userIdentity element. For more information about `CreateDBInstance` and other Amazon RDS actions, see the Amazon RDS API Reference.

Monitoring Amazon RDS with Database Activity Streams

By using Database Activity Streams, you can monitor near real-time streams of database activity.

Topics

- Overview of Database Activity Streams
- Configuring unified auditing for Oracle Database
- Configuring auditing policy for Amazon RDS for Microsoft SQL Server
- Starting a database activity stream
- Modifying a database activity stream for Amazon RDS
- Getting the status of a database activity stream
- Stopping a database activity stream
- Monitoring database activity streams
- IAM policy examples for database activity streams

Overview of Database Activity Streams

As an Amazon RDS database administrator, you need to safeguard your database and meet compliance and regulatory requirements. One strategy is to integrate database activity streams with your monitoring tools. In this way, you monitor and set alarms for auditing activity in your database.

Security threats are both external and internal. To protect against internal threats, you can control administrator access to data streams by configuring the Database Activity Streams feature. Amazon RDS DBAs don't have access to the collection, transmission, storage, and processing of the streams.

Contents

- How database activity streams work
- Auditing in Oracle Database and Microsoft SQL Server Database
  - Unified auditing in Oracle Database
  - Auditing in Microsoft SQL Server
  - Non-native audit fields for Oracle Database and SQL Server
  - DB parameter group override
- Asynchronous mode for database activity streams

- Requirements and limitations for database activity streams
- Region and version availability
- Supported DB instance classes for database activity streams

How database activity streams work

Amazon RDS pushes activities to an Amazon Kinesis data stream in near real time. The Kinesis stream is created automatically. From Kinesis, you can configure AWS services such as Amazon Data Firehose and AWS Lambda to consume the stream and store the data.

> **Important**
>
> Use of the database activity streams feature in Amazon RDS is free, but Amazon Kinesis charges for a data stream. For more information, see Amazon Kinesis Data Streams pricing.

You can configure applications for compliance management to consume database activity streams. These applications can use the stream to generate alerts and audit activity on your database.

Amazon RDS supports database activity streams in Multi-AZ deployments. In this case, database activity streams audit both the primary and standby instances.

Auditing in Oracle Database and Microsoft SQL Server Database

Auditing is the monitoring and recording of configured database actions. Amazon RDS doesn't capture database activity by default. You create and manage audit policies in your database yourself.

Topics

- Unified auditing in Oracle Database
- Auditing in Microsoft SQL Server
- Non-native audit fields for Oracle Database and SQL Server
- DB parameter group override

Unified auditing in Oracle Database

In an Oracle database, a *unified audit policy* is a named group of audit settings that you can use to audit an aspect of user behavior. A policy can be as simple as auditing the activities of a single user. You can also create complex audit policies that use conditions.

An Oracle database writes audit records, including SYS audit records, to the *unified audit trail*. For example, if an error occurs during an `INSERT` statement, standard auditing indicates the error number and the SQL that was run. The audit trail resides in a read-only table in the `AUDSYS` schema. To access these records, query the `UNIFIED_AUDIT_TRAIL` data dictionary view.

Typically, you configure database activity streams as follows:

1. Create an Oracle Database audit policy by using the `CREATE AUDIT POLICY` command.

   The Oracle Database generates audit records.
2. Activate the audit policy by using the `AUDIT POLICY` command.
3. Configure database activity streams.

   Only activities that match the Oracle Database audit policies are captured and sent to the Amazon Kinesis data stream. When database activity streams are enabled, an Oracle database administrator can't alter the audit policy or remove audit logs.

To learn more about unified audit policies, see About Auditing Activities with Unified Audit Policies and AUDIT in the *Oracle Database Security Guide*.

Auditing in Microsoft SQL Server

Database Activity Stream uses SQLAudit feature to audit the SQL Server database.

RDS for SQL Server instance contains the following:

- Server audit – The SQL server audit collects a single instance of server or database-level actions, and a group of actions to monitor. The server-level audits `RDS_DAS_AUDIT` and `RDS_DAS_AUDIT_CHANGES` are managed by RDS.
- Server audit specification – The server audit specification records the server-level events. You can modify the `RDS_DAS_SERVER_AUDIT_SPEC` specification. This specification is linked to the server audit `RDS_DAS_AUDIT`. The `RDS_DAS_CHANGES_AUDIT_SPEC` specification is managed by RDS.

- Database audit specification – The database audit specification records the database-level events. You can create a database audit specification RDS_DAS_DB_<name> and link it to RDS_DAS_AUDIT server audit.

You can configure database activity streams by using the console or CLI. Typically, you configure database activity streams as follows:

1. (Optional) Create a database audit specification with the CREATE DATABASE AUDIT SPECIFICATION command and link it to RDS_DAS_AUDIT server audit.
2. (Optional) Modify the server audit specification with the ALTER SERVER AUDIT SPECIFICATION command and define the policies.
3. Activate the database and server audit policies. For example:

   ```
   ALTER DATABASE AUDIT SPECIFICATION [<Your database specification>] WITH (STATE=ON)

   ALTER SERVER AUDIT SPECIFICATION [RDS_DAS_SERVER_AUDIT_SPEC] WITH (STATE=ON)
   ```
4. Configure database activity streams.

   Only activities that match the server and database audit policies are captured and sent to the Amazon Kinesis data stream. When database activity streams are enabled and the policies are locked, a database administrator can't alter the audit policy or remove audit logs.

   > **Important**
   >
   > If the database audit specification for a specific database is enabled and the policy is in a locked state, then the database can't be dropped.

For more information about SQL Server auditing, see SQL Server Audit Components in the *Microsoft SQL Server documentation*.

Non-native audit fields for Oracle Database and SQL Server

When you start a database activity stream, every database event generates a corresponding activity stream event. For example, a database user might run `SELECT` and `INSERT` statements. The database audits these events and sends them to an Amazon Kinesis data stream.

The events are represented in the stream as JSON objects. A JSON object contains a `DatabaseActivityMonitoringRecord`, which contains a `databaseActivityEventList` array. Predefined fields in the array include `class`, `clientApplication`, and `command`.

By default, an activity stream doesn't include engine-native audit fields. You can configure Amazon RDS for Oracle and SQL Server so that it includes these extra fields in the `engineNativeAuditFields` JSON object.

In Oracle Database, most events in the unified audit trail map to fields in the RDS data activity stream. For example, the `UNIFIED_AUDIT_TRAIL.SQL_TEXT` field in unified auditing maps to the `commandText` field in a database activity stream. However, Oracle Database audit fields such as `OS_USERNAME` don't map to predefined fields in a database activity stream.

In SQL Server, most of the event's fields that are recorded by the SQLAudit map to the fields in RDS database activity stream. For example, the `code` field from `sys.fn_get_audit_file` in the audit maps to the `commandText` field in a database activity stream. However, SQL Server database audit fields, such as `permission_bitmask`, don't map to predefined fields in a database activity stream.

For more information about databaseActivityEventList, see databaseActivityEventList JSON array for database activity streams.

DB parameter group override

Typically, you turn on unified auditing in RDS for Oracle by attaching a parameter group. However, Database Activity Streams require additional configuration. To improve your customer experience, Amazon RDS performs the following:

- If you activate an activity stream, RDS for Oracle ignores the auditing parameters in the parameter group.
- If you deactivate an activity stream, RDS for Oracle stops ignoring the auditing parameters.

The database activity stream for SQL Server is independent of any parameters you set in the SQL Audit option.

Asynchronous mode for database activity streams

Activity streams in Amazon RDS are always asynchronous. When a database session generates an activity stream event, the session returns to normal activities immediately. In the background, Amazon RDS makes the activity stream event into a durable record.

If an error occurs in the background task, Amazon RDS generates an event. This event indicates the beginning and end of any time windows where activity stream event records might have been lost. Asynchronous mode favors database performance over the accuracy of the activity stream.

Requirements and limitations for database activity streams

In RDS, database activity streams have the following requirements and limitations:

- Amazon Kinesis is required for database activity streams.
- AWS Key Management Service (AWS KMS) is required for database activity streams because they are always encrypted.
- Applying additional encryption to your Amazon Kinesis data stream is incompatible with database activity streams, which are already encrypted with your AWS KMS key.
- You create and manage audit policies yourself. Unlike Amazon Aurora, RDS for Oracle doesn't capture database activities by default.
- You create and manage audit policies or specifications yourself. Unlike Amazon Aurora, Amazon RDS doesn't capture database activities by default.
- In a Multi-AZ deployment, start the database activity stream only on the primary DB instance. The activity stream audits both the primary and standby DB instances automatically. No additional steps are required during a failover.
- Renaming a DB instance doesn't create a new Kinesis stream.
- CDBs aren't supported for RDS for Oracle.
- Read replicas aren't supported.

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability with database activity streams, see Supported Regions and DB engines for database activity streams in Amazon RDS.

Supported DB instance classes for database activity streams

For RDS for Oracle you can use database activity streams with the following DB instance classes:

- db.m4.*large
- db.m5.*large
- db.m5d.*large
- db.m6i.*large
- db.r4.*large
- db.r5.*large
- db.r5.*large.tpc*.mem*x
- db.r5b.*large
- db.r5b.*large.tpc*.mem*x
- db.r5d.*large
- db.r6i.*large
- db.r6i.*large.tpc*.mem*x
- db.x2idn.*large
- db.x2iedn.*large
- db.x2iezn.*large
- db.z1d.*large

For RDS for SQL Server you can use database activity streams with the following DB instance classes:

- db.m4.*large
- db.m5.*large
- db.m5d.*large
- db.m6i.*large
- db.r4.*large
- db.r5.*large
- db.r5b.*large
- db.r5d.*large

- db.r6i.*large
- db.x1e.*large
- db.x2iedn.*large
- db.z1d.*large

For more information about instance class types, see DB instance classes.

Configuring unified auditing for Oracle Database

When you configure unified auditing for use with database activity streams, the following situations are possible:

- Unified auditing isn't configured for your Oracle database.

  In this case, create new policies with the `CREATE AUDIT POLICY` command, then activate them with the `AUDIT POLICY` command. The following example creates and activates a policy to monitor users with specific privileges and roles.

  ```
  CREATE AUDIT POLICY table_pol
  PRIVILEGES CREATE ANY TABLE, DROP ANY TABLE
  ROLES emp_admin, sales_admin;

  AUDIT POLICY table_pol;
  ```

  For complete instructions, see Configuring Audit Policies in the Oracle Database documentation.
- Unified auditing is configured for your Oracle database.

  When you activate a database activity stream, RDS for Oracle automatically clears existing audit data. It also revokes audit trail privileges. RDS for Oracle can no longer do the following:
  - Purge unified audit trail records.
  - Add, delete, or modify the unified audit policy.
  - Update the last archived timestamp.

  Important

  We strongly recommend that you back up your audit data before activating a database activity stream.

For a description of the UNIFIED_AUDIT_TRAIL view, see UNIFIED_AUDIT_TRAIL. If you have an account with Oracle Support, see How To Purge The UNIFIED AUDIT TRAIL.

Configuring auditing policy for Amazon RDS for Microsoft SQL Server

A SQL Server database instance has the server audit RDS_DAS_AUDIT, which is managed by Amazon RDS. You can define the policies to record server events in the server audit specification RDS_DAS_SERVER_AUDIT_SPEC. You can create a database audit specification, such as RDS_DAS_DB_<name>, and define the policies to record database events. For the list of server and database level audit action groups, see SQL Server Audit Action Groups and Actions in the *Microsoft SQL Server documentation*.

The default server policy monitors only failed logins and changes to any database or server audit specifications for database activity streams.

Limitations for the audit and audit specifications include the following:

- You can't modify the server or database audit specifications when the database activity stream is in a *locked* state.
- You can't modify the server audit RDS_DAS_AUDIT specification.
- You can't modify the SQL Server audit RDS_DAS_CHANGES or its related server audit specification RDS_DAS_CHANGES_AUDIT_SPEC.
- When creating a database audit specification, you must use the format RDS_DAS_DB_<name> for example, RDS_DAS_DB_databaseActions.

Important

For smaller instance classes, we recommend that you don't audit all but only the data that is required. This helps to reduce the performance impact of Database Activity Streams on these instance classes.

The following sample code modifies the server audit specification RDS_DAS_SERVER_AUDIT_SPEC and audits any logout and successful login actions:

```
ALTER SERVER AUDIT SPECIFICATION [RDS_DAS_SERVER_AUDIT_SPEC]
```

```
      WITH (STATE=OFF);
ALTER SERVER AUDIT SPECIFICATION [RDS_DAS_SERVER_AUDIT_SPEC]
      ADD (LOGOUT_GROUP),
      ADD (SUCCESSFUL_LOGIN_GROUP)
      WITH (STATE = ON );
```

The following sample code creates a database audit specification `RDS_DAS_DB_database_spec` and attaches it to the server audit `RDS_DAS_AUDIT`:

```
USE testDB;
CREATE DATABASE AUDIT SPECIFICATION [RDS_DAS_DB_database_spec]
     FOR SERVER AUDIT [RDS_DAS_AUDIT]
     ADD ( INSERT, UPDATE, DELETE
          ON testTable BY testUser )
     WITH (STATE = ON);
```

After the audit specifications are configured, make sure that the specifications `RDS_DAS_SERVER_AUDIT_SPEC` and `RDS_DAS_DB_<name>` are set to a state of `ON`. Now they can send the audit data to your database activity stream.

Starting a database activity stream

When you start an activity stream for the DB instance, each database activity event that you configured in the audit policy generates an activity stream event. SQL commands such as `CONNECT` and `SELECT` generate access events. SQL commands such as `CREATE` and `INSERT` generate change events.

> **Important**
>
> Turning on an activity stream for an Oracle DB instance clears existing audit data. It also revokes audit trail privileges. When the stream is enabled, RDS for Oracle can no longer do the following:
>
> - Purge unified audit trail records.
> - Add, delete, or modify the unified audit policy.
> - Update the last archived time stamp.

Console

To start a database activity stream

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the Amazon RDS database instance on which you want to start an activity stream. In a Multi-AZ deployment, start the stream on only the primary instance. The activity stream audits both the primary and the standby instances.
4. For **Actions**, choose **Start activity stream**.

   The **Start database activity stream:** *name* window appears, where *name* is your RDS instance.
5. Enter the following settings:

   - For **AWS KMS key**, choose a key from the list of AWS KMS keys.

     Amazon RDS uses the KMS key to encrypt the key that in turn encrypts database activity. Choose a KMS key other than the default key. For more information about encryption keys and AWS KMS, see What is AWS Key Management Service? in the *AWS Key Management Service Developer Guide.*
   - For **Database activity events**, choose **Enable engine-native audit fields** to include the engine specific audit fields.
   - Choose **Immediately**.

     When you choose **Immediately**, the RDS instance restarts right away. If you choose **During the next maintenance window**, the RDS instance doesn't restart right away. In this case, the database activity stream doesn't start until the next maintenance window.
6. Choose **Start database activity stream**.

   The status for the the database shows that the activity stream is starting.

> **Note**
>
> If you get the error `You can't start a database activity stream in this configuration`, check Supported DB instance classes for database activity streams to see whether your RDS instance is using a supported instance class.

AWS CLI

To start database activity streams for a DB instance, configure the database using the start-activity-stream AWS CLI command.

- `--resource-arn` *arn* – Specifies the Amazon Resource Name (ARN) of the DB instance.
- `--kms-key-id` *key* – Specifies the KMS key identifier for encrypting messages in the database activity stream. The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the AWS KMS key.
- `--engine-native-audit-fields-included` – Includes engine-specific auditing fields in the data stream. To exclude these fields, specify `--no-engine-native-audit-fields-included` (default).

The following example starts a database activity stream for a DB instance in asynchronous mode.

For Linux, macOS, or Unix:

```
aws rds start-activity-stream \
    --mode async \
    --kms-key-id my-kms-key-arn \
    --resource-arn my-instance-arn \
    --engine-native-audit-fields-included \
    --apply-immediately
```

For Windows:

```
aws rds start-activity-stream ^
    --mode async ^
    --kms-key-id my-kms-key-arn ^
    --resource-arn my-instance-arn ^
    --engine-native-audit-fields-included ^
    --apply-immediately
```

Amazon RDS API

To start database activity streams for a DB instance, configure the instance using the StartActivityStream operation.

Call the action with the parameters below:

- `Region`
- `KmsKeyId`
- `ResourceArn`
- `Mode`
- `EngineNativeAuditFieldsIncluded`

Modifying a database activity stream for Amazon RDS

You might want to customize your Amazon RDS audit policy when your activity stream is started. If you don't want to lose time and data by stopping your activity stream, you can change the *audit policy state* to either of the following settings:

Locked (default)

The audit policies in your database are read-only.

Unlocked

The audit policies in your database are read/write.

The basic steps are as follows:

1. Modify the audit policy state to unlocked.
2. Customize your audit policy.
3. Modify the audit policy state to locked.

Console

To modify the audit policy state of your activity stream

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. For **Actions**, choose **Modify database activity stream**.

   The **Modify database activity stream: *name*** window appears, where *name* is your RDS instance.
4. Choose either of the following options:

Locked

When you lock your audit policy, it becomes read-only. You can't edit your audit policy unless you unlock the policy or stop the activity stream.

Unlocked

When you unlock your audit policy, it becomes read/write. You can edit your audit policy while the activity stream is started.

5. Choose **Modify DB activity stream**.

   The status for the Amazon RDS database shows **Configuring activity stream**.

6. (Optional) Choose the DB instance link. Then choose the **Configuration** tab.

   The **Audit policy status** field shows one of the following values:

   - **Locked**
   - **Unlocked**
   - **Locking policy**
   - **Unlocking policy**

AWS CLI

To modify the activity stream state for the database instance, use the modify-activity-stream AWS CLI command.

Option	Required?	Description
`--resource-arn` *my-instance-ARN*	Yes	The Amazon Resource Name (ARN) of your RDS database instance.
`--audit-policy-state`	No	The new state of the audit policy for the database activity stream on your instance: `locked` or `unlocked`.

The following example unlocks the audit policy for the activity stream started on *my-instance-ARN*.

For Linux, macOS, or Unix:

```
aws rds modify-activity-stream \
    --resource-arn my-instance-ARN \
    --audit-policy-state unlocked
```

For Windows:

```
aws rds modify-activity-stream ^
    --resource-arn my-instance-ARN ^
    --audit-policy-state unlocked
```

The following example describes the instance *my-instance*. The partial sample output shows that the audit policy is unlocked.

```
aws rds describe-db-instances --db-instance-identifier my-instance

{
    "DBInstances": [
        {
            ...
            "Engine": "oracle-ee",
            ...
            "ActivityStreamStatus": "started",
            "ActivityStreamKmsKeyId": "ab12345e-1111-2bc3-12a3-ab1cd12345e",
            "ActivityStreamKinesisStreamName": "aws-rds-das-db-
AB1CDEFG23GHIJK4LMNOPQRST",
            "ActivityStreamMode": "async",
            "ActivityStreamEngineNativeAuditFieldsIncluded": true,
            "ActivityStreamPolicyStatus": "unlocked",
            ...
        }
    ]
}
```

RDS API

To modify the policy state of your database activity stream, use the ModifyActivityStream operation.

Call the action with the parameters below:

- `AuditPolicyState`
- `ResourceArn`

Getting the status of a database activity stream

You can get the status of an activity stream for your Amazon RDS database instance using the console or AWS CLI.

Console

To get the status of a database activity stream

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance link.
3. Choose the **Configuration** tab, and check **Database activity stream** for status.

AWS CLI

You can get the activity stream configuration for a database instance as the response to a describe-db-instances CLI request.

The following example describes *my-instance*.

```
aws rds --region my-region describe-db-instances --db-instance-identifier my-db
```

The following example shows a JSON response. The following fields are shown:

- `ActivityStreamKinesisStreamName`
- `ActivityStreamKmsKeyId`
- `ActivityStreamStatus`
- `ActivityStreamMode`
- `ActivityStreamPolicyStatus`

```
{
    "DBInstances": [
        {
            ...
```

```
            "Engine": "oracle-ee",
            ...
            "ActivityStreamStatus": "starting",
            "ActivityStreamKmsKeyId": "ab12345e-1111-2bc3-12a3-ab1cd12345e",
            "ActivityStreamKinesisStreamName": "aws-rds-das-db-
AB1CDEFG23GHIJK4LMNOPQRST",
            "ActivityStreamMode": "async",
            "ActivityStreamEngineNativeAuditFieldsIncluded": true,
            "ActivityStreamPolicyStatus": locked",
            ...
        }
    ]
}
```

RDS API

You can get the activity stream configuration for a database as the response to a DescribeDBInstances operation.

Stopping a database activity stream

You can stop an activity stream using the console or AWS CLI.

If you delete your Amazon RDS database instance, the activity stream is stopped and the underlying Amazon Kinesis stream is deleted automatically.

Console

To turn off an activity stream

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose a database that you want to stop the database activity stream for.
4. For **Actions**, choose **Stop activity stream**. The **Database Activity Stream** window appears.
    a. Choose **Immediately**.

        When you choose **Immediately**, the RDS instance restarts right away. If you choose **During the next maintenance window**, the RDS instance doesn't restart right away. In this case, the database activity stream doesn't stop until the next maintenance window.
    b. Choose **Continue**.

AWS CLI

To stop database activity streams for your database, configure the DB instance using the AWS CLI command stop-activity-stream. Identify the AWS Region for the DB instance using the `--region` parameter. The `--apply-immediately` parameter is optional.

For Linux, macOS, or Unix:

```
aws rds --region MY_REGION \
    stop-activity-stream \
    --resource-arn MY_DB_ARN \
    --apply-immediately
```

For Windows:

```
aws rds --region MY_REGION ^
    stop-activity-stream ^
    --resource-arn MY_DB_ARN ^
    --apply-immediately
```

RDS API

To stop database activity streams for your database, configure the DB instance using the StopActivityStream operation. Identify the AWS Region for the DB instance using the `Region` parameter. The `ApplyImmediately` parameter is optional.

Monitoring database activity streams

Database activity streams monitor and report activities. The stream of activity is collected and transmitted to Amazon Kinesis. From Kinesis, you can monitor the activity stream, or other services and applications can consume the activity stream for further analysis. You can find the underlying Kinesis stream name by using the AWS CLI command `describe-db-instances` or the RDS API `DescribeDBInstances` operation.

Amazon RDS manages the Kinesis stream for you as follows:

- Amazon RDS creates the Kinesis stream automatically with a 24-hour retention period.
- Amazon RDS scales the Kinesis stream if necessary.
- If you stop the database activity stream or delete the DB instance, Amazon RDS deletes the Kinesis stream.

The following categories of activity are monitored and put in the activity stream audit log:

- **SQL commands** – All SQL commands are audited, and also prepared statements, built-in functions, and functions in PL/SQL. Calls to stored procedures are audited. Any SQL statements issued inside stored procedures or functions are also audited.
- **Other database information** – Activity monitored includes the full SQL statement, the row count of affected rows from DML commands, accessed objects, and the unique database name. Database activity streams also monitor the bind variables and stored procedure parameters.

  > **Important**
  >
  > The full SQL text of each statement is visible in the activity stream audit log, including any sensitive data. However, database user passwords are redacted if Oracle can determine them from the context, such as in the following SQL statement.
  >
  > ```
  > ALTER ROLE role-name WITH password
  > ```

- **Connection information** – Activity monitored includes session and network information, the server process ID, and exit codes.

If an activity stream has a failure while monitoring your DB instance, you are notified through RDS events.

In the following sections, you can access, audit, and process database activity streams.

Topics

- Accessing an activity stream from Amazon Kinesis
- Audit log contents and examples for database activity streams
- databaseActivityEventList JSON array for database activity streams
- Processing a database activity stream using the AWS SDK

Accessing an activity stream from Amazon Kinesis

When you enable an activity stream for a database, a Kinesis stream is created for you. From Kinesis, you can monitor your database activity in real time. To further analyze database activity, you can connect your Kinesis stream to consumer applications. You can also connect the stream to

compliance management applications such as IBM's Security Guardium or Imperva's SecureSphere Database Audit and Protection.

You can access your Kinesis stream either from the RDS console or the Kinesis console.

To access an activity stream from Kinesis using the RDS console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the Amazon RDS database instance on which you started an activity stream.
4. Choose **Configuration**.
5. Under **Database activity stream**, choose the link under **Kinesis stream**.
6. In the Kinesis console, choose **Monitoring** to begin observing the database activity.

To access an activity stream from Kinesis using the Kinesis console

1. Open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose your activity stream from the list of Kinesis streams.

   An activity stream's name includes the prefix `aws-rds-das-db-` followed by the resource ID of the database. The following is an example.

   ```
   aws-rds-das-db-NHVOV4PCLWHGF52NP
   ```

   To use the Amazon RDS console to find the resource ID for the database, choose your DB instance from the list of databases, and then choose the **Configuration** tab.

   To use the AWS CLI to find the full Kinesis stream name for an activity stream, use a describe-db-instances CLI request and note the value of `ActivityStreamKinesisStreamName` in the response.
3. Choose **Monitoring** to begin observing the database activity.

For more information about using Amazon Kinesis, see What Is Amazon Kinesis Data Streams?.

Audit log contents and examples for database activity streams

Monitored events are represented in the database activity stream as JSON strings. The structure consists of a JSON object containing a `DatabaseActivityMonitoringRecord`, which in turn contains a `databaseActivityEventList` array of activity events.

> **Note**
>
> For database activity streams, the `paramList` JSON array doesn't include null values from Hibernate applications.

Topics

- Examples of an audit log for an activity stream
- DatabaseActivityMonitoringRecords JSON object
- databaseActivityEvents JSON Object

Examples of an audit log for an activity stream

Following are sample decrypted JSON audit logs of activity event records.

Example Activity event record of a CONNECT SQL statement

The following activity event record shows a login with the use of a `CONNECT` SQL statement (`command`) by a JDBC Thin Client (`clientApplication`) for your Oracle DB.


```
{
    "class": "Standard",
    "clientApplication": "JDBC Thin Client",
    "command": "LOGON",
    "commandText": null,
    "dbid": "0123456789",
    "databaseName": "ORCL",
    "dbProtocol": "oracle",
    "dbUserName": "TEST",
    "endTime": null,
    "errorMessage": null,
    "exitCode": 0,
    "logTime": "2021-01-15 00:15:36.233787",
    "netProtocol": "tcp",
```

```
    "objectName": null,
    "objectType": null,
    "paramList": [],
    "pid": 17904,
    "remoteHost": "123.456.789.012",
    "remotePort": "25440",
    "rowCount": null,
    "serverHost": "987.654.321.098",
    "serverType": "oracle",
    "serverVersion": "19.0.0.0.ru-2020-01.rur-2020-01.r1.EE.3",
    "serviceName": "oracle-ee",
    "sessionId": 987654321,
    "startTime": null,
    "statementId": 1,
    "substatementId": null,
    "transactionId": "0000000000000000",
    "engineNativeAuditFields": {
        "UNIFIED_AUDIT_POLICIES": "TEST_POL_EVERYTHING",
        "FGA_POLICY_NAME": null,
        "DV_OBJECT_STATUS": null,
        "SYSTEM_PRIVILEGE_USED": "CREATE SESSION",
        "OLS_LABEL_COMPONENT_TYPE": null,
        "XS_SESSIONID": null,
        "ADDITIONAL_INFO": null,
        "INSTANCE_ID": 1,
        "DBID": 123456789
        "DV_COMMENT": null,
        "RMAN_SESSION_STAMP": null,
        "NEW_NAME": null,
        "DV_ACTION_NAME": null,
        "OLS_PROGRAM_UNIT_NAME": null,
        "OLS_STRING_LABEL": null,
        "RMAN_SESSION_RECID": null,
        "OBJECT_PRIVILEGES": null,
        "OLS_OLD_VALUE": null,
        "XS_TARGET_PRINCIPAL_NAME": null,
        "XS_NS_ATTRIBUTE": null,
        "XS_NS_NAME": null,
        "DBLINK_INFO": null,
        "AUTHENTICATION_TYPE": "(TYPE\u003d(DATABASE));(CLIENT ADDRESS\u003d((ADDRESS
\u003d(PROTOCOL\u003dtcp)(HOST\u003d205.251.233.183)(PORT\u003d25440))));",
        "OBJECT_EDITION": null,
        "OLS_PRIVILEGES_GRANTED": null,
        "EXCLUDED_USER": null,
```

```
        "DV_ACTION_OBJECT_NAME": null,
        "OLS_LABEL_COMPONENT_NAME": null,
        "EXCLUDED_SCHEMA": null,
        "DP_TEXT_PARAMETERS1": null,
        "XS_USER_NAME": null,
        "XS_ENABLED_ROLE": null,
        "XS_NS_ATTRIBUTE_NEW_VAL": null,
        "DIRECT_PATH_NUM_COLUMNS_LOADED": null,
        "AUDIT_OPTION": null,
        "DV_EXTENDED_ACTION_CODE": null,
        "XS_PACKAGE_NAME": null,
        "OLS_NEW_VALUE": null,
        "DV_RETURN_CODE": null,
        "XS_CALLBACK_EVENT_TYPE": null,
        "USERHOST": "a1b2c3d4e5f6.amazon.com",
        "GLOBAL_USERID": null,
        "CLIENT_IDENTIFIER": null,
        "RMAN_OPERATION": null,
        "TERMINAL": "unknown",
        "OS_USERNAME": "sumepate",
        "OLS_MAX_READ_LABEL": null,
        "XS_PROXY_USER_NAME": null,
        "XS_DATASEC_POLICY_NAME": null,
        "DV_FACTOR_CONTEXT": null,
        "OLS_MAX_WRITE_LABEL": null,
        "OLS_PARENT_GROUP_NAME": null,
        "EXCLUDED_OBJECT": null,
        "DV_RULE_SET_NAME": null,
        "EXTERNAL_USERID": null,
        "EXECUTION_ID": null,
        "ROLE": null,
        "PROXY_SESSIONID": 0,
        "DP_BOOLEAN_PARAMETERS1": null,
        "OLS_POLICY_NAME": null,
        "OLS_GRANTEE": null,
        "OLS_MIN_WRITE_LABEL": null,
        "APPLICATION_CONTEXTS": null,
        "XS_SCHEMA_NAME": null,
        "DV_GRANTEE": null,
        "XS_COOKIE": null,
        "DBPROXY_USERNAME": null,
        "DV_ACTION_CODE": null,
        "OLS_PRIVILEGES_USED": null,
        "RMAN_DEVICE_TYPE": null,
```

```
        "XS_NS_ATTRIBUTE_OLD_VAL": null,
        "TARGET_USER": null,
        "XS_ENTITY_TYPE": null,
        "ENTRY_ID": 1,
        "XS_PROCEDURE_NAME": null,
        "XS_INACTIVITY_TIMEOUT": null,
        "RMAN_OBJECT_TYPE": null,
        "SYSTEM_PRIVILEGE": null,
        "NEW_SCHEMA": null,
        "SCN": 5124715
    }
}
```

The following activity event record shows a login failure for your SQL Server DB.

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "LOGIN",
            "clientApplication": "Microsoft SQL Server Management Studio",
            "command": "LOGIN FAILED",
            "commandText": "Login failed for user 'test'. Reason: Password did not
 match that for the login provided. [CLIENT: local-machine]",
            "databaseName": "",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
            "errorMessage": null,
            "exitCode": 0,
            "logTime": "2022-10-06 21:34:42.7113072+00",
            "netProtocol": null,
            "objectName": "",
            "objectType": "LOGIN",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
```

```
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 0,
            "startTime": null,
            "statementId": "0x1eb0d1808d34a94b9d3dcf5432750f02",
            "substatementId": 1,
            "transactionId": "0",
            "type": "record",
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
                "target_database_principal_name": "",
                "server_principal_id": 0,
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "",
                "target_server_principal_name": "",
                "schema_name": "",
                "is_column_permission": false,
                "object_id": 0,
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "<action_info "xmlns=\"http://
schemas.microsoft.com/sqlserver/2008/sqlaudit_data\"><pooled_connection>0</
pooled_connection><error>0x00004818</error><state>8</state><address>local machine</
address><PasswordFirstNibbleHash>B</PasswordFirstNibbleHash></action_info>"-->,
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000000",
                "data_sensitivity_information": "",
                "session_server_principal_name": "",
                "connection_id": "98B4F537-0F82-49E3-AB08-B9D33B5893EF",
                "audit_schema_version": 1,
                "database_principal_id": 0,
                "server_principal_sid": null,
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```

> **Note**
>
> If a database activity stream isn't enabled, then the last field in the JSON document is `"engineNativeAuditFields": { }`.

Example Activity event record of a CREATE TABLE statement

The following example shows a `CREATE TABLE` event for your Oracle database.


```
{
    "class": "Standard",
    "clientApplication": "sqlplus@ip-12-34-5-678 (TNS V1-V3)",
    "command": "CREATE TABLE",
    "commandText": "CREATE TABLE persons(\n    person_id NUMBER GENERATED BY DEFAULT AS
 IDENTITY,\n    first_name VARCHAR2(50) NOT NULL,\n    last_name VARCHAR2(50) NOT NULL,
\n    PRIMARY KEY(person_id)\n)",
    "dbid": "0123456789",
    "databaseName": "ORCL",
    "dbProtocol": "oracle",
    "dbUserName": "TEST",
    "endTime": null,
    "errorMessage": null,
    "exitCode": 0,
    "logTime": "2021-01-15 00:22:49.535239",
    "netProtocol": "beq",
    "objectName": "PERSONS",
    "objectType": "TEST",
    "paramList": [],
    "pid": 17687,
    "remoteHost": "123.456.789.0",
    "remotePort": null,
    "rowCount": null,
    "serverHost": "987.654.321.01",
    "serverType": "oracle",
    "serverVersion": "19.0.0.0.ru-2020-01.rur-2020-01.r1.EE.3",
    "serviceName": "oracle-ee",
    "sessionId": 1234567890,
    "startTime": null,
    "statementId": 43,
    "substatementId": null,
    "transactionId": "090011007F0D0000",
    "engineNativeAuditFields": {
```

```
        "UNIFIED_AUDIT_POLICIES": "TEST_POL_EVERYTHING",
        "FGA_POLICY_NAME": null,
        "DV_OBJECT_STATUS": null,
        "SYSTEM_PRIVILEGE_USED": "CREATE SEQUENCE, CREATE TABLE",
        "OLS_LABEL_COMPONENT_TYPE": null,
        "XS_SESSIONID": null,
        "ADDITIONAL_INFO": null,
        "INSTANCE_ID": 1,
        "DV_COMMENT": null,
        "RMAN_SESSION_STAMP": null,
        "NEW_NAME": null,
        "DV_ACTION_NAME": null,
        "OLS_PROGRAM_UNIT_NAME": null,
        "OLS_STRING_LABEL": null,
        "RMAN_SESSION_RECID": null,
        "OBJECT_PRIVILEGES": null,
        "OLS_OLD_VALUE": null,
        "XS_TARGET_PRINCIPAL_NAME": null,
        "XS_NS_ATTRIBUTE": null,
        "XS_NS_NAME": null,
        "DBLINK_INFO": null,
        "AUTHENTICATION_TYPE": "(TYPE\u003d(DATABASE));(CLIENT ADDRESS\u003d((PROTOCOL
\u003dbeq)(HOST\u003d123.456.789.0)));",
        "OBJECT_EDITION": null,
        "OLS_PRIVILEGES_GRANTED": null,
        "EXCLUDED_USER": null,
        "DV_ACTION_OBJECT_NAME": null,
        "OLS_LABEL_COMPONENT_NAME": null,
        "EXCLUDED_SCHEMA": null,
        "DP_TEXT_PARAMETERS1": null,
        "XS_USER_NAME": null,
        "XS_ENABLED_ROLE": null,
        "XS_NS_ATTRIBUTE_NEW_VAL": null,
        "DIRECT_PATH_NUM_COLUMNS_LOADED": null,
        "AUDIT_OPTION": null,
        "DV_EXTENDED_ACTION_CODE": null,
        "XS_PACKAGE_NAME": null,
        "OLS_NEW_VALUE": null,
        "DV_RETURN_CODE": null,
        "XS_CALLBACK_EVENT_TYPE": null,
        "USERHOST": "ip-10-13-0-122",
        "GLOBAL_USERID": null,
        "CLIENT_IDENTIFIER": null,
        "RMAN_OPERATION": null,
```

```
        "TERMINAL": "pts/1",
        "OS_USERNAME": "rdsdb",
        "OLS_MAX_READ_LABEL": null,
        "XS_PROXY_USER_NAME": null,
        "XS_DATASEC_POLICY_NAME": null,
        "DV_FACTOR_CONTEXT": null,
        "OLS_MAX_WRITE_LABEL": null,
        "OLS_PARENT_GROUP_NAME": null,
        "EXCLUDED_OBJECT": null,
        "DV_RULE_SET_NAME": null,
        "EXTERNAL_USERID": null,
        "EXECUTION_ID": null,
        "ROLE": null,
        "PROXY_SESSIONID": 0,
        "DP_BOOLEAN_PARAMETERS1": null,
        "OLS_POLICY_NAME": null,
        "OLS_GRANTEE": null,
        "OLS_MIN_WRITE_LABEL": null,
        "APPLICATION_CONTEXTS": null,
        "XS_SCHEMA_NAME": null,
        "DV_GRANTEE": null,
        "XS_COOKIE": null,
        "DBPROXY_USERNAME": null,
        "DV_ACTION_CODE": null,
        "OLS_PRIVILEGES_USED": null,
        "RMAN_DEVICE_TYPE": null,
        "XS_NS_ATTRIBUTE_OLD_VAL": null,
        "TARGET_USER": null,
        "XS_ENTITY_TYPE": null,
        "ENTRY_ID": 12,
        "XS_PROCEDURE_NAME": null,
        "XS_INACTIVITY_TIMEOUT": null,
        "RMAN_OBJECT_TYPE": null,
        "SYSTEM_PRIVILEGE": null,
        "NEW_SCHEMA": null,
        "SCN": 5133083
    }
}
```


The following example shows a `CREATE TABLE` event for your SQL Server database.


```
{
    "type": "DatabaseActivityMonitoringRecord",
```

```
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "SCHEMA",
            "clientApplication": "Microsoft SQL Server Management Studio - Query",
            "command": "ALTER",
            "commandText": "Create table [testDB].[dbo].[TestTable2](\r\ntextA
 varchar(6000),\r\n    textB varchar(6000)\r\n)",
            "databaseName": "testDB",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
            "errorMessage": null,
            "exitCode": 1,
            "logTime": "2022-10-06 21:44:38.4120677+00",
            "netProtocol": null,
            "objectName": "dbo",
            "objectType": "SCHEMA",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 84,
            "startTime": null,
            "statementId": "0x5178d33d56e95e419558b9607158a5bd",
            "substatementId": 1,
            "transactionId": "4561864",
            "type": "record",
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
                "target_database_principal_name": "",
                "server_principal_id": 2,
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "dbo",
                "target_server_principal_name": "",
                "schema_name": "",
```

```
                "is_column_permission": false,
                "object_id": 1,
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "",
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000000",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "EE1FE3FD-EF2C-41FD-AF45-9051E0CD983A",
                "audit_schema_version": 1,
                "database_principal_id": 1,
                "server_principal_sid":
 "0x0105000000000005150000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```

Example Activity event record of a SELECT statement

The following example shows a SELECT event for your Oracle DB.

```
{
    "class": "Standard",
    "clientApplication": "sqlplus@ip-12-34-5-678 (TNS V1-V3)",
    "command": "SELECT",
    "commandText": "select count(*) from persons",
    "databaseName": "1234567890",
    "dbProtocol": "oracle",
    "dbUserName": "TEST",
    "endTime": null,
    "errorMessage": null,
    "exitCode": 0,
    "logTime": "2021-01-15 00:25:18.850375",
    "netProtocol": "beq",
    "objectName": "PERSONS",
    "objectType": "TEST",
    "paramList": [],
    "pid": 17687,
    "remoteHost": "123.456.789.0",
    "remotePort": null,
```

```
    "rowCount": null,
    "serverHost": "987.654.321.09",
    "serverType": "oracle",
    "serverVersion": "19.0.0.0.ru-2020-01.rur-2020-01.r1.EE.3",
    "serviceName": "oracle-ee",
    "sessionId": 1080639707,
    "startTime": null,
    "statementId": 44,
    "substatementId": null,
    "transactionId": null,
    "engineNativeAuditFields": {
        "UNIFIED_AUDIT_POLICIES": "TEST_POL_EVERYTHING",
        "FGA_POLICY_NAME": null,
        "DV_OBJECT_STATUS": null,
        "SYSTEM_PRIVILEGE_USED": null,
        "OLS_LABEL_COMPONENT_TYPE": null,
        "XS_SESSIONID": null,
        "ADDITIONAL_INFO": null,
        "INSTANCE_ID": 1,
        "DV_COMMENT": null,
        "RMAN_SESSION_STAMP": null,
        "NEW_NAME": null,
        "DV_ACTION_NAME": null,
        "OLS_PROGRAM_UNIT_NAME": null,
        "OLS_STRING_LABEL": null,
        "RMAN_SESSION_RECID": null,
        "OBJECT_PRIVILEGES": null,
        "OLS_OLD_VALUE": null,
        "XS_TARGET_PRINCIPAL_NAME": null,
        "XS_NS_ATTRIBUTE": null,
        "XS_NS_NAME": null,
        "DBLINK_INFO": null,
        "AUTHENTICATION_TYPE": "(TYPE\u003d(DATABASE));(CLIENT ADDRESS\u003d((PROTOCOL
\u003dbeq)(HOST\u003d123.456.789.0)));",
        "OBJECT_EDITION": null,
        "OLS_PRIVILEGES_GRANTED": null,
        "EXCLUDED_USER": null,
        "DV_ACTION_OBJECT_NAME": null,
        "OLS_LABEL_COMPONENT_NAME": null,
        "EXCLUDED_SCHEMA": null,
        "DP_TEXT_PARAMETERS1": null,
        "XS_USER_NAME": null,
        "XS_ENABLED_ROLE": null,
        "XS_NS_ATTRIBUTE_NEW_VAL": null,
```

```
        "DIRECT_PATH_NUM_COLUMNS_LOADED": null,
        "AUDIT_OPTION": null,
        "DV_EXTENDED_ACTION_CODE": null,
        "XS_PACKAGE_NAME": null,
        "OLS_NEW_VALUE": null,
        "DV_RETURN_CODE": null,
        "XS_CALLBACK_EVENT_TYPE": null,
        "USERHOST": "ip-12-34-5-678",
        "GLOBAL_USERID": null,
        "CLIENT_IDENTIFIER": null,
        "RMAN_OPERATION": null,
        "TERMINAL": "pts/1",
        "OS_USERNAME": "rdsdb",
        "OLS_MAX_READ_LABEL": null,
        "XS_PROXY_USER_NAME": null,
        "XS_DATASEC_POLICY_NAME": null,
        "DV_FACTOR_CONTEXT": null,
        "OLS_MAX_WRITE_LABEL": null,
        "OLS_PARENT_GROUP_NAME": null,
        "EXCLUDED_OBJECT": null,
        "DV_RULE_SET_NAME": null,
        "EXTERNAL_USERID": null,
        "EXECUTION_ID": null,
        "ROLE": null,
        "PROXY_SESSIONID": 0,
        "DP_BOOLEAN_PARAMETERS1": null,
        "OLS_POLICY_NAME": null,
        "OLS_GRANTEE": null,
        "OLS_MIN_WRITE_LABEL": null,
        "APPLICATION_CONTEXTS": null,
        "XS_SCHEMA_NAME": null,
        "DV_GRANTEE": null,
        "XS_COOKIE": null,
        "DBPROXY_USERNAME": null,
        "DV_ACTION_CODE": null,
        "OLS_PRIVILEGES_USED": null,
        "RMAN_DEVICE_TYPE": null,
        "XS_NS_ATTRIBUTE_OLD_VAL": null,
        "TARGET_USER": null,
        "XS_ENTITY_TYPE": null,
        "ENTRY_ID": 13,
        "XS_PROCEDURE_NAME": null,
        "XS_INACTIVITY_TIMEOUT": null,
        "RMAN_OBJECT_TYPE": null,
```

```
        "SYSTEM_PRIVILEGE": null,
        "NEW_SCHEMA": null,
        "SCN": 5136972
    }
}
```

The following example shows a SELECT event for your SQL Server DB.

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "TABLE",
            "clientApplication": "Microsoft SQL Server Management Studio - Query",
            "command": "SELECT",
            "commandText": "select * from [testDB].[dbo].[TestTable]",
            "databaseName": "testDB",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
            "errorMessage": null,
            "exitCode": 1,
            "logTime": "2022-10-06 21:24:59.9422268+00",
            "netProtocol": null,
            "objectName": "TestTable",
            "objectType": "TABLE",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 62,
            "startTime": null,
            "statementId": "0x03baed90412f564fad640ebe51f89b99",
            "substatementId": 1,
            "transactionId": "4532935",
            "type": "record",
```

```
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
                "target_database_principal_name": "",
                "server_principal_id": 2,
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "dbo",
                "target_server_principal_name": "",
                "schema_name": "dbo",
                "is_column_permission": true,
                "object_id": 581577110,
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "",
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000001",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "AD3A5084-FB83-45C1-8334-E923459A8109",
                "audit_schema_version": 1,
                "database_principal_id": 1,
                "server_principal_sid":
 "0x0105000000000005150000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```


DatabaseActivityMonitoringRecords JSON object

The database activity event records are in a JSON object that contains the following information.

JSON Field	Data Type	Description
`type`	string	The type of JSON record. The value is `DatabaseActivityMonitoringRecords`.

JSON Field	Data Type	Description
`version`	string	The version of the database activity monitoring records. Oracle DB uses version 1.3 and SQL Server uses version 1.4. These engine versions introduce the `engineNativeAuditFields` JSON object.
databaseActivityEvents	string	A JSON object that contains the activity events.
key	string	An encryption key that you use to decrypt the databaseActivityEventList JSON array

databaseActivityEvents JSON Object

The `databaseActivityEvents` JSON object contains the following information.

Top-level fields in JSON record

Each event in the audit log is wrapped inside a record in JSON format. This record contains the following fields.

type

This field always has the value `DatabaseActivityMonitoringRecords`.

version

This field represents the version of the database activity stream data protocol or contract. It defines which fields are available.

databaseActivityEvents

An encrypted string representing one or more activity events. It's represented as a base64 byte array. When you decrypt the string, the result is a record in JSON format with fields as shown in the examples in this section.

key

The encrypted data key used to encrypt the `databaseActivityEvents` string. This is the same AWS KMS key that you provided when you started the database activity stream.

The following example shows the format of this record.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.3",
  "databaseActivityEvents":"encrypted audit records",
  "key":"encrypted key"
}
```

```
          "type":"DatabaseActivityMonitoringRecords",
          "version":"1.4",
          "databaseActivityEvents":"encrypted audit records",
          "key":"encrypted key"
```

Take the following steps to decrypt the contents of the `databaseActivityEvents` field:

1. Decrypt the value in the `key` JSON field using the KMS key you provided when starting database activity stream. Doing so returns the data encryption key in clear text.
2. Base64-decode the value in the `databaseActivityEvents` JSON field to obtain the ciphertext, in binary format, of the audit payload.
3. Decrypt the binary ciphertext with the data encryption key that you decoded in the first step.
4. Decompress the decrypted payload.
   - The encrypted payload is in the `databaseActivityEvents` field.
   - The `databaseActivityEventList` field contains an array of audit records. The `type` fields in the array can be `record` or `heartbeat`.

The audit log activity event record is a JSON object that contains the following information.

JSON Field	Data Type	Description
`type`	string	The type of JSON record. The value is `DatabaseActivityMonitoringRecord` .
`instanceId`	string	The DB instance resource identifier. It corresponds to the DB instance attribute `DbiResourceId` .
databaseActivityEventList JSON array	string	An array of activity audit records or heartbeat messages.

databaseActivityEventList JSON array for database activity streams

The audit log payload is an encrypted `databaseActivityEventList` JSON array. The following table lists alphabetically the fields for each activity event in the decrypted `DatabaseActivityEventList` array of an audit log.

When unified auditing is enabled in Oracle Database, the audit records are populated in this new audit trail. The `UNIFIED_AUDIT_TRAIL` view displays audit records in tabular form by retrieving the audit records from the audit trail. When you start a database activity stream, a column in `UNIFIED_AUDIT_TRAIL` maps to a field in the `databaseActivityEventList` array.

Important

The event structure is subject to change. Amazon RDS might add new fields to activity events in the future. In applications that parse the JSON data, make sure that your code can ignore or take appropriate actions for unknown field names.

databaseActivityEventList fields for Amazon RDS for Oracle

The following are `databaseActivityEventList` fields for Amazon RDS for Oracle.

Field	Data Type	Source	Description
class	string	AUDIT_TYPE column in UNIFIED_AUDIT_TRAIL	The class of activity event. This corresponds to the AUDIT_TYPE column in the UNIFIED_AUDIT_TRAIL view. Valid values for Amazon RDS for Oracle are the following: • Standard • FineGrainedAudit • XS • Database Vault • Label Security • RMAN_AUDIT • Datapump • Direct path API For more information, see UNIFIED_AUDIT_TRAIL in the Oracle documentation.
clientApplication	string	CLIENT_PROGRAM_NAME in UNIFIED_AUDIT_TRAIL	The application the client used to connect as reported by the client. The client doesn't have to provide this information, so the value can be null. A sample value is JDBC Thin Client.

Field	Data Type	Source	Description
command	string	ACTION_NAME column in UNIFIED_AUDIT_TRAIL	Name of the action executed by the user. To understand the complete action, read both the command name and the AUDIT_TYPE value. A sample value is ALTER DATABASE.
commandText	string	SQL_TEXT column in UNIFIED_AUDIT_TRAIL	The SQL statement associated with the event. A sample value is ALTER DATABASE BEGIN BACKUP.
databaseName	string	NAME column in V$DATABASE	The name of the database.
dbid	numb	DBID column in UNIFIED_AUDIT_TRAIL	Numerical identifier for the database. A sample value is 1559204751 .
dbProtocol	string	N/A	The database protocol. In this beta, the value is oracle.
dbUserName	string	DBUSERNAME column in UNIFIED_AUDIT_TRAIL	Name of the database user whose actions were audited. A sample value is RDSADMIN.
endTime	string	N/A	This field isn't used for RDS for Oracle and is always null.

Field	Data Type	Source	Description
engineNativeAuditFields	objec	UNIFIED_AUDIT_TRAIL	By default, this object is empty. When you start the activity stream with the --engine-native-audit-fields-included option, this object includes the following columns and their values: ADDITIONAL_INFO APPLICATION_CONTEXTS AUDIT_OPTION AUTHENTICATION_TYPE CLIENT_IDENTIFIER CURRENT_USER DBLINK_INFO DBPROXY_USERNAME DIRECT_PATH_NUM_COLUMNS_LOADED DP_BOOLEAN_PARAMETERS1 DP_TEXT_PARAMETERS1 DV_ACTION_CODE DV_ACTION_NAME DV_ACTION_OBJECT_NAME DV_COMMENT DV_EXTENDED_ACTION_CODE DV_FACTOR_CONTEXT DV_GRANTEE DV_OBJECT_STATUS DV_RETURN_CODE

Field	Data Type	Source	Description
			DV_RULE_SET_NAME ENTRY_ID EXCLUDED_OBJECT EXCLUDED_SCHEMA EXCLUDED_USER EXECUTION_ID EXTERNAL_USERID FGA_POLICY_NAME GLOBAL_USERID INSTANCE_ID KSACL_SER VICE_NAME KSACL_SOURCE_LOCA TION KSACL_USER_NAME NEW_NAME NEW_SCHEMA OBJECT_EDITION OBJECT_PRIVILEGES OLS_GRANTEE OLS_LABEL_COM PONENT_NAME OLS_LABEL_COMPO NENT_TYPE OLS_MAX_READ_LABEL OLS_MAX_WRITE_LA BEL OLS_MIN_WRITE_ LABEL OLS_NEW_VALUE OLS_OLD_VALUE OLS_PARENT_ GROUP_NAME OLS_POLICY_NAME OLS_PRIVILEGES_GRA NTED OLS_PRIVILEGE S_USED OLS_PROGRAM _UNIT_NAME OLS_STRING_LABEL

Field	Data Type	Source	Description
			OS_USERNAME PROTOCOL_ACTIO N_NAME PROTOCOL_MESSAGE PROTOCOL_RET URN_CODE PROTOCOL_SESSION_I D PROTOCOL_USERHOST PROXY_SESSIONID RLS_INFO RMAN_DEVICE_TYPE RMAN_OBJECT_TYPE RMAN_OPERATION RMAN_SESSION_RECID RMAN_SESSION_STAMP ROLE SCN SYSTEM_PRIVILEGE SYSTEM_PRIVIL EGE_USED TARGET_USER TERMINAL UNIFIED_AUDIT_P OLICIES USERHOST XS_CALLBAC K_EVENT_TYPE XS_COOKIE XS_DATASEC_PO LICY_NAME XS_ENABLED_ROLE XS_ENTITY_TYPE XS_INACTIVITY _TIMEOUT XS_NS_ATTRIBUTE XS_NS_ATTRI BUTE_NEW_VAL XS_NS_ATTRIBUT E_OLD_VAL XS_NS_NAME

Field	Data Type	Source	Description
			XS_PACKAGE_NAME XS_PROCEDURE_NAME XS_PROXY_USER_NAME XS_SCHEMA_NAME XS_SESSIONID XS_TARGET_PRINC IPAL_NAME XS_USER_NAME For more informati on, see UNIFIED_A UDIT_TRAIL in the Oracle Database documenta tion.
errorMessage	string	N/A	This field isn't used for RDS for Oracle and is always null.
exitCode	numb	RETURN_CODE column in UNIFIED_AUDIT_TRAIL	Oracle Database error code generated by the action. If the action succeeded, the value is 0.
logTime	string	EVENT_TIMESTAMP_UT C column in UNIFIED_A UDIT_TRAIL	Timestamp of the creation of the audit trail entry. A sample value is 2020-11-27 06:56:14.981404 .
netProtocol	string	AUTHENTICATION_TYP E column in UNIFIED_A UDIT_TRAIL	The network communica tion protocol. A sample value is TCP.

Field	Data Type	Source	Description
objectName	string	OBJECT_NAME column in UNIFIED_AUDIT_TRAIL	The name of the object affected by the action. A sample value is employees .
objectType	string	OBJECT_SCHEMA column in UNIFIED_AUDIT_TRAIL	The schema name of object affected by the action. A sample value is hr.
paramList	list	SQL_BINDS column in UNIFIED_AUDIT_TRAIL	The list of bind variables , if any, associated with SQL_TEXT. A sample value is parameter _1,parameter_2 .
pid	numb	OS_PROCESS column in UNIFIED_AUDIT_TRAIL	Operating system process identifier of the Oracle database process. A sample value is 22396.
remoteHost	string	AUTHENTICATION_TYP E column in UNIFIED_A UDIT_TRAIL	Either the client IP address or name of the host from which the session was spawned. A sample value is 123.456.789.123 .
remotePort	string	AUTHENTICATION_TYP E column in UNIFIED_A UDIT_TRAIL	The client port number. A typical value in Oracle Database environments is 1521.

Field	Data Type	Source	Description
rowCount	numb	N/A	This field isn't used for RDS for Oracle and is always null.
serverHost	string	Database host	The IP address of the database server host. A sample value is 123.456.789.123 .
serverType	string	N/A	The database server type. The value is always ORACLE.
serverVersion	string	Database host	The Amazon RDS for Oracle version, Release Update (RU), and Release Update Revision (RUR). A sample value is 19.0.0.0.ru-2020-01.rur-2020-01.r1.EE.3 .
serviceName	string	Database host	The name of the service. A sample value is oracle-ee .
sessionId	numb	SESSIONID column in UNIFIED_AUDIT_TRAIL	The session identifier of the audit. An example is 1894327130 .
startTime	string	N/A	This field isn't used for RDS for Oracle and is always null.

Field	Data Type	Source	Description
statementId	numb	STATEMENT_ID column in UNIFIED_AUDIT_TRAIL	Numeric ID for each statement run. A statement can cause many actions. A sample value is 142197.
substatementId	N/A	N/A	This field isn't used for RDS for Oracle and is always null.
transactionId	string	TRANSACTION_ID column in UNIFIED_AUDIT_TRAIL	The identifier of the transaction in which the object is modified. A sample value is 02000800D5030000 .

databaseActivityEventList fields for Amazon RDS for SQL Server

The following are databaseActivityEventList fields for Amazon RDS for SQL Server.

Field	Data Type	Source	Description
class	string	sys.fn_get_audit_file.class_type mapped to sys.dm_audit_class_type_map.class_type_desc	The class of activity event. For more information, see SQL Server Audit (Database Engine) in the Microsoft documentation.
clientApplication	string	sys.fn_get_audit_file.application_name	The application that the client connects as reported by the client (SQL Server version 14 and higher). This field is null in SQL Server version 13.

Field	Data Type	Source	Description
command	string	sys.fn_get_audit_file.action_id mapped to sys.dm_audit_actions.name	The general category of the SQL statement. The value for this field depends on the value of the class.
commandText	string	sys.fn_get_audit_file.statement	This field indicates the SQL statement.
databaseName	string	sys.fn_get_audit_file.database_name	Name of the database.
dbProtocol	string	N/A	The database protocol. This value is SQLSERVER .
dbUserName	string	sys.fn_get_audit_file.server_principal_name	The database user for the client authentication.
endTime	string	N/A	This field isn't used by Amazon RDS for SQL Server and the value is null.
engineNativeAuditFields	object	Each field in sys.fn_get_audit_file that is not listed in this column.	By default, this object is empty. When you start the activity stream with the --engine-native-audit-fields-included option, this object includes other native engine audit fields, which are not returned by this JSON map.
errorMessage	string	N/A	This field isn't used by Amazon RDS for SQL Server and the value is null.

Field	Data Type	Source	Description
exitCode	integer	sys.fn_get_audit_file.succeeded	Indicates whether the action that started the event succeeded. This field can't be null. For all the events except login events, this field reports whether the permission check succeeded or failed, but not whether the operation succeeded or failed. Values include: • 0 – Fail • 1 – Success
logTime	string	sys.fn_get_audit_file.event_time	The event timestamp that is recorded by the SQL Server.
netProtocol	string	N/A	This field isn't used by Amazon RDS for SQL Server and the value is null.
objectName	string	sys.fn_get_audit_file.object_name	The name of the database object if the SQL statement is operating on an object.
objectType	string	sys.fn_get_audit_file.class_type mapped to sys.dm_audit_class_type_map.class_type_desc	The database object type if the SQL statement is operating on an object type.
paramList	string	N/A	This field isn't used by Amazon RDS for SQL Server and the value is null.

Field	Data Type	Source	Description
pid	integer	N/A	This field isn't used by Amazon RDS for SQL Server and the value is null.
remoteHost	string	sys.fn_get_audit_file.client_ip	The IP address or hostname of the client that issued the SQL statement (SQL Server version 14 and higher). This field is null in SQL Server version 13.
remotePort	integer	N/A	This field isn't used by Amazon RDS for SQL Server and the value is null.
rowCount	integer	sys.fn_get_audit_file.affected_rows	The number of table rows affected by the SQL statement (SQL Server version 14 and higher). This field is in SQL Server version 13.
serverHost	string	Database Host	The IP address of the host database server.
serverType	string	N/A	The database server type. The value is SQLSERVER .
serverVersion	string	Database Host	The database server version, for example, 15.00.4073.23.v1.R1 for SQL Server 2017.
serviceName	string	Database Host	The name of the service. An example value is sqlserver-ee.

Field	Data Type	Source	Description
sessionId	integer	sys.fn_get_audit_file.session_id	Unique identifier of the session.
startTime	string	N/A	This field isn't used by Amazon RDS for SQL Server and the value is null.
statementId	string	sys.fn_get_audit_file.sequence_group_id	A unique identifier for the client's SQL statement. The identifier is different for each event that is generated. A sample value is 0x38eaf4156267184094bb82071aaab644 .
substatementId	integer	sys.fn_get_audit_file.sequence_number	An identifier to determine the sequence number for a statement. This identifier helps when large records are split into multiple records.
transactionId	integer	sys.fn_get_audit_file.transaction_id	An identifier of a transaction. If there aren't any active transactions, the value is zero.
type	string	Database activity stream generated	The type of event. The values are record or heartbeat .

Processing a database activity stream using the AWS SDK

You can programmatically process an activity stream by using the AWS SDK. The following are fully functioning Java and Python examples of using Database Activity Streams records for instance based enablement.

Java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import
 com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
```

```
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[instance-external-
resource-id]"; // aws-rds-das-db-ABCD123456
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
 application name for dynamo table generation that holds kinesis shard tracking
    private static final String AWS_ACCESS_KEY =
 "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
    private static final String AWS_SECRET_KEY =
 "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String RESOURCE_ID = "[external-resource-id]"; // db-
ABCD123456
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
 BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
 AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
            .withRegion(REGION_NAME)
            .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
```

```
        String command;
        String commandText;
        String databaseName;
        String dbProtocol;
        String dbUserName;
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
        String objectName;
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
        String serverHost;
        String serverType;
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId; // note that clusterId will contain an empty string on RDS
Oracle and RDS SQL Server
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }
```

```
   static class RecordProcessor implements IRecordProcessor {

       private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
       private static final int PROCESSING_RETRIES_MAX = 10;
       private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
       private static final Gson GSON = new
GsonBuilder().serializeNulls().create();

       private static final Cipher CIPHER;
       static {
           Security.insertProviderAt(new BouncyCastleProvider(), 1);
           try {
               CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
           } catch (NoSuchAlgorithmException | NoSuchPaddingException |
NoSuchProviderException e) {
               throw new ExceptionInInitializerError(e);
           }
       }

       private long nextCheckpointTimeInMillis;

       @Override
       public void initialize(String shardId) {
       }

       @Override
       public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
           for (final Record record : records) {
               processSingleBlob(record.getData());
           }

           if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
               checkpoint(checkpointer);
               nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
           }
       }

       @Override
       public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason) {
           if (reason == ShutdownReason.TERMINATE) {
               checkpoint(checkpointer);
```

```
            }
        }

        private void processSingleBlob(final ByteBuffer bytes) {
            try {
                // JSON $Activity
                final Activity activity = GSON.fromJson(new String(bytes.array(),
StandardCharsets.UTF_8), Activity.class);

                // Base64.Decode
                final byte[] decoded =
Base64.decode(activity.databaseActivityEvents);
                final byte[] decodedDataKey = Base64.decode(activity.key);

                Map<String, String> context = new HashMap<>();
                context.put("aws:rds:db-id", RESOURCE_ID);

                // Decrypt
                final DecryptRequest decryptRequest = new DecryptRequest()

.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
                final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
                final byte[] decrypted = decrypt(decoded,
getByteArray(decryptResult.getPlaintext()));

                // GZip Decompress
                final byte[] decompressed = decompress(decrypted);
                // JSON $ActivityRecords
                final ActivityRecords activityRecords = GSON.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

                // Iterate throught $ActivityEvents
                for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
                    System.out.println(GSON.toJson(event));
                }
            } catch (Exception e) {
                // Handle error.
                e.printStackTrace();
            }
        }

        private static byte[] decompress(final byte[] src) throws IOException {
```

```
            ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(src);
            GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
            return IOUtils.toByteArray(gzipInputStream);
        }

        private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
            for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {
                try {
                    checkpointer.checkpoint();
                    break;
                } catch (ShutdownException se) {
                    // Ignore checkpoint if the processor instance has been shutdown
(fail over).
                    System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
                    break;
                } catch (ThrottlingException e) {
                    // Backoff and re-attempt checkpoint upon transient failures
                    if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                        System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                        break;
                    } else {
                        System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
                    }
                } catch (InvalidStateException e) {
                    // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
                    System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
                    break;
                }
                try {
                    Thread.sleep(BACKOFF_TIME_IN_MILLIS);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted sleep" + e);
                }
            }
        }
    }
```

```
   private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
       // Create a JCE master key provider using the random key and an AES-GCM
encryption algorithm
       final JceMasterKey masterKey = JceMasterKey.getInstance(new
SecretKeySpec(decodedDataKey, "AES"),
               "BC", "DataKey", "AES/GCM/NoPadding");
       try (final CryptoInputStream<JceMasterKey> decryptingStream =
CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
            final ByteArrayOutputStream out = new ByteArrayOutputStream()) {
           IOUtils.copy(decryptingStream, out);
           return out.toByteArray();
       }
   }

   public static void main(String[] args) throws Exception {
       final String workerId = InetAddress.getLocalHost().getCanonicalHostName() +
":" + UUID.randomUUID();
       final KinesisClientLibConfiguration kinesisClientLibConfiguration =
               new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
CREDENTIALS_PROVIDER, workerId);

kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
       kinesisClientLibConfiguration.withRegionName(REGION_NAME);
       final Worker worker = new Builder()
               .recordProcessorFactory(new RecordProcessorFactory())
               .config(kinesisClientLibConfiguration)
               .build();

       System.out.printf("Running %s to process stream %s as worker %s...\n",
APPLICATION_NAME, STREAM_NAME, workerId);

       try {
           worker.run();
       } catch (Throwable t) {
           System.err.println("Caught throwable while processing data.");
           t.printStackTrace();
           System.exit(1);
       }
       System.exit(0);
   }

   private static byte[] getByteArray(final ByteBuffer b) {
       byte[] byteArray = new byte[b.remaining()];
```

```
        b.get(byteArray);
        return byteArray;
    }
}
```

Python

```
import base64
import json
import zlib
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>'                    # us-east-1
RESOURCE_ID = '<external-resource-id>'      # db-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID  # aws-rds-das-db-ABCD123456

enc_client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
 WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                                        wrapping_key=plain_key,
 wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key


def decrypt_payload(payload, data_key):
```

```
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,

 materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManag
    return decrypted_plaintext


def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)


def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)

    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
 ShardId=shard['ShardId'],

 ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
 base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result =
 kms.decrypt(CiphertextBlob=data_key_decoded,

 EncryptionContext={'aws:rds:db-id': RESOURCE_ID})
                print (decrypt_decompress(payload_decoded,
 data_key_decrypt_result['Plaintext']))
```

```
            if 'NextShardIterator' in response:
                next_shard_iters.append(response['NextShardIterator'])
        shard_iters = next_shard_iters


if __name__ == '__main__':
    main()
```

IAM policy examples for database activity streams

Any user with appropriate AWS Identity and Access Management (IAM) role privileges for database activity streams can create, start, stop, and modify the activity stream settings for a DB instance. These actions are included in the audit log of the stream. For best compliance practices, we recommend that you don't provide these privileges to DBAs.

You set access to database activity streams using IAM policies. For more information about Amazon RDS authentication, see Identity and access management for Amazon RDS. For more information about creating IAM policies, see Creating and using an IAM policy for IAM database access.

Example Policy to allow configuring database activity streams

To give users fine-grained access to modify activity streams, use the service-specific operation context keys `rds:StartActivityStream` and `rds:StopActivityStream` in an IAM policy. The following IAM policy example allows a user or role to configure activity streams.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ConfigureActivityStreams",
            "Effect": "Allow",
            "Action": [
                "rds:StartActivityStream",
                "rds:StopActivityStream"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

Example Policy to allow starting database activity streams

The following IAM policy example allows a user or role to start activity streams.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStartActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
    ]
}
```

Example Policy to allow stopping database activity streams

The following IAM policy example allows a user or role to stop activity streams.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStopActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example Policy to deny starting database activity streams

The following IAM policy example prevents a user or role from starting activity streams.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStartActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example Policy to deny stopping database activity streams

The following IAM policy example prevents a user or role from stopping activity streams.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStopActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
    ]
}
```

Monitoring threats with Amazon GuardDuty RDS Protection

Amazon GuardDuty is a threat detection service that helps protect your accounts, containers, workloads, and the data within your AWS environment. Using machine learning (ML) models, and anomaly and threat detection capabilities, GuardDuty continuously monitors different log sources and runtime activity to identify and prioritize potential security risks and malicious activities in your environment.

GuardDuty RDS Protection analyzes and profiles login events for potential access threats to your Amazon RDS databases. When you turn on RDS Protection, GuardDuty consumes RDS login events from your RDS databases. RDS Protection monitors these events and profiles them for potential insider threats or external actors.

For more information about enabling GuardDuty RDS Protection, see GuardDuty RDS Protection in the *Amazon GuardDuty User Guide*.

When RDS Protection detects a potential threat, such as an unusual pattern in successful or failed login attempts, GuardDuty generates a new finding with details about the potentially compromised database. You can view the finding details in the finding summary section in the Amazon GuardDuty console. The finding details vary based on the finding type. The primary details, resource type and resource role, determine the kind of information available for any finding. For more information about the commonly available details for findings and the finding types, see Finding details and GuardDuty RDS Protection finding types respectively in the *Amazon GuardDuty User Guide*.

You can turn the RDS Protection feature on or off for any AWS account in any AWS Region where this feature is available. When RDS Protection isn't enabled, GuardDuty doesn't detect potentially compromised RDS databases or provide details of the compromise.

An existing GuardDuty account can enable RDS Protection with a 30-day trial period. For a new GuardDuty account, RDS Protection is already enabled and included in the 30-day free trial period. For more information, see Estimating GuardDuty cost in the *Amazon GuardDuty User Guide*.

For information about the AWS Regions where GuardDuty doesn't yet support RDS Protection, see Region-specific feature availability in the *Amazon GuardDuty User Guide*.

For information about the RDS database versions that GuardDuty RDS Protection supports, see Supported Amazon Aurora, Amazon RDS, and Aurora Limitless databases in the *Amazon GuardDuty User Guide*.

Amazon RDS Custom

Amazon RDS Custom automates database administration tasks and operations. RDS Custom makes it possible for you as a database administrator to access and customize your database environment and operating system. With RDS Custom, you can customize to meet the requirements of legacy, custom, and packaged applications.

For the latest webinars and blogs about RDS Custom, see Amazon RDS Custom resources.

Topics

- Addressing the challenge of database customization
- Management model and benefits for Amazon RDS Custom
- Amazon RDS Custom architecture
- Security in Amazon RDS Custom
- Working with RDS Custom for Oracle
- Working with RDS Custom for SQL Server

Addressing the challenge of database customization

Amazon RDS Custom brings the benefits of Amazon RDS to a market that can't easily move to a fully managed service because of customizations that are required with third-party applications. Amazon RDS Custom saves administrative time, is durable, and scales with your business.

If you need the entire database and operating system to be fully managed by AWS, we recommend Amazon RDS. If you need administrative rights to the database and underlying operating system to make dependent applications available, Amazon RDS Custom is the better choice. If you want full management responsibility and simply need a managed compute service, the best option is self-managing your commercial databases on Amazon EC2.

To deliver a managed service experience, Amazon RDS doesn't let you access the underlying host. Amazon RDS also restricts access to some procedures and objects that require high-level privileges. However, for some applications, you might need to perform operations as a privileged operating system (OS) user.

For example, you might need to do the following:

- Install custom database and OS patches and packages.

- Configure specific database settings.
- Configure file systems to share files directly with their applications.

Previously, if you needed to customize your application, you had to deploy your database on-premises or on Amazon EC2. In this case, you bear most or all of the responsibility for database management, as summarized in the following table.

Feature	On-premises responsibility	Amazon EC2 responsibility	Amazon RDS responsibility
Application optimization	Customer	Customer	Customer
Scaling	Customer	Customer	AWS
High availability	Customer	Customer	AWS
Database backups	Customer	Customer	AWS
Database software patching	Customer	Customer	AWS
Database software install	Customer	Customer	AWS
OS patching	Customer	Customer	AWS
OS installation	Customer	Customer	AWS
Server maintenance	Customer	AWS	AWS
Hardware lifecycle	Customer	AWS	AWS
Power, network, and cooling	Customer	AWS	AWS

When you manage database software yourself, you gain more control, but you're also more prone to user errors. For example, when you make changes manually, you might accidentally

cause application downtime. You might spend hours checking every change to identify and fix an issue. Ideally, you want a managed database service that automates common DBA tasks, but also supports privileged access to the database and underlying operating system.

Management model and benefits for Amazon RDS Custom

Amazon RDS Custom is a managed database service for legacy, custom, and packaged applications that require access to the underlying operating system and database environment. RDS Custom automates setup, operation, and scaling of databases in the AWS Cloud while granting you access to the database and underlying operating system. With this access, you can configure settings, install patches, and enable native features to meet the dependent application's requirements. With RDS Custom, you can run your database workload using the AWS Management Console or the AWS CLI.

RDS Custom supports only the Oracle Database and Microsoft SQL Server DB engines.

Topics

- Shared responsibility model in RDS Custom
- Support perimeter and unsupported configurations in RDS Custom
- Key benefits of RDS Custom

Shared responsibility model in RDS Custom

With RDS Custom, you use the managed features of Amazon RDS, but you manage the host and customize the OS as you do in Amazon EC2. You take on additional database management responsibilities beyond what you do in Amazon RDS. The result is that you have more control over database and DB instance management than you do in Amazon RDS, while still benefiting from RDS automation.

Shared responsibility means the following:

1. You own part of the process when using an RDS Custom feature.

   For example, in RDS Custom for Oracle, you control which Oracle database patches to use and when to apply them to your DB instances.
2. You are responsible for making sure that any customizations to RDS Custom features work correctly.

To help protect against invalid customization, RDS Custom has automation software that runs outside of your DB instance. If your underlying Amazon EC2 instance becomes impaired, RDS Custom attempts to resolve these problems automatically by either rebooting or replacing the EC2 instance. The only user-visible change is a new IP address. For more information, see Amazon RDS Custom host replacement.

The following table details the shared responsibility model for different features of RDS Custom.

Feature	Amazon EC2 responsibility	Amazon RDS responsibility	RDS Custom for Oracle responsibility	RDS Custom for SQL Server responsibility
Application optimization	Customer	Customer	Customer	Customer
Scaling	Customer	AWS	Shared	Shared
High availability	Customer	AWS	AWS	AWS
Database backups	Customer	AWS	Shared	AWS
Database software patching	Customer	AWS	Shared	AWS for RPEV, Customer for CEV[1]
Database software install	Customer	AWS	Shared	AWS for RPEV, Customer for CEV[1]
OS patching	Customer	AWS	Customer	AWS for RPEV, Customer for CEV[1]
OS installation	Customer	AWS	Shared	AWS
Server maintenance	AWS	AWS	AWS	AWS

Feature	Amazon EC2 responsibility	Amazon RDS responsibility	RDS Custom for Oracle responsibility	RDS Custom for SQL Server responsibility
Hardware lifecycle	AWS	AWS	AWS	AWS
Power, network, and cooling	AWS	AWS	AWS	AWS

[1] A custom engine version (CEV) is a binary volume snapshot of a database version and Amazon Machine Image (AMI). An RDS provided engine version (RPEV) is the default Amazon Machine Image (AMI) and Microsoft SQL Server installation.

You can create an RDS Custom DB instance using Microsoft SQL Server. In this case:

- You can choose from two licensing models: License Included (LI) and Bring Your Own Media (BYOM).
- With LI, you don't need to purchase SQL Server licenses separately. AWS holds the license for the SQL Server database software.
- With BYOM, you provide and install your own Microsoft SQL Server binaries and licensing.

You can create an RDS Custom DB instance using Oracle Database. In this case, you do the following:

- Manage your own media.

  When using RDS Custom, you upload your own database installation files and patches. You create a custom engine version (CEV) from these files. Then you can create an RDS Custom DB instance by using this CEV.
- Manage your own licenses.

  You bring your own Oracle Database licenses and manage licenses by yourself.

Support perimeter and unsupported configurations in RDS Custom

RDS Custom provides a monitoring capability called the *support perimeter*. This feature ensures that your host and database environment are configured correctly. If you make a change that causes your DB instance to go outside the support perimeter, RDS Custom changes the instance status to `unsupported-configuration` until you manually fix the configuration problems. For more information, see RDS Custom support perimeter.

Key benefits of RDS Custom

With RDS Custom, you can do the following:

- Automate many of the same administrative tasks as Amazon RDS, including the following:
  - Lifecycle management of databases
  - Automated backups and point-in-time recovery (PITR)
  - Monitoring the health of RDS Custom DB instances and observing changes to the infrastructure, operating system, and database processes.
  - Notification or taking action to fix issues depending on disruption to the DB instance
- Install third-party applications.

  You can install software to run custom applications and agents. Because you have privileged access to the host, you can modify file systems to support legacy applications.
- Install custom patches.

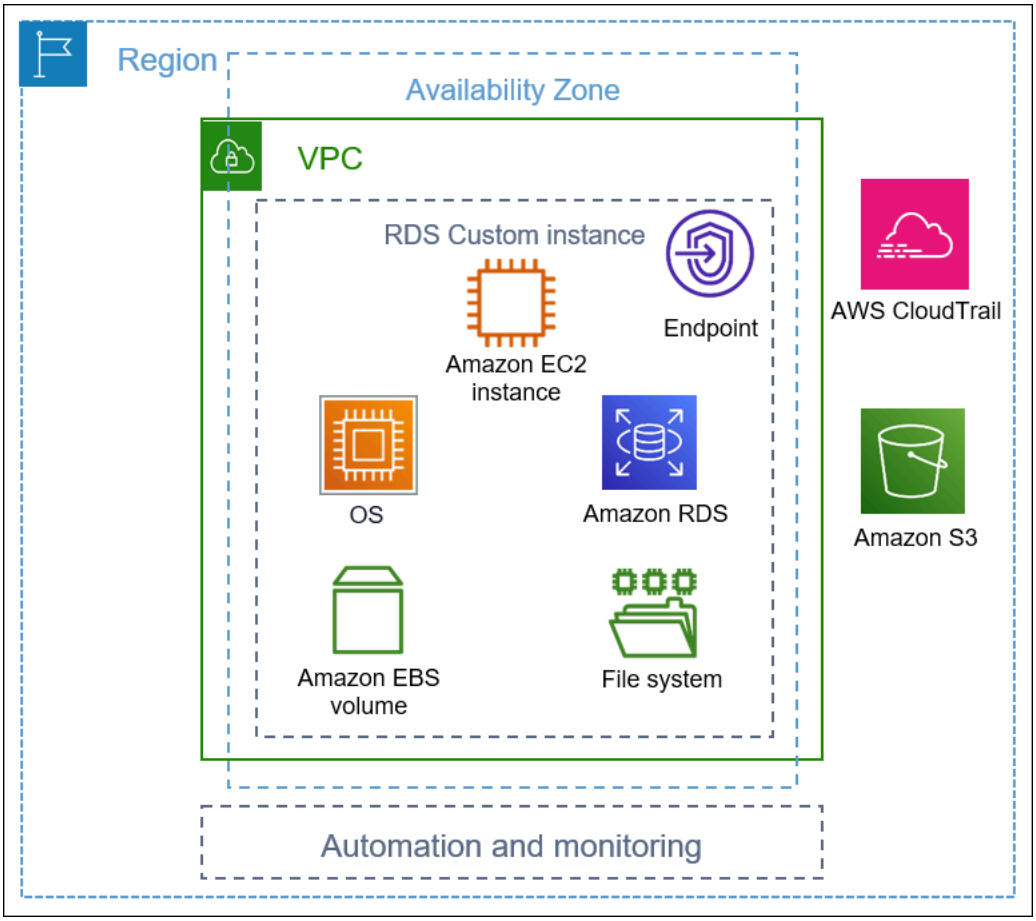
  You can apply custom database patches or modify OS packages on your RDS Custom DB instances.
- Stage an on-premises database before moving it to a fully managed service.

  If you manage your own on-premises database, you can stage the database to RDS Custom as-is. After you familiarize yourself with the cloud environment, you can migrate your database to a fully managed Amazon RDS DB instance.
- Create your own automation.

  You can create, schedule, and run custom automation scripts for reporting, management, or diagnostic tools.

Amazon RDS Custom architecture

Amazon RDS Custom architecture is based on Amazon RDS, with important differences. The following diagram shows the key components of the RDS Custom architecture.



Topics

- VPC
- RDS Custom automation and monitoring
- Amazon S3
- AWS CloudTrail

VPC

As in Amazon RDS, your RDS Custom DB instance resides in a virtual private cloud (VPC).



Your RDS Custom DB instance consists of the following main components:

- Amazon EC2 instance
- Instance endpoint
- Operating system installed on the Amazon EC2 instance
- Amazon EBS storage, which contains any additional file systems

RDS Custom automation and monitoring

RDS Custom has automation software that runs outside of the DB instance. This software communicates with agents on the DB instance and with other components within the overall RDS Custom environment.

The RDS Custom monitoring and recovery features offer similar functionality to Amazon RDS. By default, RDS Custom is in full automation mode. The automation software has the following primary responsibilities:

- Collect metrics and send notifications
- Perform automatic instance recovery

An important responsibility of RDS Custom automation is responding to problems with your Amazon EC2 instance. For various reasons, the host might become impaired or unreachable. RDS Custom resolves these problems by either rebooting or replacing the Amazon EC2 instance.

Topics

- Amazon RDS Custom host replacement
- RDS Custom support perimeter

Amazon RDS Custom host replacement

If the Amazon EC2 host becomes impaired, RDS Custom attempts to reboot it. If this effort fails, RDS Custom uses the same stop and start feature included in Amazon EC2. The only customer-visible change when a host is replaced is a new public IP address.

Topics

- Stopping and starting the host
- Effects of host replacement
- Best practices for Amazon EC2 hosts

Stopping and starting the host

RDS Custom automatically takes the following steps, with no user intervention required:

1. Stops the Amazon EC2 host.

   The EC2 instance performs a normal shutdown and stops running. Any Amazon EBS volumes remain attached to the instance, and their data persists. Any data stored in the instance store volumes (not supported on RDS Custom) or RAM of the host computer is gone.

   For more information, see Stop and start your instance in the *Amazon EC2 User Guide*.
2. Starts the Amazon EC2 host.

   The EC2 instance migrates to a new underlying host hardware. In some cases, the RDS Custom DB instance remains on the original host.

Effects of host replacement

In RDS Custom, you have full control over the root device volume and Amazon EBS storage volumes. The root volume can contain important data and configurations that you don't want to lose.

RDS Custom for Oracle retains all database and customer data after the operation, including root volume data. No user intervention is required. On RDS Custom for SQL Server, database data is retained, but any data on the C: drive, including operating system and customer data, is lost.

After the host replacement process, the Amazon EC2 host has a new public IP address. The new host retains the data and metadata shown in the following table.

Metadata or data	RDS Custom for Oracle retention	RDS Custom for SQL Server retention
EC2 instance ID	Yes	No
EC2 instance metadata	Yes	Yes
Data storage volume data	Yes	Yes
Root volume data	Yes	No
Private IP addresses	Yes	Yes
Elastic IP addresses	Yes	Yes

Best practices for Amazon EC2 hosts

The Amazon EC2 host replacement feature covers the majority of Amazon EC2 impairment scenarios. We recommend that you adhere to the following best practices:

- Before you change your configuration or the operating system, back up your data. If the root volume or operating system becomes corrupt, host replacement can't repair it. Your only options are restoring from a DB snapshot or point-in-time recovery.
- Don't manually stop or terminate the physical Amazon EC2 host. Both actions result in the instance being put outside the RDS Custom support perimeter.
- (RDS Custom for SQL Server) If you attach additional volumes to the Amazon EC2 host, configure them to remount upon restart. If the host is impaired, RDS Custom might stop and start the host automatically.

RDS Custom support perimeter

RDS Custom provides additional monitoring capability called the *support perimeter*. This additional monitoring ensures that your RDS Custom DB instance uses a supported AWS infrastructure, operating system, and database.

The support perimeter checks that your DB instance conforms to the requirements listed in Fixing unsupported configurations in RDS Custom for Oracle and Fixing unsupported configurations in RDS Custom for SQL Server. If any of these requirements aren't met, RDS Custom considers your DB instance to be outside of the support perimeter.

Topics

- Unsupported configurations in RDS Custom
- Troubleshooting unsupported configurations

Unsupported configurations in RDS Custom

When your DB instance is outside the support perimeter, RDS Custom changes the DB instance status to `unsupported-configuration` and sends event notifications. After you fix the configuration problems, RDS Custom changes the DB instance status back to `available`.

While your DB instance is in the `unsupported-configuration` state, the following is true:

- Your database is reachable. An exception is when the DB instance is in the `unsupported-configuration` because the database is shutting down unexpectedly.
- You can't modify your DB instance.
- You can't take DB snapshots.
- Automatic backups aren't created.
- For RDS Custom for SQL Server DB instances only, RDS Custom doesn't replace the underlying Amazon EC2 instance if it becomes impaired. For more information about host replacement, see Amazon RDS Custom host replacement.
- You can delete your DB instance, but most other RDS Custom API operations aren't available.
- RDS Custom continues to support point-in-time recovery (PITR) by archiving redo log files and uploading them to Amazon S3. PITR in an `unsupported-configuration` state differs in the following ways:

- PITR can take a long time to completely restore to a new RDS Custom DB instance. This situation occurs because you can't take either automated or manual snapshots while the instance is in the `unsupported-configuration` state.
- PITR has to replay more redo logs starting from the most recent snapshot taken before the instance entered the `unsupported-configuration` state.
- In some cases, the DB instance is in the `unsupported-configuration` state because you made a change that prevented the uploading of archived redo log files. Examples include stopping the EC2 instance, stopping the RDS Custom agent, and detaching EBS volumes. In such cases, PITR can't restore the DB instance to the latest restorable time.

Troubleshooting unsupported configurations

RDS Custom provides troubleshooting guidance for the `unsupported-configuration` state. Although some guidance applies to both RDS Custom for Oracle and RDS Custom for SQL Server, other guidance depends on your DB engine. For engine-specific troubleshooting information, see the following topics:

- Fixing unsupported configurations in RDS Custom for Oracle
- Fixing unsupported configurations in RDS Custom for SQL Server

Amazon S3

If you use RDS Custom for Oracle, you upload installation media to a user-created Amazon S3 bucket. RDS Custom for Oracle uses the media in this bucket to create a custom engine version (CEV). A *CEV* is a binary volume snapshot of a database version and Amazon Machine Image (AMI). From the CEV, you can create an RDS Custom DB instance. For more information, see Working with custom engine versions for Amazon RDS Custom for Oracle.

For both RDS Custom for Oracle and RDS Custom for SQL Server, RDS Custom automatically creates an Amazon S3 bucket prefixed with the string `do-not-delete-rds-custom-`. RDS Custom uses the `do-not-delete-rds-custom-` S3 bucket to store the following types of files:

- AWS CloudTrail logs for the trail created by RDS Custom
- Support perimeter artifacts (see RDS Custom support perimeter)
- Database redo log files (RDS Custom for Oracle only)
- Transaction logs (RDS Custom for SQL Server only)

- Custom engine version artifacts (RDS Custom for Oracle only)

RDS Custom creates the `do-not-delete-rds-custom-` S3 bucket when you create either of the following resources:

- Your first CEV for RDS Custom for Oracle
- Your first DB instance for RDS Custom for SQL Server

RDS Custom creates one bucket for each combination of the following:

- AWS account ID
- Engine type (either RDS Custom for Oracle or RDS Custom for SQL Server)
- AWS Region

For example, if you create RDS Custom for Oracle CEVs in a single AWS Region, one `do-not-delete-rds-custom-` bucket exists. If you create multiple RDS Custom for SQL Server instances, and they reside in different AWS Regions, one `do-not-delete-rds-custom-` bucket exists in each AWS Region. If you create one RDS Custom for Oracle instance and two RDS Custom for SQL Server instances in a single AWS Region, two `do-not-delete-rds-custom-` buckets exist.

AWS CloudTrail

RDS Custom automatically creates an AWS CloudTrail trail whose name begins with `do-not-delete-rds-custom-`. The RDS Custom support perimeter relies on the events from CloudTrail to determine whether your actions affect RDS Custom automation. For more information, see Troubleshooting unsupported configurations.

RDS Custom creates the trail when you create your first DB instance. RDS Custom creates one trail for each combination of the following:

- AWS account ID
- Engine type (either RDS Custom for Oracle or RDS Custom for SQL Server)
- AWS Region

When you delete an RDS Custom DB instance, the CloudTrail for this instance isn't automatically removed. In this case, your AWS account continues to be billed for the undeleted CloudTrail. RDS

Custom is not responsible for the deletion of this resource. To learn how to remove the CloudTrail manually, see Deleting a trail in the *AWS CloudTrail User Guide*.

Security in Amazon RDS Custom

Familiarize yourself with the security considerations for RDS Custom.

For more information about security for RDS Custom, see the following topics.

- Securing your Amazon S3 bucket against the confused deputy problem
- Rotating RDS Custom for Oracle credentials for compliance programs

How RDS Custom securely manages tasks on your behalf

RDS Custom uses the following tools and techniques to securely run operations on your behalf:

AWSServiceRoleForRDSCustom service-linked role

A *service-linked role* is predefined by the service and includes all permissions that the service needs to call other AWS services on your behalf. For RDS Custom, `AWSServiceRoleForRDSCustom` is a service-linked role that is defined according to the principle of least privilege. RDS Custom uses the permissions in `AmazonRDSCustomServiceRolePolicy`, which is the policy attached to this role, to perform most provisioning and all off-host management tasks. For more information, see AmazonRDSCustomServiceRolePolicy.

When it performs tasks on the host, RDS Custom automation uses credentials from the service-linked role to run commands using AWS Systems Manager. You can audit the command history through the Systems Manager command history and AWS CloudTrail. Systems Manager connects to your RDS Custom DB instance using your networking setup. For more information, see Step 4: Configure IAM for RDS Custom for Oracle.

Temporary IAM credentials

When provisioning or deleting resources, RDS Custom sometimes uses temporary credentials derived from the credentials of the calling IAM principal. These IAM credentials are restricted by the IAM policies attached to that principal and expire after the operation is completed. To learn about the permissions required for IAM principals who use RDS Custom, see Step 5: Grant required permissions to your IAM user or role.

Amazon EC2 instance profile

An EC2 instance profile is a container for an IAM role that you can use to pass role information to an EC2 instance. An EC2 instance underlies an RDS Custom DB instance. You provide an

instance profile when you create an RDS Custom DB instance. RDS Custom uses EC2 instance profile credentials when it performs host-based management tasks such as backups. For more information, see Create your IAM role and instance profile manually.

SSH key pair

When RDS Custom creates the EC2 instance that underlies a DB instance, it creates an SSH key pair on your behalf. The key uses the naming prefix `do-not-delete-rds-custom-ssh-privatekey-db-` or `rds-custom!oracle-do-not-delete-`*`db_resource_id-uuid`*`-ssh-privatekey`. AWS Secrets Manager stores this SSH private key as a secret in your AWS account. Amazon RDS doesn't store, access, or use these credentials. For more information, see Amazon EC2 key pairs and Linux instances.

SSL certificates

RDS Custom DB instances don't support managed SSL certificates. If you want to deploy SSL, you can self-manage SSL certificates in your own wallet and create an SSL listener to secure the connections between the client database or for database replication. For more information, see Configuring Transport Layer Security Authentication in the Oracle Database documentation.

Securing your Amazon S3 bucket against the confused deputy problem

When you create an Amazon RDS Custom for Oracle custom engine version (CEV) or an RDS Custom for SQL Server DB instance, RDS Custom creates an Amazon S3 bucket. The S3 bucket stores files such as CEV artifacts, redo (transaction) logs, configuration items for the support perimeter, and AWS CloudTrail logs.

You can make these S3 buckets more secure by using the global condition context keys to prevent the *confused deputy problem*. For more information, see Preventing cross-service confused deputy problems.

The following RDS Custom for Oracle example shows the use of the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in an S3 bucket policy. For RDS Custom for Oracle, make sure to include the Amazon Resource Names (ARNs) for the CEVs and the DB instances. For RDS Custom for SQL Server, make sure to include the ARN for the DB instances.

```
...
{
```

```
  "Sid": "AWSRDSCustomForOracleInstancesObjectLevelAccess",
  "Effect": "Allow",
  "Principal": {
     "Service": "custom.rds.amazonaws.com"
  },
  "Action": [
     "s3:GetObject",
     "s3:GetObjectVersion",
     "s3:DeleteObject",
     "s3:DeleteObjectVersion",
     "s3:GetObjectRetention",
     "s3:BypassGovernanceRetention"
  ],
  "Resource": "arn:aws:s3:::do-not-delete-rds-custom-123456789012-us-east-2-c8a6f7/
RDSCustomForOracle/Instances/*",
  "Condition": {
     "ArnLike": {
        "aws:SourceArn": [
            "arn:aws:rds:us-east-2:123456789012:db:*",
            "arn:aws:rds:us-east-2:123456789012:cev:*/*"
        ]
     },
     "StringEquals": {
        "aws:SourceAccount": "123456789012"
    }
  }
},
...
```

Rotating RDS Custom for Oracle credentials for compliance programs

Some compliance programs require database user credentials to change periodically, for example, every 90 days. RDS Custom for Oracle automatically rotates credentials for some predefined database users.

Topics

- Automatic rotation of credentials for predefined users
- Guidelines for rotating user credentials
- Rotating user credentials manually

Automatic rotation of credentials for predefined users

If your RDS Custom for Oracle DB instance is hosted in Amazon RDS, credentials for the following predefined Oracle users rotate every 30 days automatically. Credentials for the preceding users reside in AWS Secrets Manager.

Database user	Created by	Supported engine versions	Notes
SYS	Oracle	custom-oracle-ee custom-oracle-ee-cdb custom-oracle-se2 custom-oracle-se2-cdb	
SYSTEM	Oracle	custom-oracle-ee custom-oracle-ee-cdb custom-oracle-se2 custom-oracle-se2-cdb	
RDSADMIN	RDS	custom-oracle-ee custom-oracle-se2	
C##RDSADMIN	RDS	custom-oracle-ee-cdb custom-oracle-se2-cdb	User names with a C## prefix exist only in CDBs. For more information about CDBs, see Overview of Amazon RDS Custom for Oracle architecture.
RDS_DATAGUARD	RDS	custom-oracle-ee	This user exists only in read replicas, source databases for read replicas, and databases that you have physically

Database user	Created by	Supported engine versions	Notes
			migrated into RDS Custom using Oracle Data Guard.
C##RDS_DATAGUARD	RDS	custom-oracle-ee-cdb	This user exists only in read replicas, source databases for read replicas, and databases that you have physically migrated into RDS Custom using Oracle Data Guard. User names with a C## prefix exist only in CDBs. For more information about CDBs, see Overview of Amazon RDS Custom for Oracle architecture.

An exception to the automatic credential rotation is an RDS Custom for Oracle DB instance that you have manually configured as a standby database. RDS only rotates credentials for read replicas that you have created using the `create-db-instance-read-replica` CLI command or `CreateDBInstanceReadReplica` API.

Guidelines for rotating user credentials

To make sure that your credentials rotate according to your compliance program, note the following guidelines:

- If your DB instance rotates credentials automatically, don't manually change or delete a secret, password file, or password for users listed in Predefined Oracle users. Otherwise, RDS Custom might place your DB instance outside of the support perimeter, which suspends automatic rotation.
- The RDS master user is not predefined, so you are responsible for either changing the password manually or setting up automatic rotation in Secrets Manager. For more information, see Rotate AWS Secrets Manager secrets.

Rotating user credentials manually

For the following categories of databases, RDS doesn't automatically rotate the credentials for the users listed in Predefined Oracle users:

- A database that you configured manually to function as a standby database.
- An on-premises database.
- A DB instance that is outside of the support perimeter or in a state in which the RDS Custom automation can't run. In this case, RDS Custom also doesn't rotate keys.

If your database is in any of the preceding categories, you must rotate your user credentials manually.

To rotate user credentials manually for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In **Databases**, make sure that RDS isn't currently backing up your DB instance or performing operations such configuring high availability.
3. In the database details page, choose **Configuration** and note the Resource ID for the DB instance. Or you can use the AWS CLI command `describe-db-instances`.
4. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
5. In the search box, enter the resource ID of your database and search for a secret using either of the following naming conventions:

   ```
   do-not-delete-rds-custom-resource_id-uuid
   rds-custom!oracle-do-not-delete-resource_id-uuid
   ```

   This secret stores the password for RDSADMIN, SYS, and SYSTEM. The following sample keys are for the DB instance with the resource ID db-ABCDEFG12HIJKLNMNOPQRS3TUVWX and UUID 123456:

   ```
   do-not-delete-rds-custom-db-ABCDEFG12HIJKLNMNOPQRS3TUVWX-123456
   rds-custom!oracle-do-not-delete-db-ABCDEFG12HIJKLNMNOPQRS3TUVWX-123456
   ```

Important

If your DB instance is a read replica and uses the `custom-oracle-ee-cdb` engine, two secrets exist with the suffix *db-resource_id-uuid*, one for the master user and the other for RDSADMIN, SYS, and SYSTEM. To find the correct secret, run the following command on the host:

```
cat /opt/aws/rdscustomagent/config/database_metadata.json | python3 -c
 "import sys,json; print(json.load(sys.stdin)['dbMonitoringUserPassword'])"
```

The dbMonitoringUserPassword attribute indicates the secret for RDSADMIN, SYS, and SYSTEM.

6. If your DB instance exists in an Oracle Data Guard configuration, search for a secret using either of the following naming conventions:

```
do-not-delete-rds-custom-resource_id-uuid-dg
rds-custom!oracle-do-not-delete-resource_id-uuid-dg
```

This secret stores the password for RDS_DATAGUARD. The following sample keys are for the DB instance with the DB resource ID db-ABCDEFG12HIJKLNMNOPQRS3TUVWX and UUID **789012**:

```
do-not-delete-rds-custom-db-ABCDEFG12HIJKLNMNOPQRS3TUVWX-789012-dg
rds-custom!oracle-do-not-delete-db-ABCDEFG12HIJKLNMNOPQRS3TUVWX-789012-dg
```

7. For all database users listed in Predefined Oracle users, update the passwords by following the instructions in Modify an AWS Secrets Manager secret.
8. If your database is a standalone database or a source database in an Oracle Data Guard configuration:

   a. Start your Oracle SQL client and log in as SYS.

   b. Run a SQL statement in the following form for each database user listed in Predefined Oracle users:

```
ALTER USER user-name IDENTIFIED BY pwd-from-secrets-manager ACCOUNT UNLOCK;
```

For example, if the new password for RDSADMIN stored in Secrets Manager is pwd-123, run the following statement:

```
ALTER USER RDSADMIN IDENTIFIED BY pwd-123 ACCOUNT UNLOCK;
```

9. If your DB instance runs Oracle Database 12c Release 1 (12.1) and is managed by Oracle Data Guard, manually copy the password file (orapw) from the primary DB instance to each standby DB instance.

   If your DB instance is hosted in Amazon RDS, the password file location is /rdsdbdata/config/orapw. For databases that aren't hosted in Amazon RDS, the default location is $ORACLE_HOME/dbs/orapw$ORACLE_SID on Linux and UNIX and %ORACLE_HOME%\database\PWD%ORACLE_SID%.ora on Windows.

Working with RDS Custom for Oracle

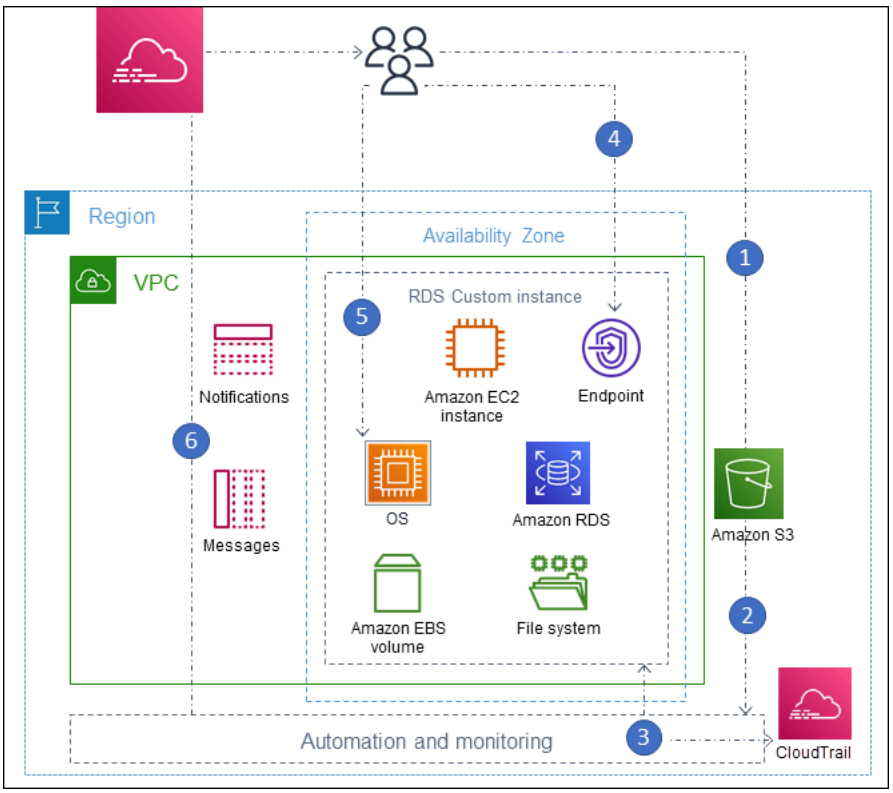
Following, you can find instructions for creating, managing, and maintaining your RDS Custom for Oracle DB instances.

Topics

- RDS Custom for Oracle workflow
- Database architecture for Amazon RDS Custom for Oracle
- Feature availability and support for RDS Custom for Oracle
- RDS Custom for Oracle requirements and limitations
- Setting up your environment for Amazon RDS Custom for Oracle
- Working with custom engine versions for Amazon RDS Custom for Oracle
- Configuring a DB instance for Amazon RDS Custom for Oracle
- Managing an Amazon RDS Custom for Oracle DB instance
- Managing a Multi-AZ deployment for RDS Custom for Oracle
- Working with Oracle replicas for RDS Custom for Oracle
- Backing up and restoring an Amazon RDS Custom for Oracle DB instance
- Working with option groups in RDS Custom for Oracle
- Migrating an on-premises database to RDS Custom for Oracle
- Upgrading a DB instance for Amazon RDS Custom for Oracle
- Troubleshooting DB issues for Amazon RDS Custom for Oracle
- Known issues for Amazon RDS Custom for Oracle

RDS Custom for Oracle workflow

The following diagram shows the typical workflow for RDS Custom for Oracle.



The steps are as follows:

1. Upload your database software to your Amazon S3 bucket.

   For more information, see Step 3: Upload your installation files to Amazon S3.

2. Create an RDS Custom for Oracle custom engine version (CEV) from your media.

   Choose either the CDB architecture or the traditional non-CDB architecture. For more information, see Creating a CEV.

3. Create an RDS Custom for Oracle DB instance from a CEV.

   For more information, see Creating an RDS Custom for Oracle DB instance.

4. Connect your application to the DB instance endpoint.

   For more information, see Connecting to your RDS Custom DB instance using SSH and Connecting to your RDS Custom DB instance using Session Manager.

5. (Optional) Access the host to customize your software.

6. Monitor notifications and messages generated by RDS Custom automation.

Database installation files

Your responsibility for media is a key difference between Amazon RDS and RDS Custom. Amazon RDS, which is a fully managed service, supplies the Amazon Machine Image (AMI) and database software. The Amazon RDS database software is preinstalled, so you need only choose a database engine and version, and create your database.

For RDS Custom, you supply your own media. When you create a custom engine version, RDS Custom installs the media that you provide. RDS Custom media contains your database installation files and patches. This service model is called *Bring Your Own Media (BYOM)*.

Custom engine versions for RDS Custom for Oracle

An *RDS Custom for Oracle custom engine version* (CEV) is a binary volume snapshot of a database version and AMI. By default, RDS Custom for Oracle uses the most recent AMI that Amazon EC2 makes available. You can also choose to reuse an existing AMI.

CEV manifest

After you download Oracle database installation files from Oracle, you upload them to an Amazon S3 bucket. When you create your CEV, you specify the file names in a JSON document called a *CEV manifest*. RDS Custom for Oracle uses the specified files and the AMI to create your CEV.

RDS Custom for Oracle provides JSON manifest templates with our recommended .zip files for each supported Oracle Database release. For example, the following template is for the 19.17.0.0.0 RU.

```
{
    "mediaImportTemplateVersion": "2020-08-14",
    "databaseInstallationFileNames": [
        "V982063-01.zip"
    ],
    "opatchFileNames": [
        "p6880880_190000_Linux-x86-64.zip"
    ],
    "psuRuPatchFileNames": [
        "p34419443_190000_Linux-x86-64.zip",
        "p34411846_190000_Linux-x86-64.zip"
    ],
    "otherPatchFileNames": [
        "p28852325_190000_Linux-x86-64.zip",
```

```
        "p29997937_190000_Linux-x86-64.zip",
        "p31335037_190000_Linux-x86-64.zip",
        "p32327201_190000_Linux-x86-64.zip",
        "p33613829_190000_Linux-x86-64.zip",
        "p34006614_190000_Linux-x86-64.zip",
        "p34533061_190000_Linux-x86-64.zip",
        "p34533150_190000_Generic.zip",
        "p28730253_190000_Linux-x86-64.zip",
        "p29213893_1917000DBRU_Generic.zip",
        "p33125873_1917000DBRU_Linux-x86-64.zip",
        "p34446152_1917000DBRU_Linux-x86-64.zip"
    ]
}
```

You can also specify installation parameters in the JSON manifest. For example, you can set nondefault values for the Oracle base, Oracle home, and the ID and name of the UNIX/Linux user and group. For more information, see JSON fields in the CEV manifest.

CEV naming format

Name your RDS Custom for Oracle CEV using a customer-specified string. The name format is the following, depending on your Oracle Database release:

- `19.`*`customized_string`*
- `18.`*`customized_string`*
- `12.2.`*`customized_string`*
- `12.1.`*`customized_string`*

You can use 1–50 alphanumeric characters, underscores, dashes, and periods. For example, you might name your CEV `19.my_cev1`.

Oracle multitenant architecture in RDS Custom for Oracle

The Oracle multitenant architecture enables an Oracle database to function as a container database (CDB). A CDB includes zero, one, or many customer-created pluggable databases (PDBs). A PDB is a portable collection of schemas and objects that appears to an application as a traditional non-CDB.

When you create an RDS Custom for Oracle CEV, you specify the either the CDB or non-CDB architecture. You can create an RDS Custom for Oracle CDB only when the CEV that you used to

create it uses the Oracle multitenant architecture. For more information, see Working with custom engine versions for Amazon RDS Custom for Oracle.

Creating a DB instance for RDS Custom for Oracle

After you create your CEV, it's available for use. You can create multiple CEVs, and you can create multiple RDS Custom for Oracle DB instances from any CEV. You can also change the status of a CEV to make it available or inactive.

You can either create your RDS Custom for Oracle DB instance with the Oracle multitenant architecture (`custom-oracle-ee-cdb` or `custom-oracle-se2-cdb` engine type) or with the traditional non-CDB architecture (`custom-oracle-ee` or `custom-oracle-se2` engine type). When you create a container database (CDB), it contains one pluggable database (PDB) and one PDB seed. You can create additional PDBs manually using Oracle SQL.

To create your RDS Custom for Oracle DB instance, use the `create-db-instance` command. In this command, specify which CEV to use. The procedure is similar to creating an Amazon RDS DB instance. However, some parameters are different. For more information, see Configuring a DB instance for Amazon RDS Custom for Oracle.

Database connection

Like an Amazon RDS DB instance, an RDS Custom DB instance resides in a virtual private cloud (VPC). Your application connects to the Oracle database using an Oracle listener.

If your database is a CDB, you can use the listener `L_RDSCDB_001` to connect to the CDB root and to a PDB. If you plug a non-CDB into a CDB, make sure to set `USE_SID_AS_SERVICE_LISTENER = ON` so that migrated applications keep the same settings.

When you connect to a non-CDB, the master user is the user for the non-CDB. When you connect to a CDB, the master user is the user for the PDB. To connect to the CDB root, log in to the host, start a SQL client, and create an administrative user with SQL commands.

RDS Custom customization

You can access the RDS Custom host to install or customize software. To avoid conflicts between your changes and the RDS Custom automation, you can pause the automation for a specified period. During this period, RDS Custom doesn't perform monitoring or instance recovery. At the end of the period, RDS Custom resumes full automation. For more information, see Pausing and resuming your RDS Custom DB instance.

Database architecture for Amazon RDS Custom for Oracle

RDS Custom for Oracle supports both the Oracle multitenant and non-multitenant architecture.

Topics

- Supported Oracle database architectures
- Supported engine types
- Supported features in the Oracle multitenant architecture

Supported Oracle database architectures

The *Oracle multitenant architecture*, also called the *CDB architecture*, allows an Oracle database to function as a container database (CDB). A CDB includes pluggable databases (PDBs). A PDB is a collection of schemas and objects that appears to an application as a traditional Oracle database. For more information, see Introduction to the Multitenant Architecture in the *Oracle Multitenant Administrator's Guide*.

The CDB and non-CDB architectures are mutually exclusive. If an Oracle database isn't a CDB, it's a non-CDB and so can't contain PDBs. In RDS Custom for Oracle, only Oracle Database 19c supports the CDB architecture. Thus, if you create DB instances using previous Oracle database releases, you can create only non-CDBs. For more information, see Multitenant architecture considerations.

Supported engine types

When you create an Amazon RDS Custom for Oracle CEV or DB instance, choose either a CDB engine type or a non-CDB engine type:

- `custom-oracle-ee-cdb` and `custom-oracle-se2-cdb`

  These engine types specify the Oracle multitenant architecture. This option is available only for Oracle Database 19c. When you create an RDS for Oracle DB instance using the multitenant architecture, your CDB includes the following containers:

  - CDB root (`CDB$ROOT`)
  - PDB seed (`PDB$SEED`)
  - Initial PDB

  You can create more PDBs using the Oracle SQL command `CREATE PLUGGABLE DATABASE`. You can't use RDS APIs to create or delete PDBs.

- `custom-oracle-ee` and `custom-oracle-se2`

  These engine types specify the traditional non-CDB architecture. A non-CDB can't contain pluggable databases (PDBs).

For more information, see Multitenant architecture considerations.

Supported features in the Oracle multitenant architecture

An RDS Custom for Oracle CDB instance supports the following features:

- Backups
- Restoring and point-time-restore (PITR) from backups
- Read replicas
- Minor version upgrades

Feature availability and support for RDS Custom for Oracle

In this topic, you can find a summary of the RDS Custom for Oracle feature availability and support for quick reference.

Topics

- AWS Region and database version support for RDS Custom for Oracle
- Database version support for RDS Custom for Oracle
- Edition and licensing support for RDS Custom for Oracle
- DB instance class support for RDS Custom for Oracle
- Option group support for RDS Custom for Oracle

AWS Region and database version support for RDS Custom for Oracle

Feature availability and support vary across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability of RDS Custom for Oracle, see Supported Regions and DB engines for RDS Custom.

Database version support for RDS Custom for Oracle

RDS Custom for Oracle supports the following Oracle database versions:

- Oracle Database 19c
- Oracle Database 18c
- Oracle Database 12c Release 2 (12.2)
- Oracle Database 12c Release 1 (12.1)

Edition and licensing support for RDS Custom for Oracle

RDS Custom for Oracle supports Enterprise Edition (EE) and Standard Edition 2 (SE2) on the BYOL model.

Note the following limitations for Standard Edition 2:

- Oracle Data Guard isn't supported. Thus, you can't create Oracle read replicas.
- You can only use DB instance classes that have 16 or fewer vCPUs (up to 4xlarge).
- A CDB instance on Standard Edition 2 supports a maximum of 3 tenant databases.

- You can't migrate data between Enterprise Edition and Standard Edition 2.

DB instance class support for RDS Custom for Oracle

RDS Custom for Oracle supports the following DB instance classes. If you create a DB instance on Standard Edition 2, you can only use instance classes with 16 or fewer vCPUs (up to 4x large).

Type	Size
db.m7i	db.m7i.large |db.m7i.xlarge |db.m7i.2xlarge |db.m7i.4xlarge |db.m7i.8xlarge |db.m7i.12xlarge |db.m7i.16xlarge |db.m7i.24xlarge |db.m7i.48xlarge |db.m7i.metal-24xl |db.m7i.metal-48xl
db.m6i	db.m6i.large |db.m6i.xlarge |db.m6i.2xlarge |db.m6i.4xlarge |db.m6i.8xlarge |db.m6i.12xlarge |db.m6i.16xlarge |db.m6i.24xlarge |db.m6i.32xlarge |db.m6i.metal
db.m6ic	db.m6id.metal
db.m6ir	db.m6in.metal
db.m5	db.m5.large |db.m5.xlarge |db.m5.2xlarge |db.m5.4xlarge | db.m5.8xlarge |db.m5.12xlarge |db.m5.16xlarge |db.m5.24xlarge
db.r7i	db.r7i.large |db.r7i.xlarge |db.r7i.2xlarge |db.r7i.4xlarge |db.r7i.8xlarge |db.r7i.12xlarge |db.r7i.16xlarge |db.r7i.24xlarge |db.r7i.48xlarge |db.r7i.metal-24xl |db.r7i.metal-48xl
db.r6i	db.r6i.large |db.r6i.xlarge |db.r6i.2xlarge |db.r6i.4xlarge |db.r6i.8xlarge |db.r6i.12xlarge |db.r6i.16xlarge |db.r6i.24xlarge |db.r6i.32xlarge |db.r6i.metal
db.r6id	db.r6id.metal
db.r6in	db.r6in.metal
db.r5b	db.r5b.large |db.r5b.xlarge |db.r5b.2xlarge |db.r5b.4xlarge |db.r5b.8xlarge |db.r5b.12xlarge |db.r5b.16xlarge |db.r5b.24xlarge

Type	Size
db.r5	db.r5.large |db.r5.xlarge |db.r5.2xlarge |db.r5.4xlarge | db.r5.8xlarge |db.r5.12xlarge |db.r5.16xlarge |db.r5.24xlarge
db.x2iec	db.x2iedn.xlarge |db.x2iedn.2xlarge |db.x2iedn.4xlarge | db.x2iedn.8xlarge |db.x2iedn.16xlarge |db.x2iedn.24xlarge | db.x2iedn.32xlarge |db.x2iedn.metal
db.x2idr	db.x2idn.metal
db.x2iez	db.x2iezn.2xlarge |db.x2iezn.4xlarge |db.x2iezn.6xlarge | db.x2iezn.8xlarge |db.x2iezn.12xlarge |db.x2iezn.metal
db.t3	db.t3.medium |db.t3.large |db.t3.xlarge |db.t3.2xlarge

Option group support for RDS Custom for Oracle

You can specify an option group when you create or modify an RDS Custom for Oracle DB instance. For more information, see Working with option groups in RDS Custom for Oracle.

RDS Custom for Oracle requirements and limitations

In this topic, you can find a summary of the Amazon RDS Custom for Oracle feature availability and requirements for quick reference.

Topics

- General requirements for RDS Custom for Oracle
- General limitations for RDS Custom for Oracle
- CEV and AMI limitations for RDS Custom for Oracle
- Unsupported settings for create and modify workflows
- DB instance quotas for your AWS account

General requirements for RDS Custom for Oracle

Make sure to meet the following requirements for Amazon RDS Custom for Oracle:

- You have access to My Oracle Support and Oracle Software Delivery Cloud to download the supported list of installation files and patches for RDS Custom for Oracle. If you use an unknown patch, custom engine version (CEV) creation fails. In this case, contact the RDS Custom support team and ask it to add the missing patch. For more information, see Step 2: Download your database installation files and patches from Oracle Software Delivery Cloud.
- You have access to Amazon S3. You need this service for the following reasons:
  - You upload your Oracle installation files to S3 buckets. You use the uploaded installation files to create your RDS Custom CEV.
  - RDS Custom for Oracle uses scripts downloaded from internally defined S3 buckets to perform actions on your DB instances. These scripts are necessary for onboarding and RDS Custom automation.
  - RDS Custom for Oracle uploads certain files to S3 buckets located in your customer account. These buckets use the following naming format: `do-not-delete-rds-custom-`*`account_id`*`-`*`region`*`-`*`uuid`*. For example, you might have a bucket named `do-not-delete-rds-custom-123456789012-us-east-1-12a3b4`.

  For more information, see Step 3: Upload your installation files to Amazon S3 and Creating a CEV.
- You use the DB instance classes listed in DB instance class support for RDS Custom for Oracle to create your RDS Custom for Oracle DB instances.

- Your RDS Custom for Oracle DB instances run Oracle Linux 8 (recommended) or Oracle Linux 7. If you require Oracle Linux 7, contact Support. For more information, see Considerations for RDS Custom for Oracle database upgrades.
- You specify the gp2, gp3, or io1 solid state drives for Amazon EBS storage. The maximum storage size is 64 TiB.
- You have an AWS KMS key to create an RDS Custom for Oracle DB instance. For more information, see Step 1: Create or reuse a symmetric encryption AWS KMS key.
- You have the AWS Identity and Access Management (IAM) role and instance profile required for creating RDS Custom for Oracle DB instances. For more information, see Step 4: Configure IAM for RDS Custom for Oracle.
- The AWS Identity and Access Management (IAM) user that creates a CEV or RDS Custom DB instance has the required permissions for IAM, CloudTrail, and Amazon S3.

  For more information, see Step 5: Grant required permissions to your IAM user or role.
- You supply your own virtual private cloud (VPC) and security group configuration. For more information, see Step 6: Configure your VPC for RDS Custom for Oracle.
- You supply a networking configuration that RDS Custom for Oracle can use to access other AWS services. For specific requirements, see Step 4: Configure IAM for RDS Custom for Oracle.

General limitations for RDS Custom for Oracle

The following limitations apply to RDS Custom for Oracle:

- You can't modify the DB instance identifier of an existing RDS Custom for Oracle DB instance.
- You can't specify the Oracle multitenant architecture for any release except Oracle Database 19c.
- You can't create multiple Oracle databases on a single RDS Custom for Oracle DB instance.
- You can’t stop your RDS Custom for Oracle DB instance or its underlying Amazon EC2 instance. Billing for an RDS Custom for Oracle DB instance can't be stopped.
- You can't use automatic shared memory management because RDS Custom for Oracle supports automatic memory management only. For more information, see Automatic Memory Management in the *Oracle Database Administrator’s Guide*.
- Make sure not to change the `DB_UNIQUE_NAME` for the primary DB instance. Changing the name causes any restore operation to become stuck.
- You can't make more than 20 snapshot copies at the same time in the same Region.

- You can't use the `describe-reserved-db-instances` API for RDS Custom for Oracle DB instances.

For limitations specific to modifying an RDS Custom for Oracle DB instance, see Modifying your RDS Custom for Oracle DB instance. For replication limitations, see General limitations for RDS Custom for Oracle replication.

CEV and AMI limitations for RDS Custom for Oracle

The following limitations apply to RDS Custom for Oracle CEVs and AMIs:

- You can't provide your own AMI for use in an RDS Custom for Oracle CEV. You can specify either the default AMI, which uses Oracle Linux 8, or an AMI that has been previously used by an RDS Custom for Oracle CEV.

  > **Note**
  >
  > RDS Custom for Oracle releases a new default AMI when common vulnerabilities and exposures are discovered. No fixed schedule is available or guaranteed. RDS Custom for Oracle tends to publish a new default AMI every 30 days.

- You can't modify a CEV to use a different AMI.
- You can't create a CDB instance from a CEV that uses the `custom-oracle-ee` or `custom-oracle-se2` engine types. The CEV must use `custom-oracle-ee-cdb` or `custom-oracle-se2-cdb`.
- RDS Custom for Oracle doesn't currently allow you to upgrade the OS of your RDS Custom for Oracle DB instance with RDS API calls. As a workaround, you can update your OS manually with the following command: `sudo yum update --security`.

Unsupported settings for create and modify workflows

When you create or modify an RDS Custom for Oracle DB instance, you can't do the following:

- Change the number of CPU cores and threads per core on the DB instance class.
- Turn on storage autoscaling.
- Set backup retention to `0`.
- Configure Kerberos authentication.

- Specify your own DB parameter group or option group.
- Turn on Performance Insights.
- Turn on automatic minor version upgrade.

DB instance quotas for your AWS account

Make sure that the combined number of RDS Custom and Amazon RDS DB instances doesn't exceed your quota limit. For example, if your quota for Amazon RDS is 40 DB instances, you can have 20 RDS Custom for Oracle DB instances and 20 Amazon RDS DB instances.

Setting up your environment for Amazon RDS Custom for Oracle

Before you create an Amazon RDS Custom for Oracle DB instance, perform the following tasks.

Topics

- Step 1: Create or reuse a symmetric encryption AWS KMS key
- Step 2: Download and install the AWS CLI
- Step 3: Extract the CloudFormation templates for RDS Custom for Oracle
- Step 4: Configure IAM for RDS Custom for Oracle
- Step 5: Grant required permissions to your IAM user or role
- Step 6: Configure your VPC for RDS Custom for Oracle

Step 1: Create or reuse a symmetric encryption AWS KMS key

Customer managed keys are AWS KMS keys in your AWS account that you create, own, and manage. A customer managed symmetric encryption KMS key is required for RDS Custom. When you create an RDS Custom for Oracle DB instance, you supply the KMS key identifier. For more information, see Configuring a DB instance for Amazon RDS Custom for Oracle.

You have the following options:

- If you have an existing customer managed KMS key in your AWS account, you can use it with RDS Custom. No further action is necessary.
- If you already created a customer managed symmetric encryption KMS key for a different RDS Custom engine, you can reuse the same KMS key. No further action is necessary.
- If you don't have an existing customer managed symmetric encryption KMS key in your account, create a KMS key by following the instructions in Creating keys in the *AWS Key Management Service Developer Guide*.
- If you're creating a CEV or RDS Custom DB instance, and your KMS key is in a different AWS account, make sure to use the AWS CLI. You can't use the AWS console with cross-account KMS keys.

> **Important**
>
> RDS Custom doesn't support AWS managed KMS keys.

Make sure that your symmetric encryption key grants access to the `kms:Decrypt` and `kms:GenerateDataKey` operations to the AWS Identity and Access Management (IAM) role in your IAM instance profile. If you have a new symmetric encryption key in your account, no changes are required. Otherwise, make sure that your symmetric encryption key's policy grants access to these operations.

For more information, see Step 4: Configure IAM for RDS Custom for Oracle.

For more information about configuring IAM for RDS Custom for Oracle, see Step 4: Configure IAM for RDS Custom for Oracle.

Step 2: Download and install the AWS CLI

AWS provides you with a command-line interface to use RDS Custom features. You can use either version 1 or version 2 of the AWS CLI.

For information about downloading and installing the AWS CLI, see Installing or updating the latest version of the AWS CLI.

Skip this step if either of the following is true:

- You plan to access RDS Custom only from the AWS Management Console.
- You have already downloaded the AWS CLI for Amazon RDS or a different RDS Custom DB engine.

Step 3: Extract the CloudFormation templates for RDS Custom for Oracle

To simplify setup, we strongly recommend that you use CloudFormation templates to create CloudFormation stacks. If you plan to configure IAM and your VPC manually, skip this step.

Topics

- Step 3a: Download the CloudFormation template files
- Step 3b: Extract custom-oracle-iam.json
- Step 3c: Extract custom-vpc.json

Step 3a: Download the CloudFormation template files

A *CloudFormation template* is a declaration of the AWS resources that make up a stack. The template is stored as a JSON file.

To download the CloudFormation template files

1. Open the context (right-click) menu for the link custom-oracle-iam.zip and choose **Save Link As**.
2. Save the file to your computer.
3. Repeat the previous steps for the link custom-vpc.zip.

   If you already configured your VPC for RDS Custom, skip this step.

Step 3b: Extract custom-oracle-iam.json

Open the `custom-oracle-iam.zip` file that you downloaded, and then extract the file `custom-oracle-iam.json`. The beginning of the file looks like the following.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "EncryptionKey": {
      "Type": "String",
      "Default": "*",
      "Description": "KMS Key ARN for encryption of data managed by RDS Custom and by
 DB Instances."
    }
  },
  "Resources": {
    "RDSCustomInstanceServiceRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "RoleName": { "Fn::Sub": "AWSRDSCustomInstanceRole-${AWS::Region}" },
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                "Service": "ec2.amazonaws.com"
              }
            }
          ]
        },...
```

Step 3c: Extract custom-vpc.json

> **Note**
>
> If you already configured an existing VPC for RDS Custom for Oracle, skip this step. For more information, see Configure your VPC manually for RDS Custom for Oracle.

Open the `custom-vpc.zip` file that you downloaded, and then extract the file `custom-vpc.json`. The beginning of the file looks like the following.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Parameters": {
    "PrivateVpc": {
      "Type": "AWS::EC2::VPC::Id",
      "Description": "Private VPC Id to use for RDS Custom DB Instances"
    },
    "PrivateSubnets": {
      "Type": "List<AWS::EC2::Subnet::Id>",
      "Description": "Private Subnets to use for RDS Custom DB Instances"
    },
    "RouteTable": {
      "Type": "String",
      "Description": "Route Table that must be associated with the PrivateSubnets and
 used by S3 VPC Endpoint",
      "AllowedPattern": "rtb-[0-9a-z]+"
    }
  },
  "Resources": {
    "DBSubnetGroup": {
      "Type": "AWS::RDS::DBSubnetGroup",
      "Properties": {
        "DBSubnetGroupName": "rds-custom-private",
        "DBSubnetGroupDescription": "RDS Custom Private Network",
        "SubnetIds": {
          "Ref": "PrivateSubnets"
        }
      }
    },...
```

Step 4: Configure IAM for RDS Custom for Oracle

You use an IAM role or IAM user (known as an *IAM entity*) to create an RDS Custom DB instance using the console or AWS CLI. This IAM entity must have the necessary permissions for instance creation.

You can configure IAM using either CloudFormation or manual steps.

> **Important**
>
> We strongly recommend that you configure your RDS Custom for Oracle environment using CloudFormation. This technique is the easiest and least error-prone.

Topics

- Configure IAM using CloudFormation
- Create your IAM role and instance profile manually

Configure IAM using CloudFormation

When you use the CloudFormation template for IAM, it creates the following required resources:

- An instance profile named `AWSRDSCustomInstanceProfile-`*`region`*
- A service role named `AWSRDSCustomInstanceRole-`*`region`*
- An access policy named `AWSRDSCustomIamRolePolicy` that is attached to the service role

To configure IAM using CloudFormation

1. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Start the Create Stack wizard, and choose **Create Stack**.
3. On the **Create stack** page, do the following:
   a. For **Prepare template**, choose **Template is ready**.
   b. For **Template source**, choose **Upload a template file**.
   c. For **Choose file**, navigate to, then choose **custom-oracle-iam.json**.
   d. Choose **Next**.

4. On the **Specify stack details** page, do the following:

   a. For **Stack name**, enter **`custom-oracle-iam`**.

   b. Choose **Next**.

5. On the **Configure stack options page**, choose **Next**.

6. On the **Review custom-oracle-iam** page, do the following:

   a. Select the **I acknowledge that CloudFormation might create IAM resources with custom names** check box.

   b. Choose **Submit**.

   CloudFormation creates the IAM roles that RDS Custom for Oracle requires. In the left panel, when **custom-oracle-iam** shows **CREATE_COMPLETE**, proceed to the next step.

7. In the left panel, choose **custom-oracle-iam**. In the right panel, do the following:

   a. Choose **Stack info**. Your stack has an ID in the format **arn:aws:cloudformation:*region*:*account-no*:stack/custom-oracle-iam/*identifier***.

   b. Choose **Resources**. You should see the following:

      - An instance profile named **AWSRDSCustomInstanceProfile-*region***
      - A service role named **AWSRDSCustomInstanceRole-*region***

      When you create your RDS Custom DB instance, you need to supply the instance profile ID.

Create your IAM role and instance profile manually

Configuration is easiest when you use CloudFormation. However, you can also configure IAM manually. For manual setup, do the following:

- Step 1: Create the IAM role AWSRDSCustomInstanceRoleForRdsCustomInstance.
- Step 2: Add an access policy to AWSRDSCustomInstanceRoleForRdsCustomInstance.
- Step 2: Add an access policy to AWSRDSCustomInstanceRoleForRdsCustomInstance.
- Step 4: Add AWSRDSCustomInstanceRoleForRdsCustomInstance to AWSRDSCustomInstanceProfile.

Step 1: Create the IAM role AWSRDSCustomInstanceRoleForRdsCustomInstance

In this step, you create the role using the naming format `AWSRDSCustomInstanceRole-`*`region`*. Using the trust policy, Amazon EC2 can assume the role. The following example assumes that you have set the environment variable `$REGION` to the AWS Region in which you want to create your DB instance.

```
aws iam create-role \
  --role-name AWSRDSCustomInstanceRole-$REGION \
  --assume-role-policy-document '{
    "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "sts:AssumeRole",
          "Effect": "Allow",
          "Principal": {
              "Service": "ec2.amazonaws.com"
          }
        }
      ]
    }'
```

Step 2: Add an access policy to AWSRDSCustomInstanceRoleForRdsCustomInstance

When you embed an inline policy in an IAM role, the inline policy is used as part of the role's access (permissions) policy. You create the `AWSRDSCustomIamRolePolicy` policy that permits Amazon EC2 to send and receive messages and perform various actions.

The following example creates the access policy named `AWSRDSCustomIamRolePolicy`, and adds it to the IAM role `AWSRDSCustomInstanceRole-`*`region`*. This example assumes that you have set the following environment variables:

`$REGION`

Set this variable to the AWS Region in which you plan to create your DB instance.

`$ACCOUNT_ID`

Set this variable to your AWS account number.

$KMS_KEY

Set this variable to the Amazon Resource Name (ARN) of the AWS KMS key that you want to use for your RDS Custom DB instances. To specify more than one KMS key, add it to the `Resources` section of statement ID (Sid) 11.

```
aws iam put-role-policy \
  --role-name AWSRDSCustomInstanceRole-$REGION \
  --policy-name AWSRDSCustomIamRolePolicy \
  --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Action": [
                "ssm:DescribeAssociation",
                "ssm:GetDeployablePatchSnapshotForInstance",
                "ssm:GetDocument",
                "ssm:DescribeDocument",
                "ssm:GetManifest",
                "ssm:GetParameter",
                "ssm:GetParameters",
                "ssm:ListAssociations",
                "ssm:ListInstanceAssociations",
                "ssm:PutInventory",
                "ssm:PutComplianceItems",
                "ssm:PutConfigurePackageResult",
                "ssm:UpdateAssociationStatus",
                "ssm:UpdateInstanceAssociationStatus",
                "ssm:UpdateInstanceInformation",
                "ssm:GetConnectionStatus",
                "ssm:DescribeInstanceInformation",
                "ssmmessages:CreateControlChannel",
                "ssmmessages:CreateDataChannel",
                "ssmmessages:OpenControlChannel",
                "ssmmessages:OpenDataChannel"
            ],
            "Resource": [
                "*"
            ]
        },
```

```
        {
            "Sid": "2",
            "Effect": "Allow",
            "Action": [
                "ec2messages:AcknowledgeMessage",
                "ec2messages:DeleteMessage",
                "ec2messages:FailMessage",
                "ec2messages:GetEndpoint",
                "ec2messages:GetMessages",
                "ec2messages:SendReply"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "3",
            "Effect": "Allow",
            "Action": [
                "logs:PutRetentionPolicy",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams",
                "logs:DescribeLogGroups",
                "logs:CreateLogStream",
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws:logs:'$REGION':'$ACCOUNT_ID':log-group:rds-custom-instance*"
            ]
        },
        {
            "Sid": "4",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::do-not-delete-rds-custom-*/*"
            ]
        },
        {
            "Sid": "5",
```

```
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "RDSCustomForOracle/Agent"
                    ]
                }
            }
        },
        {
            "Sid": "6",
            "Effect": "Allow",
            "Action": [
                "events:PutEvents"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "7",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:'$REGION':'$ACCOUNT_ID':secret:do-not-delete-
rds-custom-*",
                "arn:aws:secretsmanager:'$REGION':'$ACCOUNT_ID':secret:rds-custom!
oracle-do-not-delete-*"
            ]
        },
        {
           "Sid": "8",
           "Effect": "Allow",
           "Action": [
             "s3:ListBucketVersions"
```

```
            ],
            "Resource": [
              "arn:aws:s3:::do-not-delete-rds-custom-*"
            ]
          },
          {
             "Sid": "9",
             "Effect": "Allow",
             "Action": "ec2:CreateSnapshots",
             "Resource": [
                 "arn:aws:ec2:*:*:instance/*",
                 "arn:aws:ec2:*:*:volume/*"
             ],
             "Condition": {
                 "StringEquals": {
                     "ec2:ResourceTag/AWSRDSCustom": "custom-oracle"
                 }
             }
           },
           {
             "Sid": "10",
             "Effect": "Allow",
             "Action": "ec2:CreateSnapshots",
             "Resource": [
                 "arn:aws:ec2:*::snapshot/*"
             ]
           },
           {
             "Sid": "11",
             "Effect": "Allow",
             "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
             ],
             "Resource": [
               "arn:aws:kms:'$REGION':'$ACCOUNT_ID':key/'$KMS_KEY'"
             ]
           },
           {
             "Sid": "12",
             "Effect": "Allow",
             "Action": "ec2:CreateTags",
             "Resource": "*",
             "Condition": {
```

```
                "StringLike": {
                    "ec2:CreateAction": [
                        "CreateSnapshots"
                    ]
                }
            }
        },
        {
            "Sid": "13",
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage",
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueUrl"
            ],
            "Resource": "arn:aws:sqs:'$REGION':'$ACCOUNT_Id':do-not-delete-rds-custom-
*",
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/AWSRDSCustom": "custom-oracle"
                }
            }
        }
    ]
}'
```

Step 3: Create the RDS Custom instance profile AWSRDSCustomInstanceProfile

An instance profile is a container that includes a single IAM role. RDS Custom uses the instance profile to pass the role to the instance.

If you use the CLI to create a role, you create the role and instance profile as separate actions, with potentially different names. Create your IAM instance profile as follows, naming it using the format `AWSRDSCustomInstanceProfile-`*`region`*. The following example assumes that you have set the environment variable `$REGION` to the AWS Region in which you want to create your DB instance.

```
aws iam create-instance-profile \
    --instance-profile-name AWSRDSCustomInstanceProfile-$REGION
```

Step 4: Add AWSRDSCustomInstanceRoleForRdsCustomInstance to AWSRDSCustomInstanceProfile

Add your IAM role to the instance profile that you previously created. The following example assumes that you have set the environment variable `$REGION` to the AWS Region in which you want to create your DB instance.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name AWSRDSCustomInstanceProfile-$REGION \
    --role-name AWSRDSCustomInstanceRole-$REGION
```

Step 5: Grant required permissions to your IAM user or role

Make sure that the IAM principal (user or role) that creates the CEV or RDS Custom DB instance has either of the following policies:

- The `AdministratorAccess` policy
- The `AmazonRDSFullAccess` policy with required permissions for Amazon S3 and AWS KMS, CEV creation, and DB instance creation

Topics

- IAM permissions required for Amazon S3 and AWS KMS
- IAM permissions required for creating a CEV
- IAM permissions required for creating a DB instance from a CEV

IAM permissions required for Amazon S3 and AWS KMS

To create CEVs or RDS Custom for Oracle DB instances, your IAM principal needs to access Amazon S3 and AWS KMS. The following sample JSON policy grants the required permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateS3Bucket",
            "Effect": "Allow",
```

```
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketPolicy",
                "s3:PutBucketObjectLockConfiguration",
                "s3:PutBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::do-not-delete-rds-custom-*"
        },
        {
            "Sid": "CreateKmsGrant",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        }
    ]
}
```

For more information about the `kms:CreateGrant` permission, see AWS KMS key management.

IAM permissions required for creating a CEV

To create a CEV, your IAM principal needs the following additional permissions:

```
s3:GetObjectAcl
s3:GetObject
s3:GetObjectTagging
s3:ListBucket
mediaimport:CreateDatabaseBinarySnapshot
```

The following sample JSON policy grants the additional permissions necessary to access bucket *`my-custom-installation-files`* and its contents.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "AccessToS3MediaBucket",
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectAcl",
                "s3:GetObject",
                "s3:GetObjectTagging",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::my-custom-installation-files",
                "arn:aws:s3:::my-custom-installation-files/*"
            ]
        },
        {
            "Sid": "PermissionForByom",
            "Effect": "Allow",
            "Action": [
                "mediaimport:CreateDatabaseBinarySnapshot"
            ],
            "Resource": "*"
        }
    ]
}
```

You can grant similar permissions for Amazon S3 to caller accounts using an S3 bucket policy.

IAM permissions required for creating a DB instance from a CEV

To create an RDS Custom for Oracle DB instance from an existing CEV, the IAM principal needs the following additional permissions.

```
iam:SimulatePrincipalPolicy
cloudtrail:CreateTrail
cloudtrail:StartLogging
```

The following sample JSON policy grants the permissions necessary to validate an IAM role and log information to an AWS CloudTrail.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ValidateIamRole",
            "Effect": "Allow",
            "Action": "iam:SimulatePrincipalPolicy",
            "Resource": "*"
        },
        {
            "Sid": "CreateCloudTrail",
            "Effect": "Allow",
            "Action": [
                "cloudtrail:CreateTrail",
                "cloudtrail:StartLogging"
            ],
            "Resource": "arn:aws:cloudtrail:*:*:trail/do-not-delete-rds-custom-*"
        }
    ]
}
```

Step 6: Configure your VPC for RDS Custom for Oracle

Your RDS Custom DB instance is in a virtual private cloud (VPC) based on the Amazon VPC service, just like an Amazon EC2 instance or Amazon RDS instance. You provide and configure your own VPC. Unlike RDS Custom for SQL Server, RDS Custom for Oracle doesn't create an access control list or security groups. You must attach you own security group, subnets, and route tables.

You can configure your virtual private cloud (VPC) using either CloudFormation or a manual process.

> **Important**
>
> We strongly recommend that you configure your RDS Custom for Oracle environment using CloudFormation. This technique is the easiest and least error-prone.

Topics

- Configure your VPC using CloudFormation (recommended)
- Configure your VPC manually for RDS Custom for Oracle

Configure your VPC using CloudFormation (recommended)

If you've already configured your VPC for a different RDS Custom engine, and want to reuse the existing VPC, skip this step. This section assumes the following:

- You've already used CloudFormation to create your IAM instance profile and role.
- You know your route table ID.

  For a DB instance to be private, it must be in a private subnet. For a subnet to be private, it must not be associated with a route table that has a default internet gateway. For more information, see Configure route tables in the *Amazon VPC User Guide*.

When you use the CloudFormation template for your VPC, it creates the following resources:

- A private VPC
- A subnet group named `rds-custom-private`
- The following VPC endpoints, which your DB instance uses to communicate with dependent AWS services:
  - `com.amazonaws.region.ec2messages`
  - `com.amazonaws.region.events`
  - `com.amazonaws.region.logs`
  - `com.amazonaws.region.monitoring`
  - `com.amazonaws.region.s3`
  - `com.amazonaws.region.secretsmanager`
  - `com.amazonaws.region.ssm`
  - `com.amazonaws.region.ssmmessages`

  If you're creating a Multi-AZ deployment:

  - `com.amazonaws.region.sqs`

> **Note**
>
> For a complex networking setup with existing accounts, we recommend that you configure access to dependent services manually if access doesn't already exist. For more information, see Make sure your VPC can access dependent AWS services.

To configure your VPC using CloudFormation

1. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. Start the Create Stack wizard, and choose **Create Stack** and then **With new resources (standard)**.
3. On the **Create stack** page, do the following:
   a. For **Prepare template**, choose **Template is ready**.
   b. For **Template source**, choose **Upload a template file**.
   c. For **Choose file**, navigate to, then choose `custom-vpc.json`.
   d. Choose **Next**.
4. On the **Specify stack details** page, do the following:
   a. For **Stack name**, enter **`custom-vpc`**.
   b. For **Parameters**, choose the private subnets to use for RDS Custom DB instances.
   c. Choose the private VPC ID to use for RDS Custom DB instances.
   d. Enter the route table associated with the private subnets.
   e. Choose **Next**.
5. On the **Configure stack options page**, choose **Next**.
6. On the **Review custom-vpc** page, choose **Submit**.

   CloudFormation configures your private VPC. In the left panel, when **custom-vpc** shows **CREATE_COMPLETE**, proceed to the next step.
7. (Optional) Review the details of your VPC. In the **Stacks** pane, choose **custom-vpc**. In the right pane, do the following:
   a. Choose **Stack info**. Your stack has an ID in the format **arn:aws:cloudformation:*region*:*account-no*:stack/custom-vpc/*identifier***.
   b. Choose **Resources**. You should see a subnet group named **rds-custom-private** and several VPC endpoints that use the naming format **vpce-*string***. Each endpoint corresponds to an AWS service that RDS Custom needs to communicate with. For more information, see Make sure your VPC can access dependent AWS services.
   c. Choose **Parameters**. You should see the private subnets, private VPC, and the route table that you specified when you created the stack. When you create a DB instance, you need to supply the VPC ID and subnet group.

Configure your VPC manually for RDS Custom for Oracle

As an alternative to automating VPC creation with CloudFormation, you can configure your VPC manually. This option might be best when you have a complex networking setup that uses existing resources.

Topics

- Make sure your VPC can access dependent AWS services
- Configure the instance metadata service

Make sure your VPC can access dependent AWS services

RDS Custom sends communication from your DB instance to other AWS services. Make sure the following services are accessible from the subnet in which you create your RDS Custom DB instances:

- Amazon CloudWatch (`com.amazonaws.`*`region`*`.monitoring`)
- Amazon CloudWatch Logs (`com.amazonaws.`*`region`*`.logs`)
- Amazon CloudWatch Events (`com.amazonaws.`*`region`*`.events`)
- Amazon EC2 (`com.amazonaws.`*`region`*`.ec2` and `com.amazonaws.`*`region`*`.ec2messages`)
- Amazon S3 (`com.amazonaws.`*`region`*`.s3`)
- AWS Secrets Manager (`com.amazonaws.`*`region`*`.secretsmanager`)
- AWS Systems Manager (`com.amazonaws.`*`region`*`.ssm` and `com.amazonaws.`*`region`*`.ssmmessages`)

If creating Multi-AZ deployments

- Amazon Simple Queue Service (`com.amazonaws.`*`region`*`.sqs`)

If RDS Custom can't communicate with the necessary services, it publishes the following events:

```
Database instance in incompatible-network. SSM Agent connection not available. Amazon
 RDS can't connect to the dependent AWS services.
```

```
Database instance in incompatible-network. Amazon RDS can't connect to dependent AWS
 services. Make sure port 443 (HTTPS) allows outbound connections, and try again.
 "Failed to connect to the following services: s3 events"
```

To avoid `incompatible-network` errors, make sure that VPC components involved in communication between your RDS Custom DB instance and AWS services satisfy the following requirements:

- The DB instance can make outbound connections on port 443 to other AWS services.
- The VPC allows incoming responses to requests originating from your RDS Custom DB instance.
- RDS Custom can correctly resolve the domain names of endpoints for each AWS service.

If you already configured a VPC for a different RDS Custom DB engine, you can reuse that VPC and skip this process.

Configure the instance metadata service

Make sure that your instance can do the following:

- Access the instance metadata service using Instance Metadata Service Version 2 (IMDSv2).
- Allow outbound communications through port 80 (HTTP) to the IMDS link IP address.
- Request instance metadata from `http://169.254.169.254`, the IMDSv2 link.

For more information, see Use IMDSv2 in the *Amazon EC2 User Guide*.

RDS Custom for Oracle automation uses IMDSv2 by default, by setting `HttpTokens=enabled` on the underlying Amazon EC2 instance. However, you can use IMDSv1 if you want. For more information, see Configure the instance metadata options in the *Amazon EC2 User Guide*.

Working with custom engine versions for Amazon RDS Custom for Oracle

A *custom engine version (CEV)* for Amazon RDS Custom for Oracle is a binary volume snapshot of a database engine and specific Amazon Machine Image (AMI). By default, RDS Custom for Oracle uses the latest available AMI managed by RDS Custom, but you can specify an AMI that was used in a previous CEV. You store your database installation files in Amazon S3. RDS Custom uses the installation files and the AMI to create your CEV for you.

Topics

- Preparing to create a CEV
- Creating a CEV
- Modifying CEV status
- Viewing CEV details for Amazon RDS Custom for Oracle
- Deleting a CEV

Preparing to create a CEV

To create a CEV, access the installation files and patches that are stored in your Amazon S3 bucket for any of the following releases:

- Oracle Database 19c
- Oracle Database 18c
- Oracle Database 12c Release 2 (12.2)
- Oracle Database 12c Release 1 (12.1)

For example, you can use the April 2021 RU/RUR for Oracle Database 19c, or any valid combination of installation files and patches. For more information on the versions and Regions supported by RDS Custom for Oracle, see RDS Custom with RDS for Oracle.

Topics

- Step 1 (Optional): Download the manifest templates
- Step 2: Download your database installation files and patches from Oracle Software Delivery Cloud
- Step 3: Upload your installation files to Amazon S3

- Step 4 (Optional): Share your installation media in S3 across AWS accounts
- Step 5: Prepare the CEV manifest
- Step 6 (Optional): Validate the CEV manifest
- Step 7: Add necessary IAM permissions

Step 1 (Optional): Download the manifest templates

A *CEV manifest* is a JSON document that includes the list of database installation .zip files for your CEV. To create a CEV, do the following:

1. Identify the Oracle database installation files that you want to include in your CEV.
2. Download the installation files.
3. Create a JSON manifest that lists the installation files.

RDS Custom for Oracle provides JSON manifest templates with our recommended .zip files for each supported Oracle Database release. For example, the following template is for the 19.17.0.0.0 RU.

```
{
    "mediaImportTemplateVersion": "2020-08-14",
    "databaseInstallationFileNames": [
        "V982063-01.zip"
    ],
    "opatchFileNames": [
        "p6880880_190000_Linux-x86-64.zip"
    ],
    "psuRuPatchFileNames": [
        "p34419443_190000_Linux-x86-64.zip",
        "p34411846_190000_Linux-x86-64.zip"
    ],
    "otherPatchFileNames": [
        "p28852325_190000_Linux-x86-64.zip",
        "p29997937_190000_Linux-x86-64.zip",
        "p31335037_190000_Linux-x86-64.zip",
        "p32327201_190000_Linux-x86-64.zip",
        "p33613829_190000_Linux-x86-64.zip",
        "p34006614_190000_Linux-x86-64.zip",
        "p34533061_190000_Linux-x86-64.zip",
        "p34533150_190000_Generic.zip",
```

```
        "p28730253_190000_Linux-x86-64.zip",
        "p29213893_1917000DBRU_Generic.zip",
        "p33125873_1917000DBRU_Linux-x86-64.zip",
        "p34446152_1917000DBRU_Linux-x86-64.zip"
    ]
}
```

Each template has an associated readme that includes instructions for downloading the patches, URLs for the .zip files, and file checksums. You can use these templates as they are or modify them with your own patches. To review the templates, download custom-oracle-manifest.zip to your local disk and then open it with a file archiving application. For more information, see Step 5: Prepare the CEV manifest.

Step 2: Download your database installation files and patches from Oracle Software Delivery Cloud

When you have identified the installation files that you want for your CEV, download them to your local system. The Oracle Database installation files and patches are hosted on Oracle Software Delivery Cloud. Each CEV requires a base release, such as Oracle Database 19c or Oracle Database 12c Release 2 (12.2), and an optional list of patches.

To download the database installation files for Oracle Database

1. Go to https://edelivery.oracle.com/ and sign in.
2. In the search box, enter **`Oracle Database Enterprise Edition`** or **`Oracle Database Standard Edition 2`** and choose **Search**.
3. Choose one of the following base releases:

Database version	Enterprise Edition	Standard Edition 2
Oracle Database 19c	**DLP: Oracle Database 19c Enterprise Edition 19.3.0.0.0 (Oracle Database Enterprise Edition)**	**DLP: Oracle Database 19c Standard Edition 2 19.3.0.0.0 (Oracle Database Standard Edition 2)**
Oracle Database 18c	**DLP: Oracle Database 18c Enterprise Edition 18.0.0.0.0 (Oracle Database Enterprise Edition)**	**DLP: Oracle Database Standard Edition 2 18.0.0.0.0 (Oracle Database Standard Edition 2)**

Database version	Enterprise Edition	Standard Edition 2
Oracle Database 12c Release 2 (12.2.0.1)	**DLP: Oracle Database 12c Enterprise Edition 12.2.0.1.0 (Oracle Database Enterprise Edition)**	**DLP: Oracle Database Standard Edition 2 12.2.0.1.0 (Oracle Database Standard Edition 2)**
Oracle Database 12c Release 1 (12.1.0.2)	**DLP: Oracle Database 12c Enterprise Edition 12.1.0.2.0 (Oracle Database Enterprise Edition)**	**DLP: Oracle Database Standard Edition 2 12.1.0.2.0 (Oracle Database Standard Edition 2)**

4. Choose **Continue**.
5. Clear the **Download Queue** check box.
6. Choose the option that corresponds to your base release:
   - **Oracle Database 19.3.0.0.0 - Long Term Release**.
   - **Oracle Database 18.0.0.0.0**
   - **Oracle Database 12.2.0.1.0**.
   - **Oracle Database 12.1.0.2.0**.
7. Choose **Linux x86-64** in **Platform/Languages**.
8. Choose **Continue**, and then sign the Oracle License Agreement.
9. Choose the .zip file that corresponds to your database release:

Database release and edition	Zip files	SHA-256 hash
19c EE and SE2	**V982063-01.zip**	`BA8329C757133DA313ED3B6D7F86C5AC42CD9970A28BF2E6233F3235233AA8D8`
18c EE and SE2	**V978967-01.zip**	`C96A4FD768787AF98272008833FE10B172691CF84E42816B138C12D4DE63AB96`

Database release and edition	Zip files	SHA-256 hash
12.2.0.1 EE and SE2	**V839960-01.zip**	96ED97D21F15C1AC0CCE3749DA6C3DAC7059BB60672D76B008103FC754D22DDE
12.1.0.2 EE	**V46095-01_1of2.zip** **V46095-01_2of2.zip**	31FDC2AF41687B4E547A3A18F796424D8C1AF36406D2160F65B0AF6A9CD47355 for V46095-01_1of2.zip 03DA14F5E875304B28F0F3BB02AF0EC33227885B99C9865DF70749D1E220ACCD for V46095-01_2of2.zip
12.1.0.2 SE2	**V77388-01_1of2.zip** **V77388-01_2of2.zip**	73873369753230F5A0921F95ACEADB591388CB06ED72A7F3AEA7BCBCEA2403BC for V77388-01_1of2.zip 2492E1BE1E3E3531DA83D0843C09C08E435AC8CEFD9A00C0DF56BE4F15CEEBF3 for V77388-01_2of2.zip

10. Download your desired Oracle patches from `updates.oracle.com` or `support.oracle.com` to your local system. You can find the URLs for the patches in the following locations:

    - The readme files in the .zip file that you downloaded in Step 1 (Optional): Download the manifest templates
    - The patches listed in each Release Update (RU) in Release notes for Amazon Relational Database Service (Amazon RDS) for Oracle

Step 3: Upload your installation files to Amazon S3

Upload your Oracle installation and patch files to Amazon S3 using the AWS CLI. The S3 bucket that contains your installation files must be in the same AWS Region as your CEV.

Examples in this section use the following placeholders:

- *install-or-patch-file.zip* – Oracle installation media file. For example, p32126828_190000_Linux-x86-64.zip is a patch.
- *amzn-s3-demo-destination-bucket* – Your Amazon S3 bucket designated for your uploaded installation files.
- *123456789012/cev1* – An optional prefix in your Amazon S3 bucket.
- *amzn-s3-demo-source-bucket* – An Amazon S3 bucket where you can optionally stage files.

Topics

- Step 3a: Verify that your S3 bucket is in the correct AWS Region
- Step 3b: Make sure that your S3 bucket policy has the correct permissions
- Step 3c: Upload your files using the cp or sync commands
- Step 3d: List the files in your S3 bucket

Step 3a: Verify that your S3 bucket is in the correct AWS Region

Verify that your S3 bucket is in the AWS Region where you plan to run the `create-custom-db-engine-version` command.

```
aws s3api get-bucket-location --bucket amzn-s3-demo-destination-bucket
```

Step 3b: Make sure that your S3 bucket policy has the correct permissions

You can create a CEV from scratch or from a source CEV. If you plan to create new CEV from source CEVs, make sure that your S3 bucket policy has the correct permissions:

1. Identify the S3 bucket reserved by RDS Custom. The bucket name has the format `do-not-delete-rds-custom-account-region-string`. For example, the bucket might be named `do-not-delete-rds-custom-123456789012-us-east-1-abc123EXAMPLE`.
2. Make sure that the following permission is appended to your S3 bucket policy. Replace `do-not-delete-rds-custom-123456789012-us-east-1-abc123EXAMPLE` with the name of your bucket.

```
{
    "Sid": "AWSRDSCustomForOracleCustomEngineVersionGetObject",
```

```
    "Effect": "Allow",
    "Principal": {
        "Service": "custom.rds.amazonaws.com"
    },
    "Action": [
        "s3:GetObject",
        "s3:GetObjectTagging"
    ],
    "Resource": "arn:aws:s3:::do-not-delete-rds-custom-123456789012-us-
east-1-abc123EXAMPLE/CustomEngineVersions/*"
}, ...
```

Step 3c: Upload your files using the cp or sync commands

Choose either of the following options:

- Use `aws s3 cp` to upload a single .zip file.

  Upload each installation .zip file separately. Don't combine the .zip files into a single .zip file.
- Use `aws s3 sync` to upload a directory.

Example

The following example uploads *install-or-patch-file.zip* to the *123456789012/cev1* folder in the RDS Custom Amazon S3 bucket. Run a separate `aws s3` command for each .zip that you want to upload.

For Linux, macOS, or Unix:

```
aws s3 cp install-or-patch-file.zip \
    s3://amzn-s3-demo-destination-bucket/123456789012/cev1/
```

For Windows:

```
aws s3 cp install-or-patch-file.zip ^
    s3://amzn-s3-demo-destination-bucket/123456789012/cev1/
```

Example

The following example uploads the files in your local *cev1* folder to the *123456789012/cev1* folder in your Amazon S3 bucket.

For Linux, macOS, or Unix:

```
aws s3 sync cev1 \
    s3://amzn-s3-demo-destination-bucket/123456789012/cev1/
```

For Windows:

```
aws s3 sync cev1 ^
    s3://amzn-s3-demo-destination-bucket/123456789012/cev1/
```

Example

The following example uploads all files in *amzn-s3-demo-source-bucket* to the *123456789012/cev1* folder in your Amazon S3 bucket.

For Linux, macOS, or Unix:

```
aws s3 sync s3://amzn-s3-demo-source-bucket/ \
    s3://amzn-s3-demo-destination-bucket/123456789012/cev1/
```

For Windows:

```
aws s3 sync s3://amzn-s3-demo-source-bucket/ ^
    s3://amzn-s3-demo-destination-bucket/123456789012/cev1/
```

Step 3d: List the files in your S3 bucket

The following example uses the `s3 ls` command to list the files in your RDS Custom Amazon S3 bucket.

```
aws s3 ls \
    s3://amzn-s3-demo-destination-bucket/123456789012/cev1/
```

Step 4 (Optional): Share your installation media in S3 across AWS accounts

For the purposes of this section, the Amazon S3 bucket that contains your uploaded Oracle installation files is your *media bucket*. Your organization might use multiple AWS accounts in an AWS Region. If so, you might want to use one AWS account to populate your media bucket and a different AWS account to create CEVs. If you don't intend to share your media bucket, skip to the next section.

This section assumes the following:

- You can access the account that created your media bucket and a different account in which you intend to create CEVs.
- You intend to create CEVs in only one AWS Region. If you intend to use multiple Regions, create a media bucket in each Region.
- You're using the CLI. If you're using the Amazon S3 console, adapt the following steps.

To configure your media bucket for sharing across AWS accounts

1. Log in to the AWS account that contains the S3 bucket into which you uploaded your installation media.
2. Start with either a blank JSON policy template or an existing policy that you can adapt.

   The following command retrieves an existing policy and saves it as *my-policy.json*. In this example, the S3 bucket containing your installation files is named *amzn-s3-demo-bucket*.

   ```
   aws s3api get-bucket-policy \
       --bucket amzn-s3-demo-bucket \
       --query Policy \
       --output text > my-policy.json
   ```

3. Edit the media bucket permissions as follows:

   - In the `Resource` element of your template, specify the S3 bucket into which you uploaded your Oracle Database installation files.
   - In the `Principal` element, specify the ARNs for all AWS accounts that you intend to use to create CEVs. You can add the root, a user, or a role to the S3 bucket allow list. For more information, see IAM identifiers in the *AWS Identity and Access Management User Guide*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "GrantAccountsAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:root",
                    "arn:aws:iam::444455556666:user/user-name-with-path",
                    "arn:aws:iam::123456789012:role/role-name-with-path"
                ]
            },
            "Action": [
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

4. Attach the policy to your media bucket.

   In the following example, *amzn-s3-demo-bucket* is the name of the S3 bucket that contains your installation files, and *my-policy.json* is the name of your JSON file.

   ```
   aws s3api put-bucket-policy \
       --bucket amzn-s3-demo-bucket \
       --policy file://my-policy.json
   ```

5. Log in to an AWS account in which you intend to create CEVs.

6. Verify that this account can access the media bucket in the AWS account that created it.

```
aws s3 ls --query "Buckets[].Name"
```

   For more information, see aws s3 ls in the *AWS CLI Command Reference*.

7. Create a CEV by following the steps in Creating a CEV.

Step 5: Prepare the CEV manifest

A CEV manifest is a JSON document that includes the following:

- (Required) The list of installation .zip files that you uploaded to Amazon S3. RDS Custom applies the patches in the order in which they're listed in the manifest.
- (Optional) Installation parameters that set nondefault values for the Oracle base, Oracle home, and the ID and name of the UNIX/Linux user and group. Be aware that you can't modify the installation parameters for an existing CEV or an existing DB instance. You also can't upgrade from one CEV to another CEV when the installation parameters have different settings.

For sample CEV manifests, see the JSON templates that you downloaded in Step 1 (Optional): Download the manifest templates. You can also review the samples in CEV manifest examples.

Topics

- JSON fields in the CEV manifest
- Creating the CEV manifest
- CEV manifest examples

JSON fields in the CEV manifest

The following table describes the JSON fields in the manifest.

JSON field	Description
`MediaImportTemplateVersion`	Version of the CEV manifest. The date is in the format `YYYY-MM-DD` .

JSON field	Description
`databaseInstallationFileNames`	Ordered list of installation files for the database.
`opatchFileNames`	Ordered list of OPatch installers used for the Oracle DB engine. Only one value is valid. Values for `opatchFileNames` must start with `p6880880_`.
`psuRuPatchFileNames`	The PSU and RU patches for this database. **Important** If you include `psuRuPatchFileNames`, `opatchFileNames` is required. Values for `opatchFileNames` must start with `p6880880_`.
`OtherPatchFileNames`	The patches that aren't in the list of PSU and RU patches. RDS Custom applies these patches after applying the PSU and RU patches. **Important** If you include `OtherPatchFileNames`, `opatchFileNames` is required. Values for `opatchFileNames` must start with `p6880880_`.

JSON field	Description
`installationParameters`	Nondefault settings for the Oracle base, Oracle home, and the ID and name of the UNIX/Linux user and group. You can set the following parameters: **oracleBase** The directory under which your Oracle binaries are installed. It is the mount point of the binary volume that stores your files. The Oracle base directory can include multiple Oracle homes. For example, if `/home/oracle/oracle.19.0.0.0.ru-2020-04.rur-2020-04.r1.EE.1` is one of your Oracle home directories, then `/home/oracle` is the Oracle base directory. A user-specified Oracle base directory is not a symbolic link. If you don't specify the Oracle base, the default directory is `/rdsdbbin`. **oracleHome** The directory in which your Oracle database binaries are installed. For example, if you specify `/home/oracle/` as your Oracle base, then you might specify `/home/oracle/oracle.19.0.0.0.ru-2020-04.rur-2020-04.r1.EE.1/` as your Oracle home. A user-specified Oracle home directory is not a symbolic link. The Oracle home value is referenced by the `$ORACLE_HOME` environment variable. If you don't specify the Oracle home, the default naming format is `/rdsdbbin/oracle.` *`major-engine-version`* `.custom.r1.` *`engine-edition`* `.1`. **unixUname** The name of the UNIX user that owns the Oracle software. RDS Custom assumes this user when running local database commands. If you specify both `unixUid` and `unixUname`, RDS Custom creates the user if it doesn't exist, and then

JSON field	Description
	assigns the UID to the user if it's not the same as the initial UID. The default user name is `rdsdb`. **unixUid** The ID (UID) of the UNIX user that owns the Oracle software. If you specify both `unixUid` and `unixUname` , RDS Custom creates the user if it doesn't exist, and then assigns the UID to the user if it's not the same as the initial UID. The default UID is `61001`. This is the UID of the user `rdsdb`. **unixGroupName** The name of the UNIX group. The UNIX user that owns the Oracle software belongs to this group. The default group name is `rdsdb`. **unixGroupId** The ID of the UNIX group to which the UNIX user belongs. The default group ID is `1000`. This is the ID of the group `rdsdb`.

Each Oracle Database release has a different list of supported installation files. When you create your CEV manifest, make sure to specify only files that are supported by RDS Custom for Oracle. Otherwise, CEV creation fails with an error. All patches listed in Release notes for Amazon Relational Database Service (Amazon RDS) for Oracle are supported.

Creating the CEV manifest

To create a CEV manifest

1. List all installation files that you plan to apply, in the order that you want to apply them.
2. Correlate the installation files with the JSON fields described in JSON fields in the CEV manifest.

3. Do either of the following:

   - Create the CEV manifest as a JSON text file.
   - Edit the CEV manifest template when you create the CEV in the console. For more information, see Creating a CEV.

CEV manifest examples

The following examples show CEV manifest files for different Oracle Database releases. If you include a JSON field in your manifest, make sure that it isn't empty. For example, the following CEV manifest isn't valid because `otherPatchFileNames` is empty.

```
{
    "mediaImportTemplateVersion": "2020-08-14",
    "databaseInstallationFileNames": [
        "V982063-01.zip"
    ],
    "opatchFileNames": [
        "p6880880_190000_Linux-x86-64.zip"
    ],
    "psuRuPatchFileNames": [
        "p32126828_190000_Linux-x86-64.zip"
    ],
    "otherPatchFileNames": [
    ]
}
```

Topics

- Sample CEV manifest for Oracle Database 12c Release 1 (12.1)
- Sample CEV manifest for Oracle Database 12c Release 2 (12.2)
- Sample CEV manifest for Oracle Database 18c
- Sample CEV manifest for Oracle Database 19c

Example Sample CEV manifest for Oracle Database 12c Release 1 (12.1)

In the following example for the July 2021 PSU for Oracle Database 12c Release 1 (12.1), RDS Custom applies the patches in the order specified. Thus, RDS Custom applies p32768233, then

p32876425, then p18759211, and so on. The example sets new values for the UNIX user and group, and the Oracle home and Oracle base.

```
{
    "mediaImportTemplateVersion":"2020-08-14",
    "databaseInstallationFileNames":[
        "V46095-01_1of2.zip",
        "V46095-01_2of2.zip"
    ],
    "opatchFileNames":[
        "p6880880_121010_Linux-x86-64.zip"
    ],
    "psuRuPatchFileNames":[
        "p32768233_121020_Linux-x86-64.zip"
    ],
    "otherPatchFileNames":[
        "p32876425_121020_Linux-x86-64.zip",
        "p18759211_121020_Linux-x86-64.zip",
        "p19396455_121020_Linux-x86-64.zip",
        "p20875898_121020_Linux-x86-64.zip",
        "p22037014_121020_Linux-x86-64.zip",
        "p22873635_121020_Linux-x86-64.zip",
        "p23614158_121020_Linux-x86-64.zip",
        "p24701840_121020_Linux-x86-64.zip",
        "p25881255_121020_Linux-x86-64.zip",
        "p27015449_121020_Linux-x86-64.zip",
        "p28125601_121020_Linux-x86-64.zip",
        "p28852325_121020_Linux-x86-64.zip",
        "p29997937_121020_Linux-x86-64.zip",
        "p31335037_121020_Linux-x86-64.zip",
        "p32327201_121020_Linux-x86-64.zip",
        "p32327208_121020_Generic.zip",
        "p17969866_12102210119_Linux-x86-64.zip",
        "p20394750_12102210119_Linux-x86-64.zip",
        "p24835919_121020_Linux-x86-64.zip",
        "p23262847_12102201020_Linux-x86-64.zip",
        "p21171382_12102201020_Generic.zip",
        "p21091901_12102210720_Linux-x86-64.zip",
        "p33013352_12102210720_Linux-x86-64.zip",
        "p25031502_12102210720_Linux-x86-64.zip",
        "p23711335_12102191015_Generic.zip",
        "p19504946_121020_Linux-x86-64.zip"
    ],
```

```
    "installationParameters": {
        "unixGroupName": "dba",
        "unixGroupId": 12345,
        "unixUname": "oracle",
        "unixUid": 12345,
        "oracleHome": "/home/oracle/oracle.12.1.0.2",
        "oracleBase": "/home/oracle"
    }
}
```

Example Sample CEV manifest for Oracle Database 12c Release 2 (12.2)

In following example for the October 2021 PSU for Oracle Database 12c Release 2 (12.2), RDS Custom applies p33261817, then p33192662, then p29213893, and so on. The example sets new values for the UNIX user and group, and the Oracle home and Oracle base.

```
{
    "mediaImportTemplateVersion":"2020-08-14",
    "databaseInstallationFileNames":[
        "V839960-01.zip"
    ],
    "opatchFileNames":[
        "p6880880_122010_Linux-x86-64.zip"
    ],
    "psuRuPatchFileNames":[
        "p33261817_122010_Linux-x86-64.zip"
    ],
    "otherPatchFileNames":[
        "p33192662_122010_Linux-x86-64.zip",
        "p29213893_122010_Generic.zip",
        "p28730253_122010_Linux-x86-64.zip",
        "p26352615_12201211019DBOCT2021RU_Linux-x86-64.zip",
        "p23614158_122010_Linux-x86-64.zip",
        "p24701840_122010_Linux-x86-64.zip",
        "p25173124_122010_Linux-x86-64.zip",
        "p25881255_122010_Linux-x86-64.zip",
        "p27015449_122010_Linux-x86-64.zip",
        "p28125601_122010_Linux-x86-64.zip",
        "p28852325_122010_Linux-x86-64.zip",
        "p29997937_122010_Linux-x86-64.zip",
        "p31335037_122010_Linux-x86-64.zip",
        "p32327201_122010_Linux-x86-64.zip",
        "p32327208_122010_Generic.zip"
```

```
    ],
    "installationParameters": {
        "unixGroupName": "dba",
        "unixGroupId": 12345,
        "unixUname": "oracle",
        "unixUid": 12345,
        "oracleHome": "/home/oracle/oracle.12.2.0.1",
        "oracleBase": "/home/oracle"
    }
}
```

Example Sample CEV manifest for Oracle Database 18c

In following example for the October 2021 PSU for Oracle Database 18c, RDS Custom applies p32126855, then p28730253, then p27539475, and so on. The example sets new values for the UNIX user and group, and the Oracle home and Oracle base.

```
{
    "mediaImportTemplateVersion":"2020-08-14",
    "databaseInstallationFileNames":[
        "V978967-01.zip"
    ],
    "opatchFileNames":[
        "p6880880_180000_Linux-x86-64.zip"
    ],
    "psuRuPatchFileNames":[
        "p32126855_180000_Linux-x86-64.zip"
    ],
    "otherPatchFileNames":[
        "p28730253_180000_Linux-x86-64.zip",
        "p27539475_1813000DBRU_Linux-x86-64.zip",
        "p29213893_180000_Generic.zip",
        "p29374604_1813000DBRU_Linux-x86-64.zip",
        "p29782284_180000_Generic.zip",
        "p28125601_180000_Linux-x86-64.zip",
        "p28852325_180000_Linux-x86-64.zip",
        "p29997937_180000_Linux-x86-64.zip",
        "p31335037_180000_Linux-x86-64.zip",
        "p31335142_180000_Generic.zip"
    ]
    "installationParameters": {
        "unixGroupName": "dba",
        "unixGroupId": 12345,
```

```
        "unixUname": "oracle",
        "unixUid": 12345,
        "oracleHome": "/home/oracle/18.0.0.0.ru-2020-10.rur-2020-10.r1",
        "oracleBase": "/home/oracle/"
    }
}
```

Example Sample CEV manifest for Oracle Database 19c

In the following example for Oracle Database 19c, RDS Custom applies p32126828, then p29213893, then p29782284, and so on. The example sets new values for the UNIX user and group, and the Oracle home and Oracle base.

```
{
    "mediaImportTemplateVersion": "2020-08-14",
    "databaseInstallationFileNames": [
        "V982063-01.zip"
    ],
    "opatchFileNames": [
        "p6880880_190000_Linux-x86-64.zip"
    ],
    "psuRuPatchFileNames": [
        "p32126828_190000_Linux-x86-64.zip"
    ],
    "otherPatchFileNames": [
        "p29213893_1910000DBRU_Generic.zip",
        "p29782284_1910000DBRU_Generic.zip",
        "p28730253_190000_Linux-x86-64.zip",
        "p29374604_1910000DBRU_Linux-x86-64.zip",
        "p28852325_190000_Linux-x86-64.zip",
        "p29997937_190000_Linux-x86-64.zip",
        "p31335037_190000_Linux-x86-64.zip",
        "p31335142_190000_Generic.zip"
    ],
    "installationParameters": {
        "unixGroupName": "dba",
        "unixGroupId": 12345,
        "unixUname": "oracle",
        "unixUid": 12345,
        "oracleHome": "/home/oracle/oracle.19.0.0.0.ru-2020-04.rur-2020-04.r1.EE.1",
        "oracleBase": "/home/oracle"
    }
}
```

Step 6 (Optional): Validate the CEV manifest

Optionally, verify that manifest is a valid JSON file by running the `json.tool` Python script. For example, if you change into the directory containing a CEV manifest named `manifest.json`, run the following command.

```
python -m json.tool < manifest.json
```

Step 7: Add necessary IAM permissions

Make sure that the IAM principal that creates the CEV has the necessary policies described in Step 5: Grant required permissions to your IAM user or role.

Creating a CEV

You can create a CEV using the AWS Management Console or the AWS CLI. Specify either the multitenant or non-multitenant architecture. For more information, see Multitenant architecture considerations.

Typically, creating a CEV takes about two hours. After you have created the CEV, you can use it to create or upgrade an RDS Custom DB instance. For more information, see Creating an RDS Custom for Oracle DB instance and Upgrading an RDS Custom for Oracle DB instance.

> **Note**
>
> If your DB instance currently uses Oracle Linux 7.9, create a new CEV that uses the latest AMI, which uses Oracle Linux 8. Then modify your instance to use the new CEV.

Note the following requirements and limitations for creating a CEV:

- The Amazon S3 bucket containing your installation files must be in the same AWS Region as your CEV. Otherwise, the creation process fails.
- The CEV name must be in the format *`major-engine-version`*`.`*`customized_string`*, as in `19.cdb_cev1`.
- The CEV name must contain 1–50 alphanumeric characters, underscores, dashes, or periods.
- The CEV name can't contain consecutive periods, as in `19..cdb_cev1`.

Console

To create a CEV

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Custom engine versions**.

   The **Custom engine versions** page shows all CEVs that currently exist. If you haven't created any CEVs, the page is empty.
3. Choose **Create custom engine version**.
4. In **Engine options**, do the following:

   a. For **Engine type**, choose **Oracle**.

   b. For **Architecture settings**, optionally choose **Multitenant architecture** to create an Oracle multitenant CEV, which uses the DB engine `custom-oracle-ee-cdb` or `custom-oracle-se2-cdb`. You can create an RDS Custom for Oracle CDB with a Multitenant CEV only. If you don't choose this option, your CEV is a non-CDB, which uses the engine `custom-oracle-ee` or `custom-oracle-se2`.

   > **Note**
   >
   > The architecture that you choose is a permanent characteristic of your CEV. You can't modify your CEV to use a different architecture later.

   c. Choose either of the following options:

   - **Create new CEV** – Create a CEV from scratch. In this case, you must specify a JSON manifest specifying the database binaries.
   - **Create CEV from source** – In **Specify the CEV that you want to copy**, choose an existing CEV to use as the source CEV. In this case, you can specify a new Amazon Machine Image (AMI), but you can't specify different database binaries.

   d. For **Engine version**, choose the major engine version.
5. In **Version details**, do the following:

   a. Enter a valid name in **Custom engine version name**. For example, you might enter the name **`19.cdb_cev1`**.

b. (Optional) Enter a description for your CEV.

6. In **Installation media**, do the following:

   a. (Optional) For **AMI ID**, either leave the field blank to use the latest service-provided AMI, or enter an AMI that you previously used to create a CEV. To obtain valid AMI IDs, use either of the following techniques:

      - In the console, choose **Custom engine versions** in the left navigation pane, and choose the name of a CEV. The AMI ID used by the CEV appears in the **Configuration** tab.
      - In the AWS CLI, use the `describe-db-engine-versions` command. Search the output for `ImageID`.

   b. For **S3 location of manifest files**, enter the location of the Amazon S3 bucket that you specified in Step 3: Upload your installation files to Amazon S3. For example, enter **`s3://my-custom-installation-files/123456789012/cev1/`**.

      > **Note**
      >
      > The AWS Region in which you create the CEV must be in the same Region as the S3 bucket.

   c. (Create new CEV only) For **CEV manifest**, enter the JSON manifest that you created in Creating the CEV manifest.

7. In the **KMS key** section, select **Enter a key ARN** to list the available AWS KMS keys. Then select your KMS key from the list.

   An AWS KMS key is required for RDS Custom. For more information, see Step 1: Create or reuse a symmetric encryption AWS KMS key.

8. (Optional) Choose **Add new tag** to create a key-value pair for your CEV.

9. Choose **Create custom engine version**.

   If the JSON manifest is in an invalid format, the console displays **Error validating the CEV manifest**. Fix the problems, and try again.

The **Custom engine versions** page appears. Your CEV is shown with the status **Creating**. The process to create a CEV takes approximately two hours.

AWS CLI

To create a CEV by using the AWS CLI, run the create-custom-db-engine-version command.

The following options are required:

- `--engine` – Specify the engine type. For a CDB, specify either `custom-oracle-ee-cdb` or `custom-oracle-se2-cdb`. For a non-CDB, specify either `custom-oracle-ee` or `custom-oracle-se2`. You can create CDBs only from a CEV created with `custom-oracle-ee-cdb` or `custom-oracle-se2-cdb`. You can create non-CDBs only from a CEV created with `custom-oracle-ee` or `custom-oracle-se2`.
- `--engine-version` – Specify the engine version. The format is *`major-engine-version.customized_string`*. The CEV name must contain 1–50 alphanumeric characters, underscores, dashes, or periods. The CEV name can't contain consecutive periods, as in `19..cdb_cev1`.
- `--kms-key-id` – Specify an AWS KMS key.
- `--manifest` – Specify either *`manifest_json_string`* or `--manifest file:`*`file_name`*. Newline characters aren't permitted in *`manifest_json_string`*. Make sure to escape double quotes (") in the JSON code by prefixing them with a backslash (\).

  The following example shows the *`manifest_json_string`* for 19c from Step 5: Prepare the CEV manifest. The example sets new values for the Oracle base, Oracle home, and the ID and name of the UNIX/Linux user and group. If you copy this string, remove all newline characters before you paste it into your command.

```
"{\"mediaImportTemplateVersion\": \"2020-08-14\",
\"databaseInstallationFileNames\": [\"V982063-01.zip\"],
\"opatchFileNames\": [\"p6880880_190000_Linux-x86-64.zip\"],
\"psuRuPatchFileNames\": [\"p32126828_190000_Linux-x86-64.zip\"],
\"otherPatchFileNames\": [\"p29213893_1910000DBRU_Generic.zip\",
\"p29782284_1910000DBRU_Generic.zip\",\"p28730253_190000_Linux-
x86-64.zip\",\"p29374604_1910000DBRU_Linux-x86-64.zip\",
\"p28852325_190000_Linux-x86-64.zip\",\"p29997937_190000_Linux-x86-64.zip
\",\"p31335037_190000_Linux-x86-64.zip\",\"p31335142_190000_Generic.zip
\"]\"installationParameters\":{ \"unixGroupName\":\"dba\",
\ \"unixUname\":\"oracle\", \ \"oracleHome\":\"/home/oracle/
oracle.19.0.0.0.ru-2020-04.rur-2020-04.r1.EE.1\", \ \"oracleBase\":\"/
home/oracle/\"}}"
```

- `--database-installation-files-s3-bucket-name` – Specify the same bucket name that you specified in Step 3: Upload your installation files to Amazon S3. The AWS Region in which you run `create-custom-db-engine-version` must be the same Region as your Amazon S3 bucket.

You can also specify the following options:

- `--description` – Specify a description of your CEV.
- `--database-installation-files-s3-prefix` – Specify the folder name that you specified in Step 3: Upload your installation files to Amazon S3.
- `--image-id` – Specify an AMI ID that want to reuse. To find valid IDs, run the `describe-db-engine-versions` command, and then search the output for `ImageID`. By default, RDS Custom for Oracle uses the most recent available AMI.

The following example creates an Oracle multitenant CEV named `19.cdb_cev1`. The example reuses an existing AMI rather than use the latest available AMI. Make sure that the name of your CEV starts with the major engine version number.

Example

For Linux, macOS, or Unix:

```
aws rds create-custom-db-engine-version \
    --engine custom-oracle-se2-cdb \
    --engine-version 19.cdb_cev1 \
    --database-installation-files-s3-bucket-name us-east-1-123456789012-custom-
installation-files \
    --database-installation-files-s3-prefix 123456789012/cev1 \
    --kms-key-id my-kms-key \
    --description "test cev" \
    --manifest manifest_string \
    --image-id ami-012a345678901bcde
```

For Windows:

```
aws rds create-custom-db-engine-version ^
    --engine custom-oracle-se2-cdb ^
    --engine-version 19.cdb_cev1 ^
```

```
    --database-installation-files-s3-bucket-name us-east-1-123456789012-custom-
installation-files ^
    --database-installation-files-s3-prefix 123456789012/cev1 ^
    --kms-key-id my-kms-key ^
    --description "test cev" ^
    --manifest manifest_string ^
    --image-id ami-012a345678901bcde
```

Example

Get details about your CEV by using the `describe-db-engine-versions` command.

```
aws rds describe-db-engine-versions \
    --engine custom-oracle-se2-cdb \
    --include-all
```

The following partial sample output shows the engine, parameter groups, manifest, and other information.

```
{
    "DBEngineVersions": [
        {
            "Engine": "custom-oracle-se2-cdb",
            "EngineVersion": "19.cdb_cev1",
            "DBParameterGroupFamily": "custom-oracle-se2-cdb-19",
            "DBEngineDescription": "Containerized Database for Oracle Custom SE2",
            "DBEngineVersionDescription": "test cev",
            "Image": {
                "ImageId": "ami-012a345678901bcde",
                "Status": "active"
            },
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "MajorEngineVersion": "19",
            "DatabaseInstallationFilesS3BucketName": "us-east-1-123456789012-custom-
installation-files",
            "DatabaseInstallationFilesS3Prefix": "123456789012/cev1",
```

```
            "DBEngineVersionArn": "arn:aws:rds:us-east-1:123456789012:cev:custom-
oracle-se2-cdb/19.cdb_cev1/abcd12e3-4f5g-67h8-i9j0-k1234l56m789",
            "KMSKeyId": "arn:aws:kms:us-
east-1:732027699161:key/1ab2345c-6d78-9ef0-1gh2-3456i7j89k01",
            "CreateTime": "2023-03-07T19:47:58.131000+00:00",
            "TagList": [],
            "SupportsBabelfish": false,
...
```

Failure to create a CEV

If the process to create a CEV fails, RDS Custom issues `RDS-EVENT-0198` with the message `Creation failed for custom engine version` *`major-engine-version.cev_name`*, and includes details about the failure. For example, the event prints missing files.

You can't modify a failed CEV. You can only delete it, then try again to create a CEV after fixing the causes of the failure. For information about troubleshooting the reasons for CEV creation failure, see Troubleshooting custom engine version creation for RDS Custom for Oracle.

Modifying CEV status

You can modify a CEV using the AWS Management Console or the AWS CLI. You can modify the CEV description or its availability status. Your CEV has one of the following status values:

- `available` – You can use this CEV to create a new RDS Custom DB instance or upgrade a DB instance. This is the default status for a newly created CEV.
- `inactive` – You can't create or upgrade an RDS Custom instance with this CEV. You can't restore a DB snapshot to create a new RDS Custom DB instance with this CEV.

You can change the CEV from any supported status to any other supported status. You might change status to prevent the accidental use of a CEV or make a discontinued CEV eligible for use again. For example, you might change the status of your CEV from `available` to `inactive`, and from `inactive` back to `available`.

Console

To modify a CEV

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Custom engine versions**.
3. Choose a CEV whose description or status you want to modify.
4. For **Actions**, choose **Modify**.
5. Make any of the following changes:
   - For **CEV status settings**, choose a new availability status.
   - For **Version description**, enter a new description.
6. Choose **Modify CEV**.

   If the CEV is in use, the console displays **You can't modify the CEV status**. Fix the problems, and try again.

The **Custom engine versions** page appears.

AWS CLI

To modify a CEV by using the AWS CLI, run the modify-custom-db-engine-version command. You can find CEVs to modify by running the describe-db-engine-versions command.

The following options are required:

- `--engine` *`engine-type`*, where *`engine-type`* is `custom-oracle-ee`, `custom-oracle-se2`, `custom-oracle-ee-cdb`, or `custom-oracle-se2-cdb`
- `--engine-version` *`cev`*, where *`cev`* is the name of the custom engine version that you want to modify
- `--status` *`status`*, where *`status`* is the availability status that you want to assign to the CEV

The following example changes a CEV named `19.my_cev1` from its current status to `inactive`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-custom-db-engine-version \
    --engine custom-oracle-se2 \
    --engine-version 19.my_cev1 \
    --status inactive
```

For Windows:

```
aws rds modify-custom-db-engine-version ^
    --engine custom-oracle-se2 ^
    --engine-version 19.my_cev1 ^
    --status inactive
```

Viewing CEV details for Amazon RDS Custom for Oracle

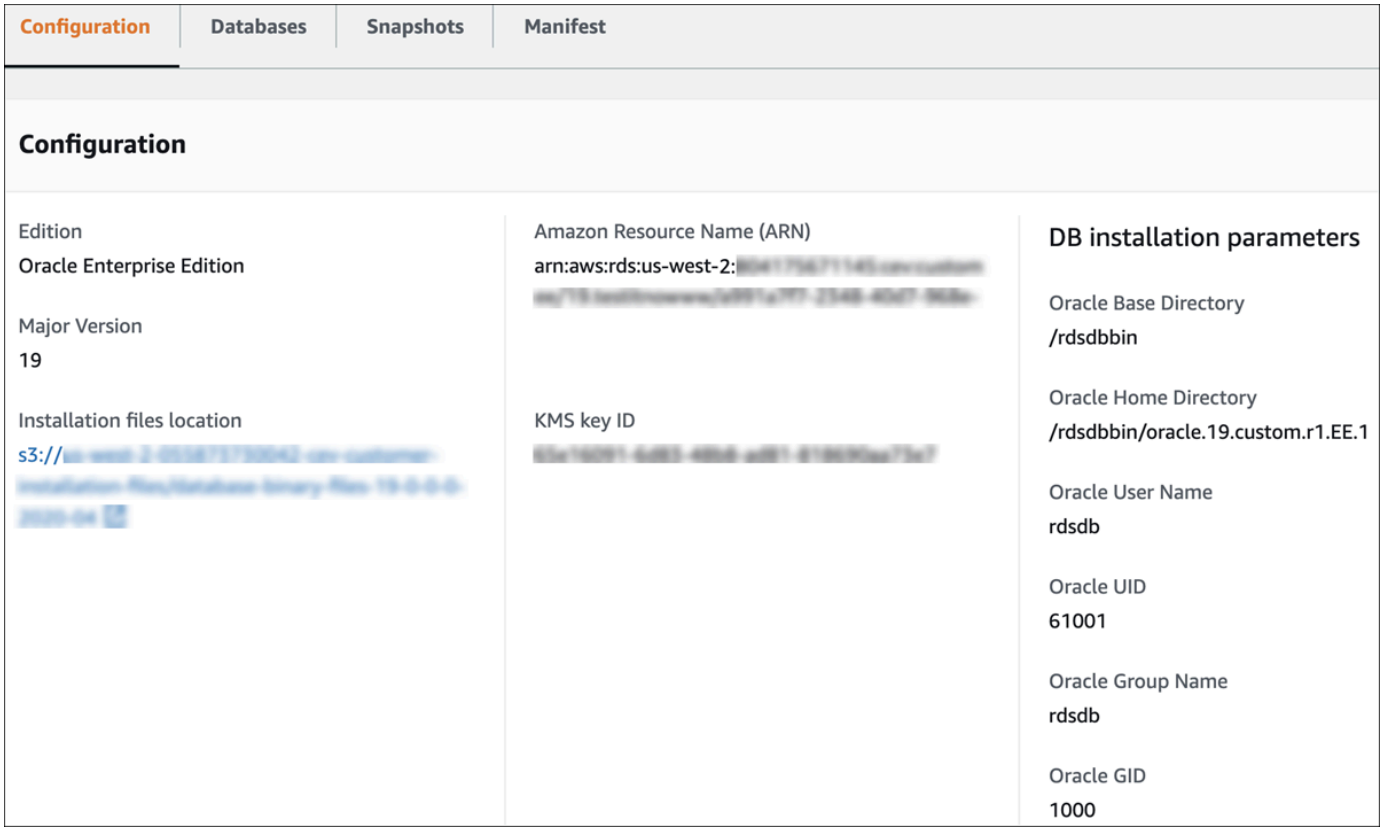
You can view details about your CEV manifest and the command used to create your CEV by using the AWS Management Console or the AWS CLI.

Console

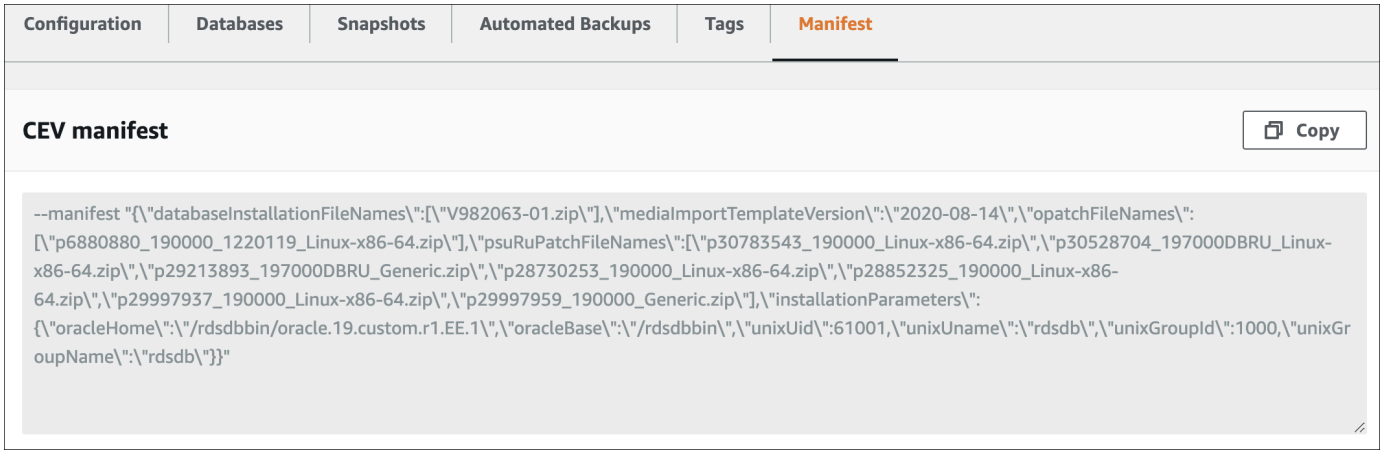
To view CEV details

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Custom engine versions**.

   The **Custom engine versions** page shows all CEVs that currently exist. If you haven't created any CEVs, the page is empty.
3. Choose the name of the CEV that you want to view.
4. Choose **Configuration** to view the installation parameters specified in your manifest.



5. Choose **Manifest** to view the installation parameters specified in the `--manifest` option of the `create-custom-db-engine-version` command. You can copy this text, replace values as needed, and use them in a new command.



AWS CLI

To view details about a CEV by using the AWS CLI, run the describe-db-engine-versions command.

The following options are required:

- `--engine` *`engine-type`*, where *`engine-type`* is `custom-oracle-ee`, `custom-oracle-se2`, `custom-oracle-ee-cdb`, or `custom-oracle-se2-cdb`
- `--engine-version` *`major-engine-version`*`.`*`customized_string`*

The following example creates a non-CDB CEV that uses Enterprise Edition. The CEV name `19.my_cev1` starts with the major engine version number, which is required.

Example

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
    --engine custom-oracle-ee \
    --engine-version 19.my_cev1
```

For Windows:

```
aws rds describe-db-engine-versions ^
    --engine custom-oracle-ee ^
    --engine-version 19.my_cev1
```

The following partial sample output shows the engine, parameter groups, manifest, and other information.

```
"DBEngineVersions": [
    {
        "Engine": "custom-oracle-ee",
        "MajorEngineVersion": "19",
        "EngineVersion": "19.my_cev1",
        "DatabaseInstallationFilesS3BucketName": "us-east-1-123456789012-cev-customer-
installation-files",
        "DatabaseInstallationFilesS3Prefix": "123456789012/cev1",
        "CustomDBEngineVersionManifest": "{\n\"mediaImportTemplateVersion\":
 \"2020-08-14\",\n\"databaseInstallationFileNames\": [\n\"V982063-01.zip\"\n],
\n\"installationParameters\": {\n\"oracleBase\":\"/tmp\",\n\"oracleHome\":\"/
tmp/Oracle\"\n},\n\"opatchFileNames\": [\n\"p6880880_190000_Linux-x86-64.zip
\"\n],\n\"psuRuPatchFileNames\": [\n\"p32126828_190000_Linux-x86-64.zip
\"\n],\n\"otherPatchFileNames\": [\n\"p29213893_1910000DBRU_Generic.zip\",\n
\"p29782284_1910000DBRU_Generic.zip\",\n\"p28730253_190000_Linux-x86-64.zip\",\n
\"p29374604_1910000DBRU_Linux-x86-64.zip\",\n\"p28852325_190000_Linux-x86-64.zip\",
```

```
\n\"p29997937_190000_Linux-x86-64.zip\",\n\"p31335037_190000_Linux-x86-64.zip\",\n
\"p31335142_190000_Generic.zip\"\n]\n}\n",
        "DBParameterGroupFamily": "custom-oracle-ee-19",
        "DBEngineDescription": "Oracle Database server EE for RDS Custom",
        "DBEngineVersionArn": "arn:aws:rds:us-west-2:123456789012:cev:custom-oracle-
ee/19.my_cev1/0a123b45-6c78-901d-23e4-5678f901fg23",
        "DBEngineVersionDescription": "test",
        "KMSKeyId": "arn:aws:kms:us-east-1:123456789012:key/ab1c2de3-f4g5-6789-h012-
h3ijk4567l89",
        "CreateTime": "2022-11-18T09:17:07.693000+00:00",
        "ValidUpgradeTarget": [
        {
            "Engine": "custom-oracle-ee",
            "EngineVersion": "19.cev.2021-01.09",
            "Description": "test",
            "AutoUpgrade": false,
            "IsMajorVersionUpgrade": false
        }
]
```


Deleting a CEV

You can delete a CEV using the AWS Management Console or the AWS CLI. Typically, deletion takes a few minutes.

To delete a CEV, it can't be in use by any of the following:

- An RDS Custom DB instance
- A snapshot of an RDS Custom DB instance
- An automated backup of your RDS Custom DB instance

Console

To delete a CEV

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Custom engine versions**.
3. Choose a CEV whose description or status you want to delete.
4. For **Actions**, choose **Delete**.

The **Delete *cev_name*?** dialog box appears.

5. Enter **`delete me`**, and then choose **Delete**.

   In the **Custom engine versions** page, the banner shows that your CEV is being deleted.

AWS CLI

To delete a CEV by using the AWS CLI, run the delete-custom-db-engine-version command.

The following options are required:

- `--engine` *engine-type*, where *engine-type* is `custom-oracle-ee`, `custom-oracle-se2`, `custom-oracle-ee-cdb`, or `custom-oracle-se2-cdb`
- `--engine-version` *cev*, where *cev* is the name of the custom engine version to be deleted

The following example deletes a CEV named `19.my_cev1`.

Example

For Linux, macOS, or Unix:

```
aws rds delete-custom-db-engine-version \
    --engine custom-oracle-ee \
    --engine-version 19.my_cev1
```

For Windows:

```
aws rds delete-custom-db-engine-version ^
    --engine custom-oracle-ee ^
    --engine-version 19.my_cev1
```

Configuring a DB instance for Amazon RDS Custom for Oracle

You can create an RDS Custom DB instance, and then connect to it using Secure Shell (SSH) or AWS Systems Manager.

For more information about connecting and logging in to a RDS Custom for Oracle DB instance, see the following topics.

- Connecting to your RDS Custom DB instance using Session Manager
- Connecting to your RDS Custom DB instance using SSH
- Logging in to your RDS Custom for Oracle database as SYS

Creating an RDS Custom for Oracle DB instance

Create an Amazon RDS Custom for Oracle DB instance using either the AWS Management Console or the AWS CLI. The procedure is similar to the procedure for creating an Amazon RDS DB instance. For more information, see Creating an Amazon RDS DB instance.

If you included installation parameters in your CEV manifest, then your DB instance uses the Oracle base, Oracle home, and the ID and name of the UNIX/Linux user and group that you specified. The `oratab` file, which is created by Oracle Database during installation, points to the real installation location rather than to a symbolic link. When RDS Custom for Oracle runs commands, it runs as the configured OS user rather than the default user `rdsdb`. For more information, see Step 5: Prepare the CEV manifest.

Before you attempt to create or connect to an RDS Custom DB instance, complete the tasks in Setting up your environment for Amazon RDS Custom for Oracle.

Console

To create an RDS Custom for Oracle DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose **Create database**.
4. In **Choose a database creation method**, select **Standard create**.

5. In the **Engine options** section, do the following:

   a. For **Engine type**, choose **Oracle**.

   b. For **Database management type**, choose **Amazon RDS Custom**.

   c. For **Architecture settings**, do one of the following:

   - Select **Multitenant architecture** to create a container database (CDB). At creation, your CDB contains one PDB seed and one initial PDB.

     > **Note**
     >
     > The **Multitenant architecture** setting is supported only for Oracle Database 19c.

   - Clear **Multitenant architecture** to create a non-CDB. A non-CDB can't contain PDBs.

   d. For **Edition**, choose **Oracle Enterprise Edition** or **Oracle Standard Edition 2**.

   e. For **Custom engine version**, choose an existing RDS Custom custom engine version (CEV). A CEV has the following format: *`major-engine-version.customized_string`*. An example identifier is `19.cdb_cev1`.

   If you chose **Multitenant architecture** in the previous step, you can only specify a CEV that uses the `custom-oracle-ee-cdb` or `custom-oracle-se2-cdb` engine type. The console filters out CEVs that were created with different engine types.

6. In **Templates**, choose **Production**.

7. In the **Settings** section, do the following:

   a. For **DB instance identifier**, enter a unique name for your DB instance.

   b. For **Master username**, enter a username. You can retrieve this value from the console later.

   When you connect to a non-CDB, the master user is the user for the non-CDB. When you connect to a CDB, the master user is the user for the PDB. To connect to the CDB root, log in to the host, start a SQL client, and create an administrative user with SQL commands.

   c. Clear **Auto generate a password**.

8. Choose a **DB instance class**.

   For supported classes, see DB instance class support for RDS Custom for Oracle.

9. In the **Storage** section, do the following:

a. For **Storage type**, choose an SSD type: io1, io2, gp2, or gp3. You have the following additional options:

- For io1, io2, or gp3, choose a rate for **Provisioned IOPS**. The default is 1000 for io1 and io2 and 12000 for gp3.
- For gp3, choose a rate for **Storage throughput**. The default is 500 MiBps.

b. For **Allocated storage**, choose a storage size. The default is 40 GiB.

10. For **Connectivity**, specify your **Virtual private cloud (VPC)**, **DB subnet group**, and **VPC security group (firewall)**.
11. For **RDS Custom security**, do the following:

a. For **IAM instance profile**, choose the instance profile for your RDS Custom for Oracle DB instance.

The IAM instance profile must begin with `AWSRDSCustom`, for example *`AWSRDSCustomInstanceProfileForRdsCustomInstance`*.

b. For **Encryption**, choose **Enter a key ARN** to list the available AWS KMS keys. Then choose your key from the list.

An AWS KMS key is required for RDS Custom. For more information, see Step 1: Create or reuse a symmetric encryption AWS KMS key.

12. For **Database options**, do the following:

a. (Optional) For **System ID (SID)**, enter a value for the Oracle SID, which is also the name of your CDB. The SID is the name of the Oracle database instance that manages your database files. In this context, the term "Oracle database instance" refers exclusively to the system global area (SGA) and Oracle background processes. If you don't specify a SID, the value defaults to **RDSCDB**.

b. (Optional) For **Initial database name**, enter a name. The default value is **ORCL**. In the multitenant architecture, the initial database name is the PDB name.

> **Note**
>
> The SID and PDB name must be different.

c. For **Option group**, choose an option group or accept the default.

> **Note**
>
> The only supported option for RDS Custom for Oracle is `Timezone`. For more information, see Oracle time zone.

d. For **Backup retention period** choose a value. You can't choose **0 days**.

e. For the remaining sections, specify your preferred RDS Custom DB instance settings. For information about each setting, see Settings for DB instances. The following settings don't appear in the console and aren't supported:

- **Processor features**
- **Storage autoscaling**
- **Password and Kerberos authentication** option in **Database authentication** (only **Password authentication** is supported)
- **Performance Insights**
- **Log exports**
- **Enable auto minor version upgrade**
- **Deletion protection**

13. Choose **Create database**.

> **Important**
>
> When you create an RDS Custom for Oracle DB instance, you might receive the following error: The service-linked role is in the process of being created. Try again later. If you do, wait a few minutes and then try again to create the DB instance.

The **View credential details** button appears on the **Databases** page.

To view the master user name and password for the RDS Custom DB instance, choose **View credential details**.

To connect to the DB instance as the master user, use the user name and password that appear.

> **Important**
>
> You can't view the master user password again in the console. If you don't record it, you might have to change it. To change the master user password after the RDS Custom DB instance is available, log in to the database and run an `ALTER USER` command. You can't reset the password using the **Modify** option in the console.

14. Choose **Databases** to view the list of RDS Custom DB instances.
15. Choose the RDS Custom DB instance that you just created.

    On the RDS console, the details for the new RDS Custom DB instance appear:

    - The DB instance has a status of **creating** until the RDS Custom DB instance is created and ready for use. When the state changes to **available**, you can connect to the DB instance. Depending on the instance class and storage allocated, it can take several minutes for the new DB instance to be available.
    - **Role** has the value **Instance (RDS Custom)**.
    - **RDS Custom automation mode** has the value **Full automation**. This setting means that the DB instance provides automatic monitoring and instance recovery.

AWS CLI

You create an RDS Custom DB instance by using the create-db-instance AWS CLI command.

The following options are required:

- `--db-instance-identifier`
- `--db-instance-class` (for a list of supported instance classes, see DB instance class support for RDS Custom for Oracle)
- `--engine` *`engine-type`*, where *`engine-type`* is `custom-oracle-ee`, `custom-oracle-se2`, `custom-oracle-ee-cdb`, or `custom-oracle-se2-cdb`
- `--engine-version` *`cev`* (where *`cev`* is the name of the custom engine version that you specified in Creating a CEV)
- `--kms-key-id` *`my-kms-key`*
- `--backup-retention-period` *`days`* (where *`days`* is a value greater than `0`)

- `--no-auto-minor-version-upgrade`
- `--custom-iam-instance-profile AWSRDSCustomInstanceProfile-`*`us-east-1`* (where *`region`* is the AWS Region where you are creating your DB instance)

The following example creates an RDS Custom DB instance named `my-cfo-cdb-instance`. The database is a CDB with the nondefault name *`MYCDB`*. The nondefault PDB name is *`MYPDB`*. The backup retention period is three days.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --engine custom-oracle-ee-cdb \
    --db-instance-identifier my-cfo-cdb-instance \
    --engine-version 19.cdb_cev1 \
    --db-name MYPDB \
    --db-system-id MYCDB \
    --allocated-storage 250 \
    --db-instance-class db.m5.xlarge \
    --db-subnet-group mydbsubnetgroup \
    --master-username myuser \
    --master-user-password mypassword \
    --backup-retention-period 3 \
    --port 8200 \
    --kms-key-id my-kms-key \
    --no-auto-minor-version-upgrade \
    --custom-iam-instance-profile AWSRDSCustomInstanceProfile-us-east-1
```

For Windows:

```
aws rds create-db-instance ^
    --engine custom-oracle-ee-cdb ^
    --db-instance-identifier my-cfo-cdb-instance ^
    --engine-version 19.cdb_cev1 ^
    --db-name MYPDB ^
    --db-system-id MYCDB ^
    --allocated-storage 250 ^
    --db-instance-class db.m5.xlarge ^
    --db-subnet-group mydbsubnetgroup ^
    --master-username myuser ^
```

```
    --master-user-password mypassword ^
    --backup-retention-period 3 ^
    --port 8200 ^
    --kms-key-id my-kms-key ^
    --no-auto-minor-version-upgrade ^
    --custom-iam-instance-profile AWSRDSCustomInstanceProfile-us-east-1
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Get details about your instance by using the `describe-db-instances` command.

Example

```
aws rds describe-db-instances --db-instance-identifier my-cfo-cdb-instance
```

The following partial output shows the engine, parameter groups, and other information.

```
        {
            "DBInstanceIdentifier": "my-cfo-cdb-instance",
            "DBInstanceClass": "db.m5.xlarge",
            "Engine": "custom-oracle-ee-cdb",
            "DBInstanceStatus": "available",
            "MasterUsername": "admin",
            "DBName": "MYPDB",
            "DBSystemID": "MYCDB",
            "Endpoint": {
                "Address": "my-cfo-cdb-instance.abcdefghijkl.us-
east-1.rds.amazonaws.com",
                "Port": 1521,
                "HostedZoneId": "A1B2CDEFGH34IJ"
            },
            "AllocatedStorage": 100,
            "InstanceCreateTime": "2023-04-12T18:52:16.353000+00:00",
            "PreferredBackupWindow": "08:46-09:16",
            "BackupRetentionPeriod": 7,
            "DBSecurityGroups": [],
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-0a1bcd2e",
```

```
                    "Status": "active"
                }
            ],
            "DBParameterGroups": [
                {
                    "DBParameterGroupName": "default.custom-oracle-ee-cdb-19",
                    "ParameterApplyStatus": "in-sync"
                }
            ],
...
```

Multitenant architecture considerations

If you create an Amazon RDS Custom for Oracle DB instance with the Oracle multitenant architecture (`custom-oracle-ee-cdb` or `custom-oracle-se2-cdb` engine type), your database is a container database (CDB). If you don't specify the Oracle multitenant architecture, your database is a traditional non-CDB that uses the `custom-oracle-ee` or `custom-oracle-se2` engine type. A non-CDB can't contain pluggable databases (PDBs). For more information, see Database architecture for Amazon RDS Custom for Oracle.

When you create an RDS Custom for Oracle CDB instance, consider the following:

- You can create a multitenant database only from an Oracle Database 19c CEV.
- You can create a CDB instance only if the CEV uses the `custom-oracle-ee-cdb` or `custom-oracle-se2-cdb` engine type.
- If you create a CDB instance using Standard Edition 2, the CDB can contain a maximum of 3 PDBs.
- By default, your CDB is named `RDSCDB`, which is also the name of the Oracle System ID (Oracle SID). You can choose a different name.
- You CDB contains only one initial PDB. The PDB name defaults to `ORCL`. You can choose a different name for your initial PDB, but the Oracle SID and the PDB name can't be the same.
- RDS Custom for Oracle doesn't supply APIs for PDBs. To create additional PDBs, use the Oracle SQL command `CREATE PLUGGABLE DATABASE`. RDS Custom for Oracle doesn't restrict the number of PDBs that you can create. In general, you are responsible for creating and managing PDBs, as in an on-premises deployment.
- You can't use RDS APIs to create, modify, and delete PDBs: you must use Oracle SQL statements. When you create a PDB using Oracle SQL, we recommend that you take a manual snapshot afterward in case you need to perform point-in-time recovery (PITR).

- You can't rename existing PDBs using Amazon RDS APIs. You also can't rename the CDB using the `modify-db-instance` command.
- The open mode for the CDB root is `READ WRITE` on the primary and `MOUNTED` on a mounted standby database. RDS Custom for Oracle attempts to open all PDBs when opening the CDB. If RDS Custom for Oracle can't open all PDBs, it issues the event `tenant database shutdown`.

RDS Custom service-linked role

A *service-linked role* gives Amazon RDS Custom access to resources in your AWS account. It makes using RDS Custom easier because you don't have to manually add the necessary permissions. RDS Custom defines the permissions of its service-linked roles, and unless defined otherwise, only RDS Custom can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy can't be attached to any other IAM entity.

When you create an RDS Custom DB instance, both the Amazon RDS and RDS Custom service-linked roles are created (if they don't already exist) and used. For more information, see Using service-linked roles for Amazon RDS.

The first time that you create an RDS Custom for Oracle DB instance, you might receive the following error: The service-linked role is in the process of being created. Try again later. If you do, wait a few minutes and then try again to create the DB instance.

Installing additional software components on your RDS Custom for Oracle DB instance

In a newly created DB instance, your database environment includes Oracle binaries, a database, and a database listener. You might want to install additional software on the host operating system of the DB instance. For example, you might want to install Oracle Application Express (APEX), the Oracle Enterprise Manager (OEM) agent, or the Guardium S-TAP agent. For guidelines and high-level instructions, see the detailed AWS blog post Install additional software components on Amazon RDS Custom for Oracle.

Connecting to your RDS Custom DB instance using Session Manager

After you create your RDS Custom DB instance, you can connect to it using AWS Systems Manager Session Manager. This is the preferred technique when your DB instance isn't publicly accessible.

Session Manager allows you to access Amazon EC2 instances through a browser-based shell or through the AWS CLI. For more information, see AWS Systems Manager Session Manager.

Console

To connect to your DB instance using Session Manager

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom DB instance to which you want to connect.
3. Choose **Configuration**.
4. Note the **Resource ID** for your DB instance. For example, the resource ID might be `db-ABCDEFGHIJKLMNOPQRS0123456`.
5. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
6. In the navigation pane, choose **Instances**.
7. Look for the name of your EC2 instance, and then click the instance ID associated with it. For example, the instance ID might be `i-abcdefghijklm01234`.
8. Choose **Connect**.
9. Choose **Session Manager**.
10. Choose **Connect**.

    A window opens for your session.

AWS CLI

You can connect to your RDS Custom DB instance using the AWS CLI. This technique requires the Session Manager plugin for the AWS CLI. To learn how to install the plugin, see Install the Session Manager plugin for the AWS CLI.

To find the DB resource ID of your RDS Custom DB instance, use `aws rds describe-db-instances`.

```
aws rds describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DbiResourceId]' \
    --output text
```

The following sample output shows the resource ID for your RDS Custom instance. The prefix is `db-`.

```
db-ABCDEFGHIJKLMNOPQRS0123456
```

To find the EC2 instance ID of your DB instance, use `aws ec2 describe-instances`. The following example uses db-ABCDEFGHIJKLMNOPQRS0123456 for the resource ID.

```
aws ec2 describe-instances \
    --filters "Name=tag:Name,Values=db-ABCDEFGHIJKLMNOPQRS0123456" \
    --output text \
    --query 'Reservations[*].Instances[*].InstanceId'
```

The following sample output shows the EC2 instance ID.

```
i-abcdefghijklm01234
```

Use the `aws ssm start-session` command, supplying the EC2 instance ID in the `--target` parameter.

```
aws ssm start-session --target "i-abcdefghijklm01234"
```

A successful connection looks like the following.

```
Starting session with SessionId: yourid-abcdefghijklm1234
[ssm-user@ip-123-45-67-89 bin]$
```

Connecting to your RDS Custom DB instance using SSH

The Secure Shell Protocol (SSH) is a network protocol that supports encrypted communication over an unsecured network. After you create your RDS Custom DB instance, you can connect to it using an ssh client. For more information, see Connecting to your Linux instance using SSH.

Your SSH connection technique depends on whether your DB instance is private, meaning that it doesn't accept connections from the public internet. In this case, you must use SSH tunneling to connect the ssh utility to your instance. This technique transports data with a dedicated data stream (tunnel) inside an existing SSH session. You can configure SSH tunneling using AWS Systems Manager.

> **Note**
>
> Various strategies are supported for accessing private instances. To learn how to connect an ssh client to private instances using bastion hosts, see Linux Bastion Hosts on AWS. To learn how to configure port forwarding, see Port Forwarding Using AWS Systems Manager Session Manager.

If your DB instance is in a public subnet and has the publicly available setting, then no SSH tunneling is required. You can connect with SSH just as would to a public Amazon EC2 instance.

To connect an ssh client to your DB instance, complete the following steps:

1. Step 1: Configure your DB instance to allow SSH connections
2. Step 2: Retrieve your SSH secret key and EC2 instance ID
3. Step 3: Connect to your EC2 instance using the ssh utility

Step 1: Configure your DB instance to allow SSH connections

To make sure that your DB instance can accept SSH connections, do the following:

- Make sure that your DB instance security group permits inbound connections on port 22 for TCP.

  To learn how to configure the security group for your DB instance, see Controlling access with security groups.
- If you don't plan to use SSH tunneling, make sure your DB instance resides in a public subnet and is publicly accessible.

  In the console, the relevant field is **Publicly accessible** on the **Connectivity & security** tab of the database details page. To check your settings in the CLI, run the following command:

  ```
  aws rds describe-db-instances \
  --query 'DBInstances[*].
  {DBInstanceIdentifier:DBInstanceIdentifier,PubliclyAccessible:PubliclyAccessible}' \
  --output table
  ```

  To change the accessibility settings for your DB instance, see Modifying an Amazon RDS DB instance.

Step 2: Retrieve your SSH secret key and EC2 instance ID

To connect to the DB instance using SSH, you need the SSH key pair associated with the instance. RDS Custom creates the SSH key pair on your behalf, using the naming convention `do-not-delete-rds-custom-ssh-privatekey-`*`resource_id`*`-`*`uuid`* or `rds-custom!oracle-do-not-delete-`*`resource_id`*`-`*`uuid`*`-ssh-privatekey`. AWS Secrets Manager stores your SSH private key as a secret.

Retrieve your SSH secret key using either AWS Management Console or the AWS CLI. If your instance has a public DNS, and you don't intend to use SSH tunneling, then also retrieve the DNS name. You specify the DNS name for public connections.

Console

To retrieve the secret SSH key

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom DB instance to which you want to connect.
3. Choose **Configuration**.
4. Note the **Resource ID** value. For example, the DB instance resource ID might be `db-ABCDEFGHIJKLMNOPQRS0123456`.
5. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
6. In the navigation pane, choose **Instances**.
7. Find the name of your EC2 instance, and choose the instance ID associated with it. For example, the EC2 instance ID might be `i-abcdefghijklm01234`.
8. In **Details**, find **Key pair name**. The pair name includes the DB instance resource ID. For example, the pair name might be `do-not-delete-rds-custom-ssh-privatekey-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c` or `rds-custom!oracle-do-not-delete-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-privatekey`.
9. If your EC2 instance is public, note the **Public IPv4 DNS**. For the example, the public Domain Name System (DNS) address might be `ec2-12-345-678-901.us-east-2.compute.amazonaws.com`.
10. Open the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
11. Choose the secret that has the same name as your key pair.

12. Choose **Retrieve secret value**.
13. Copy the SSH private key into a text file, and then save the file with the `.pem` extension. For example, save the file as `/tmp/do-not-delete-rds-custom-ssh-privatekey-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c.pem` or `/tmp/rds-custom!oracle-do-not-delete-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-privatekey.pem`.

AWS CLI

To retrieve the SSH private key and save it in a .pem file, you can use the AWS CLI.

1. Find the DB resource ID of your RDS Custom DB instance using `aws rds describe-db-instances`.

```
aws rds describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DbiResourceId]' \
    --output text
```

   The following sample output shows the resource ID for your RDS Custom instance. The prefix is `db-`.

```
db-ABCDEFGHIJKLMNOPQRS0123456
```

2. Find the EC2 instance ID of your DB instance using `aws ec2 describe-instances`. The following example uses `db-ABCDEFGHIJKLMNOPQRS0123456` for the resource ID.

```
aws ec2 describe-instances \
    --filters "Name=tag:Name,Values=db-ABCDEFGHIJKLMNOPQRS0123456" \
    --output text \
    --query 'Reservations[*].Instances[*].InstanceId'
```

   The following sample output shows the EC2 instance ID.

```
i-abcdefghijklm01234
```

3. To find the key name, specify the EC2 instance ID. The following example describes EC2 instance `i-0bdc4219e66944afa`.

```
aws ec2 describe-instances \
    --instance-ids i-0bdc4219e66944afa \
```

```
    --output text \
    --query 'Reservations[*].Instances[*].KeyName'
```

The following sample output shows the key name, which uses the naming format `do-not-delete-rds-custom-ssh-privatekey-`*`resource_id`*`-`*`uuid`* or `rds-custom!oracle-do-not-delete-`*`resource_id`*`-`*`uuid`*`-ssh-privatekey`.

```
do-not-delete-rds-custom-ssh-privatekey-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c
rds-custom!oracle-do-not-delete-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-privatekey
```

4. Save the private key in a .pem file named after the key using `aws secretsmanager`.

   The following example saves the key `do-not-delete-rds-custom-ssh-privatekey-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c` to a file in your `/tmp` directory.

```
aws secretsmanager get-secret-value \
    --secret-id do-not-delete-rds-custom-ssh-privatekey-db-
ABCDEFGHIJKLMNOPQRS0123456-0d726c \
    --query SecretString \
    --output text >/tmp/do-not-delete-rds-custom-ssh-privatekey-db-
ABCDEFGHIJKLMNOPQRS0123456-0d726c.pem
```

   The following example saves the key `rds-custom!oracle-do-not-delete-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-privatekey` to a file in your `/tmp` directory.

```
aws secretsmanager get-secret-value \
    --secret-id rds-custom!oracle-do-not-delete-db-
ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-privatekey \
    --query SecretString \
    --output text >/tmp/rds-custom!oracle-do-not-delete-db-
ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-privatekey.pem
```

Step 3: Connect to your EC2 instance using the ssh utility

Your connection technique depends on whether you are connecting to a private DB instance or connecting to a public instance. A private connection requires you to configure SSH tunneling through AWS Systems Manager.

To connect to an EC2 instance using the ssh utility

1. For private connections, modify your SSH configuration file to proxy commands to AWS Systems Manager Session Manager. For public connections, skip to Step 2.

   Add the following lines to `~/.ssh/config`. These lines proxy SSH commands for hosts whose names begin with `i-` or `mi-`.

   ```
   Host i-* mi-*
       ProxyCommand sh -c "aws ssm start-session --target %h --document-name AWS-
   StartSSHSession --parameters 'portNumber=%p'"
   ```

2. Change to the directory that contains your .pem file. Using `chmod`, set the permissions to `400`.

   The following example changes to the `/tmp` directory and sets permissions for .pem file `do-not-delete-rds-custom-ssh-privatekey-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c.pem`.

   ```
   cd /tmp
   chmod 400 do-not-delete-rds-custom-ssh-privatekey-db-
   ABCDEFGHIJKLMNOPQRS0123456-0d726c.pem
   ```

   The following example changes to the `/tmp` directory and sets permissions for .pem file `rds-custom!oracle-do-not-delete-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-privatekey.pem`.

   ```
   cd /tmp
   chmod 400 rds-custom!oracle-do-not-delete-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-
   privatekey.pem
   ```

3. Run the ssh utility, specifying the .pem file and either the public DNS name (for public connections) or the EC2 instance ID (for private connections). Log in as user `ec2-user`.

   The following example connects to a public instance using the DNS name `ec2-12-345-678-901.us-east-2.compute.amazonaws.com`.

   ```
   # .pem file using naming prefix do-not-delete
   ssh -i \
     "do-not-delete-rds-custom-ssh-privatekey-db-
   ABCDEFGHIJKLMNOPQRS0123456-0d726c.pem" \
     ec2-user@ec2-12-345-678-901.us-east-2.compute.amazonaws.com
   ```

```
# .pem file using naming prefix rds-custom!oracle-do-not-delete
ssh -i \
  "rds-custom!oracle-do-not-delete-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-
privatekey.pem" \
  ec2-user@ec2-12-345-678-901.us-east-2.compute.amazonaws.com
```

The following example connects to a private instance using the EC2 instance ID `i-0bdc4219e66944afa`.

```
# .pem file using naming prefix do-not-delete
ssh -i \
  "do-not-delete-rds-custom-ssh-privatekey-db-
ABCDEFGHIJKLMNOPQRS0123456-0d726c.pem" \
  ec2-user@i-0bdc4219e66944afa

# .pem file using naming prefix rds-custom!oracle-do-not-delete
ssh -i \
  "rds-custom!oracle-do-not-delete-db-ABCDEFGHIJKLMNOPQRS0123456-0d726c-ssh-
privatekey.pem" \
  ec2-user@i-0bdc4219e66944afa
```

Logging in to your RDS Custom for Oracle database as SYS

After you create your RDS Custom DB instance, you can log in to your Oracle database as user SYS, which gives you SYSDBA privileges. You have the following login options:

- Get the SYS password from Secrets Manager, and specify this password in your SQL client.
- Use OS authentication to log in to your database. In this case, you don't need a password.

Finding the SYS password for your RDS Custom for Oracle database

Your can log in to your Oracle database as SYS or SYSTEM or by specifying the master user name in an API call. The password for SYS and SYSTEM is stored in Secrets Manager.

The secret uses the naming format `do-not-delete-rds-custom-`*`resource_id`*`-`*`uuid`* or `rds-custom!oracle-do-not-delete-`*`resource_id`*`-`*`uuid`*. You can find the password using the AWS Management Console.

Console

To find the SYS password for your database in Secrets Manager

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the RDS console, complete the following steps:
   a. In the navigation pane, choose **Databases**.
   b. Choose the name of your RDS Custom for Oracle DB instance.
   c. Choose **Configuration**.
   d. Copy the value underneath **Resource ID**. For example, you resource ID might be **db-ABC12CDE3FGH4I5JKLMNO6PQR7**.
3. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
4. In the Secrets Manager console, complete the following steps:
   a. In the left navigation pane, choose **Secrets**.
   b. Filter the secrets by the resource ID that you copied in step 2.d.
   c. Choose the secret that uses the naming format **do-not-delete-rds-custom-*`resource_id`*-*`uuid`*** or **rds-custom!oracle-do-not-delete-*`resource_id`*-*`uuid`***. The *`resource_id`* is the resource ID that you copied in step 2.d.

      For example, if your resource ID is **db-ABC12CDE3FGH4I5JKLMNO6PQR7** and your UUID is **1234ab**, your secret is named **do-not-delete-rds-custom-db-ABC12CDE3FGH4I5JKLMNO6PQR7-1234ab** or **rds-custom!oracle-do-not-delete-db-ABC12CDE3FGH4I5JKLMNO6PQR7-1234ab**.
   d. In **Secret value**, choose **Retrieve secret value**.
   e. In **Key/value**, copy the value for **password**.
5. Install SQL*Plus on your DB instance and log in to your database as SYS. For more information, see Step 3: Connect your SQL client to an Oracle DB instance.

Logging in to your RDS Custom for Oracle database using OS authentication

The OS user `rdsdb` owns the Oracle database binaries. You can switch to the `rdsdb` user and log in to your RDS Custom for Oracle database without a password.

1. Connect to your DB instance with AWS Systems Manager. For more information, see Connecting to your RDS Custom DB instance using Session Manager.
2. Switch to the `rdsdb` user.

```
sudo su - rdsdb
```

3. Log in to your database using OS authentication. You can use `sqlplus / as sysdba` or the `sql` alias.

```
$ sqlplus / as sysdba

SQL*Plus: Release 21.0.0.0.0 - Production on Wed Apr 12 20:11:08 2023
Version 21.9.0.0.0

Copyright (c) 1982, 2020, Oracle.  All rights reserved.


Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.10.0.0.0
```

Alternatively, you can use the `sql` alias:

```
$ sql
```

Managing an Amazon RDS Custom for Oracle DB instance

Amazon RDS Custom supports a subset of the usual management tasks for Amazon RDS DB instances. Following, you can find instructions for the supported RDS Custom for Oracle management tasks using the AWS Management Console and the AWS CLI.

Topics

- Working with container databases (CDBs) in RDS Custom for Oracle
- Working with high availability features for RDS Custom for Oracle
- Customizing your RDS Custom environment
- Modifying your RDS Custom for Oracle DB instance
- Changing the character set of an RDS Custom for Oracle DB instance
- Setting the NLS_LANG value in RDS Custom for Oracle
- Support for Transparent Data Encryption
- Tagging RDS Custom for Oracle resources
- Deleting an RDS Custom for Oracle DB instance

Working with container databases (CDBs) in RDS Custom for Oracle

You can either create your RDS Custom for Oracle DB instance with the Oracle multitenant architecture (`custom-oracle-ee-cdb` or `custom-oracle-se2-cdb` engine type) or with the traditional non-CDB architecture (`custom-oracle-ee` or `custom-oracle-se2` engine type). When you create a container database (CDB), it contains one pluggable database (PDB) and one PDB seed. You can create additional PDBs manually using Oracle SQL.

PDB and CDB names

When you create an RDS Custom for Oracle CDB instance, you specify a name for the initial PDB. By default, your initial PDB is named `ORCL`. You can choose a different name.

By default, your CDB is named `RDSCDB`. You can choose a different name. The CDB name is also the name of your Oracle system identifier (SID), which uniquely identifies the memory and processes that manage your CDB. For more information about the Oracle SID, see Oracle System Identifier (SID) in *Oracle Database Concepts*.

You can't rename existing PDBs using Amazon RDS APIs. You also can't rename the CDB using the `modify-db-instance` command.

PDB management

In the RDS Custom for Oracle shared responsibility model, you are responsible for managing PDBs and creating any additional PDBs. RDS Custom doesn't restrict the number of PDBs. You can manually create, modify, and delete PDBs by connecting to the CDB root and running a SQL statement. Create PDBs on an Amazon EBS data volume to prevent the DB instance from going outside the support perimeter.

To modify your CDBs or PDBs, complete the following steps:

1. Pause automation to prevent interference with RDS Custom actions.
2. Modify your CDB or PDBs.
3. Back up any modified PDBs.
4. Resume RDS Custom automation.

Automatic recovery of the CDB root

RDS Custom keeps the CDB root open in the same way as it keeps a non-CDB open. If the state of the CDB root changes, the monitoring and recovery automation attempts to recover the CDB root to the desired state. You receive RDS event notifications when the root CDB is shut down (`RDS-EVENT-0004`) or restarted (`RDS-EVENT-0006`), similar to the non-CDB architecture. RDS Custom attempts to open all PDBs in `READ WRITE` mode at DB instance startup. If some PDBs can't be opened, RDS Custom publishes the following event: `tenant database shutdown`.

Working with high availability features for RDS Custom for Oracle

RDS Custom for Oracle provides built-in high availability through Multi-AZ deployments. Alternatively, you can use Oracle Data Guard as a customer-managed option based on your use cases.

Multi-AZ deployments (fully-managed)

With Multi-AZ deployments for RDS Custom for Oracle, Amazon RDS automatically provisions and maintains a synchronous standby replica in a different Availability Zone (AZ). The primary DB instance is synchronously replicated across AZs to a standby replica for data redundancy. Multi-AZ deployment is supported in both the Enterprise Edition and the Standard Edition 2. See the section called “Managing a Multi-AZ deployment for RDS Custom for Oracle” for details.

Oracle Data Guard (customer-managed)

Alternatively, you can achieve high availability by manually configuring Oracle Data Guard to replicate data between RDS Custom for Oracle DB instances. The primary DB instance automatically synchronizes data to the standby instances. Oracle Data Guard is supported only in the Enterprise Edition.

You can configure your high availability environment in the following ways:

- Configure standby instances in different AZs to be resilient to AZ failures.
- Place your standby databases in mounted or read-only mode. Read-only mode requires an Oracle Active Data Guard license.
- Fail over or switch over from the primary database to a standby database with no data loss.
- Migrate data by configuring high availability for your on-premises instance, and then failing over or switching over to the RDS Custom for Oracle standby database.

To learn how to configure Oracle Data Guard for high availability, see the AWS blog Build high availability for RDS Custom for Oracle using read replicas. You can perform the following tasks:

- Use a virtual private network (VPN) tunnel to encrypt data in transit for your high availability instances. Encryption in transit isn't configured automatically by RDS Custom for Oracle.
- Configure Oracle Fast-Failover Observer (FSFO) to monitor your high availability instances.
- Allow the observer to perform automatic failover when necessary conditions are met.

Customizing your RDS Custom environment

RDS Custom for Oracle includes built-in features that allow you to customize your DB instance environment without pausing automation. For example, you can use RDS APIs to customize your environment as follows:

- Create and restore DB snapshots to create a clone environment.
- Create read replicas.
- Modify storage settings.
- Change the CEV to apply release updates

For some customizations, such as changing the character set, you can't use the RDS APIs. In these cases, you need to change the environment manually by accessing your Amazon EC2 instance as the root user or logging in to your Oracle database as SYSDBA.

To customize your instance manually, you must pause and resume RDS Custom automation. This pause ensures that your customizations don't interfere with RDS Custom automation. In this way, you avoid breaking the support perimeter, which places the instance in the `unsupported-configuration` state until you fix the underlying issues. Pausing and resuming are the only supported automation tasks when you modify an RDS Custom for Oracle DB instance.

General steps for customizing your RDS Custom environment

To customize your RDS Custom DB instance, complete the following steps:

1. Pause RDS Custom automation for a specified period using the console or CLI.
2. Identify your underlying Amazon EC2 instance.
3. Connect to your underlying Amazon EC2 instance using SSH keys or AWS Systems Manager.
4. Verify your current configuration settings at the database or operating system layer.

   You can validate your changes by comparing the initial configuration to the changed configuration. Depending on the type of customization, use OS tools or database queries.
5. Customize your RDS Custom for Oracle DB instance as needed.
6. Reboot your instance or database, if required.

> **Note**
>
> In an on-premises Oracle CDB, you can preserve a specified open mode for PDBs using a built-in command or after a startup trigger. This mechanism brings PDBs to a specified state when the CDB restarts. When opening your CDB, RDS Custom automation discards any user-specified preserved states and attempts to open all PDBs. If RDS Custom can't open all PDBs, the following event is issued: `The following PDBs failed to open: list-of-PDBs`.

7. Verify your new configuration settings by comparing them with the previous settings.
8. Resume RDS Custom automation in either of the following ways:
   - Resume automation manually.

- Wait for the pause period to end. In this case, RDS Custom resumes monitoring and instance recovery automatically.

9. Verify the RDS Custom automation framework

   If you followed the preceding steps correctly, RDS Custom starts an automated backup. The status of the instance in the console shows **Available**.

For best practices and step-by-step instructions, see the AWS blog posts Make configuration changes to an Amazon RDS Custom for Oracle instance: Part 1 and Recreate an Amazon RDS Custom for Oracle database: Part 2.

Pausing and resuming your RDS Custom DB instance

You can pause and resume automation for your DB instance using the console or CLI.

Console

To pause or resume RDS Custom automation

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom DB instance that you want to modify.
3. Choose **Modify**. The **Modify DB instance** page appears.
4. For **RDS Custom automation mode**, choose one of the following options:

   - **Paused** pauses the monitoring and instance recovery for the RDS Custom DB instance. Enter the pause duration that you want (in minutes) for **Automation mode duration**. The minimum value is 60 minutes (default). The maximum value is 1,440 minutes.
   - **Full automation** resumes automation.

5. Choose **Continue** to check the summary of modifications.

   A message indicates that RDS Custom will apply the changes immediately.

6. If your changes are correct, choose **Modify DB instance**. Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

   On the RDS console, the details for the modification appear. If you paused automation, the **Status** of your RDS Custom DB instance indicates **Automation paused**.

7. (Optional) In the navigation pane, choose **Databases**, and then your RDS Custom DB instance.

   In the **Summary** pane, **RDS Custom automation mode** indicates the automation status. If automation is paused, the value is **Paused. Automation resumes in *num* minutes**.

AWS CLI

To pause or resume RDS Custom automation, use the `modify-db-instance` AWS CLI command. Identify the DB instance using the required parameter `--db-instance-identifier`. Control the automation mode with the following parameters:

- `--automation-mode` specifies the pause state of the DB instance. Valid values are `all-paused`, which pauses automation, and `full`, which resumes it.
- `--resume-full-automation-mode-minutes` specifies the duration of the pause. The default value is 60 minutes.

> **Note**
>
> Regardless of whether you specify `--no-apply-immediately` or `--apply-immediately`, RDS Custom applies modifications asynchronously as soon as possible.

In the command response, `ResumeFullAutomationModeTime` indicates the resume time as a UTC timestamp. When the automation mode is `all-paused`, you can use `modify-db-instance` to resume automation mode or extend the pause period. No other `modify-db-instance` options are supported.

The following example pauses automation for `my-custom-instance` for 90 minutes.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --automation-mode all-paused \
    --resume-full-automation-mode-minutes 90
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --automation-mode all-paused ^
    --resume-full-automation-mode-minutes 90
```

The following example extends the pause duration for an extra 30 minutes. The 30 minutes is added to the original time shown in `ResumeFullAutomationModeTime`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --automation-mode all-paused \
    --resume-full-automation-mode-minutes 30
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --automation-mode all-paused ^
    --resume-full-automation-mode-minutes 30
```

The following example resumes full automation for `my-custom-instance`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --automation-mode full \
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --automation-mode full
```

In the following partial sample output, the pending `AutomationMode` value is `full`.


```
{
    "DBInstance": {
        "PubliclyAccessible": true,
        "MasterUsername": "admin",
        "MonitoringInterval": 0,
        "LicenseModel": "bring-your-own-license",
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "0123456789abcdefg"
            }
        ],
        "InstanceCreateTime": "2020-11-07T19:50:06.193Z",
        "CopyTagsToSnapshot": false,
        "OptionGroupMemberships": [
            {
                "Status": "in-sync",
                "OptionGroupName": "default:custom-oracle-ee-19"
            }
        ],
        "PendingModifiedValues": {
            "AutomationMode": "full"
        },
        "Engine": "custom-oracle-ee",
        "MultiAZ": false,
        "DBSecurityGroups": [],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.custom-oracle-ee-19",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        ...
        "ReadReplicaDBInstanceIdentifiers": [],
        "AllocatedStorage": 250,
        "DBInstanceArn": "arn:aws:rds:us-west-2:012345678912:db:my-custom-instance",
        "BackupRetentionPeriod": 3,
        "DBName": "ORCL",
        "PreferredMaintenanceWindow": "fri:10:56-fri:11:26",
        "Endpoint": {
            "HostedZoneId": "ABCDEFGHIJKLMNO",
            "Port": 8200,
```

```
            "Address": "my-custom-instance.abcdefghijk.us-west-2.rds.amazonaws.com"
        },
        "DBInstanceStatus": "automation-paused",
        "IAMDatabaseAuthenticationEnabled": false,
        "AutomationMode": "all-paused",
        "EngineVersion": "19.my_cev1",
        "DeletionProtection": false,
        "AvailabilityZone": "us-west-2a",
        "DomainMemberships": [],
        "StorageType": "gp2",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVW",
        "ResumeFullAutomationModeTime": "2020-11-07T20:56:50.565Z",
        "KmsKeyId": "arn:aws:kms:us-west-2:012345678912:key/
aa111a11-111a-11a1-1a11-1111a11a1a1a",
        "StorageEncrypted": false,
        "AssociatedRoles": [],
        "DBInstanceClass": "db.m5.xlarge",
        "DbInstancePort": 0,
        "DBInstanceIdentifier": "my-custom-instance",
        "TagList": []
    }
```

Modifying your RDS Custom for Oracle DB instance

Modifying an RDS Custom for Oracle DB instance is similar to modifying an Amazon RDS DB instance. You can change settings such as the following:

- DB instance class
- Storage allocation and type
- Backup retention period
- Deletion protection
- Option group
- CEV (see Upgrading an RDS Custom for Oracle DB instance)
- Port

Topics

- Requirements and limitations when modifying your DB instance storage
- Requirements and limitations when modifying your DB instance class

- How RDS Custom creates your DB instance when you modify the instance class
- Modifying your RDS Custom for Oracle DB instance

Requirements and limitations when modifying your DB instance storage

Consider the following requirements and limitations when you modify the storage for an RDS Custom for Oracle DB instance:

- The minimum allocated storage for RDS Custom for Oracle is 40 GiB, and the maximum is 64 TiB.
- As with Amazon RDS, you can't decrease the allocated storage. This is a limitation of Amazon EBS volumes.
- Storage autoscaling isn't supported for RDS Custom DB instances.
- Any storage volumes that you attach manually to your RDS Custom DB instance are outside the support perimeter.

  For more information, see RDS Custom support perimeter.
- Magnetic (standard) Amazon EBS storage isn't supported for RDS Custom. You can choose only the io1, io2, gp2, or gp3 SSD storage types.

For more information about Amazon EBS storage, see Amazon RDS DB instance storage. For general information about storage modification, see Working with storage for Amazon RDS DB instances.

Requirements and limitations when modifying your DB instance class

Consider the following requirements and limitations when you modify the instance class for an RDS Custom for Oracle DB instance:

- Your DB instance must be in the `available` state.
- Your DB instance must have a minimum of 100 MiB of free space on the root volume, data volume, and binary volume.
- You can assign only a single elastic IP (EIP) to your RDS Custom for Oracle DB instance when using the default elastic network interface (ENI). If you attach multiple ENIs to your DB instance, the modify operation fails.
- All RDS Custom for Oracle tags must be present.
- If you use RDS Custom for Oracle replication, note the following requirements and limitations:

- For primary DB instances and read replicas, you can change the instance class for only one DB instance at a time.
- If your RDS Custom for Oracle DB instance has an on-premises primary or replica database, make sure to manually update private IP addresses on the on-premises DB instance after the modification completes. This action is necessary to preserve Oracle DataGuard functionality. RDS Custom for Oracle publishes an event when the modification succeeds.
- You can't modify your RDS Custom for Oracle DB instance class when the primary or read replica DB instances have FSFO (Fast-Start Failover) configured.

How RDS Custom creates your DB instance when you modify the instance class

When you modify your instance class, RDS Custom creates your DB instance as follows:

- Creates the Amazon EC2 instance.
- Creates the root volume from the latest DB snapshot. RDS Custom for Oracle doesn't retain information added to the root volume after the latest DB snapshot.
- Creates Amazon CloudWatch alarms.
- Creates an Amazon EC2 SSH key pair if you have deleted the original key pair. Otherwise, RDS Custom for Oracle retains the original key pair.
- Creates new resources using the tags that are attached to your DB instance when you initiate the modification. RDS Custom doesn't transfer tags to the new resources when they are attached directly to underlying resources.
- Transfers the binary and data volumes with the most recent modifications to the new DB instance.
- Transfers the elastic IP address (EIP). If the DB instance is publicly accessible, then RDS Custom temporarily attaches a public IP address to the new DB instance before transferring the EIP. If the DB instance isn't publicly accessible, RDS Custom doesn't create public IP addresses.

Modifying your RDS Custom for Oracle DB instance

You can modify the DB instance class or storage using the console, AWS CLI, or RDS API.

Console

To modify an RDS Custom for Oracle DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify.
4. Choose **Modify**.
5. (Optional) In **Instance configuration**, choose a value for **DB instance class**. For supported classes, see DB instance class support for RDS Custom for Oracle.
6. (Optional) In **Storage**, make the following changes as needed:
    a. Enter a new value for **Allocated storage**. It must be greater than the current value, and from 40 GiB–64 TiB.
    b. Change the value for **Storage type** to **General Purpose SSD (gp2)**, **General Purpose SSD (gp3)**, **Provisioned IOPS (io1)**, or **Provisioned IOPS (io2)**.
    c. If you specified a storage type other than **General Purpose SSD (gp2)**, you can change the **Provisioned IOPS** value.
7. (Optional) In **Additional configuration**, make the following changes as needed:
    - For **Option group**, choose a new option group. For more information, see Working with option groups in RDS Custom for Oracle.
8. Choose **Continue**.
9. Choose **Apply immediately** or **Apply during the next scheduled maintenance window**.
10. Choose **Modify DB instance**.

AWS CLI

To modify the storage for an RDS Custom for Oracle DB instance, use the modify-db-instance AWS CLI command. Set the following parameters as needed:

- `--db-instance-class` – A new instance class. For supported classes, see DB instance class support for RDS Custom for Oracle.
- `--allocated-storage` – Amount of storage to be allocated for the DB instance, in gibibytes. It must be greater than the current value, and from 40–65,536 GiB.

- `--storage-type` – The storage type: gp2, gp3, io1, or io2.
- `--iops` – Provisioned IOPS for the DB instance, if using the io1, io2, or gp3 storage types.
- `--apply-immediately` – Use `--apply-immediately` to apply the storage changes immediately.

  Or use `--no-apply-immediately` (the default) to apply the changes during the next maintenance window.

The following example changes the DB instance class of `my-cfo-instance` to `db.m5.16xlarge`. The command also changes the storage size to `1024` (1 TiB), storage type to `io2`, Provisioned IOPS to `3000`, and option group to `cfo-ee-19-mt`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-cfo-instance \
    --db-instance-class db.m5.16xlarge \
    --storage-type io2 \
    --iops 3000 \
    --allocated-storage 1024 \
    --option-group cfo-ee-19-mt \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-cfo-instance ^
    --db-instance-class db.m5.16xlarge ^
    --storage-type io2 ^
    --iops 3000 ^
    --allocated-storage 1024 ^
    --option-group cfo-ee-19-mt ^
    --apply-immediately
```

Changing the character set of an RDS Custom for Oracle DB instance

RDS Custom for Oracle defaults to the character set US7ASCII. You might want to specify different character sets to meet language or multibyte character requirements. When you use RDS Custom

for Oracle, you can pause automation and then change the character set of your database manually.

Changing the character set of an RDS Custom for Oracle DB instance has the following requirements:

- You can only change the character on a newly provisioned RDS Custom instance that has an empty or starter database with no application data. For all other scenarios, change the character set using DMU (Database Migration Assistant for Unicode).
- You can only change to a character set supported by RDS for Oracle. For more information, see Supported DB character sets.

To change the character set of an RDS Custom for Oracle DB instance

1. Pause RDS Custom automation. For more information, see Pausing and resuming your RDS Custom DB instance.
2. Log in to your database as a user with SYSDBA privileges.
3. Restart the database in restricted mode, change the character set, and then restart the database in normal mode.

   Run the following script in your SQL client:

   ```
   SHUTDOWN IMMEDIATE;
   STARTUP RESTRICT;
   ALTER DATABASE CHARACTER SET INTERNAL_CONVERT AL32UTF8;
   SHUTDOWN IMMEDIATE;
   STARTUP;
   SELECT VALUE FROM NLS_DATABASE_PARAMETERS WHERE PARAMETER = 'NLS_CHARACTERSET';
   ```

   Verify that the output shows the correct character set:

   ```
   VALUE
   --------
   AL32UTF8
   ```

4. Resume RDS Custom automation. For more information, see Pausing and resuming your RDS Custom DB instance.

Setting the NLS_LANG value in RDS Custom for Oracle

A *locale* is a set of information addressing linguistic and cultural requirements that corresponds to a given language and country. To specify locale behavior for Oracle software, set the NLS_LANG environment variable on your client host. This variable sets the language, territory, and character set used by the client application in a database session.

For RDS Custom for Oracle, you can set only the language in the NLS_LANG variable: the territory and character use defaults. The language is used for Oracle database messages, collation, day names, and month names. Each supported language has a unique name, for example, American, French, or German. If language is not specified, the value defaults to American.

After you create your RDS Custom for Oracle database, you can set NLS_LANG on your client host to a language other than English. To see a list of languages supported by Oracle Database, log in to your RDS Custom for Oracle database and run the following query:

```
SELECT VALUE FROM V$NLS_VALID_VALUES WHERE PARAMETER='LANGUAGE' ORDER BY VALUE;
```

You can set NLS_LANG on the host command line. The following example sets the language to German for your client application using the Z shell on Linux.

```
export NLS_LANG=German
```

Your application reads the NLS_LANG value when it starts and then communicates it to the database when it connects.

For more information, see Choosing a Locale with the NLS_LANG Environment Variable in the *Oracle Database Globalization Support Guide*.

Support for Transparent Data Encryption

RDS Custom supports Transparent Data Encryption (TDE) for RDS Custom for Oracle DB instances.

However, you can't enable TDE using an option in a custom option group as you can in RDS for Oracle. You turn on TDE manually. For information about using Oracle Transparent Data Encryption, see Securing stored data using Transparent Data Encryption in the Oracle documentation.

Tagging RDS Custom for Oracle resources

You can tag RDS Custom resources as with Amazon RDS resources, but with some important differences:

- Don't create or modify the `AWSRDSCustom` tag that's required for RDS Custom automation. If you do, you might break the automation.
- The `Name` tag is added to RDS Custom resources with prefix value of `do-not-delete-rds-custom` or `rds-custom!oracle-do-not-delete`. Any customer-passed value for the key is overwritten.
- Tags added to RDS Custom DB instances during creation are propagated to all other related RDS Custom resources.
- Tags aren't propagated when you add them to RDS Custom resources after DB instance creation.

For general information about resource tagging, see Tagging Amazon RDS resources.

Deleting an RDS Custom for Oracle DB instance

Warning

Deleting a DB instance is a permanent action. You can't recover a DB instance after deletion unless you have a backup or snapshot.
When you delete an RDS Custom DB instance, AWS automatically deletes the underlying Amazon EC2 instance and EBS volumes. Don't manually terminate the Amazon EC2 instance or delete the EBS volumes before deleting the DB instance through Amazon RDS. Manual deletion of these resources causes DB instance deletion and final snapshot creation to fail, preventing any possibility of recovery.

To delete an RDS Custom DB instance, do the following:

- Provide the name of the DB instance.
- Clear the option to take a final DB snapshot of the DB instance.
- Choose or clear the option to retain automated backups.

You can delete an RDS Custom DB instance using the console or the CLI. The time required to delete the DB instance can vary depending on the backup retention period (that is, how many backups to delete) and how much data is deleted.

Console

To delete an RDS Custom DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom DB instance that you want to delete. RDS Custom DB instances show the role **Instance (RDS Custom)**.
3. For **Actions**, choose **Delete**.
4. To retain automated backups, choose **Retain automated backups**.
5. Enter **`delete me`** in the box.
6. Choose **Delete**.

AWS CLI

You delete an RDS Custom DB instance by using the delete-db-instance AWS CLI command. Identify the DB instance using the required parameter `--db-instance-identifier`. The remaining parameters are the same as for an Amazon RDS DB instance, with the following exceptions:

- `--skip-final-snapshot` is required.
- `--no-skip-final-snapshot` isn't supported.
- `--final-db-snapshot-identifier` isn't supported.

The following example deletes the RDS Custom DB instance named `my-custom-instance`, and retains automated backups.

Example

For Linux, macOS, or Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier my-custom-instance \
```

```
    --skip-final-snapshot \
    --no-delete-automated-backups
```

For Windows:

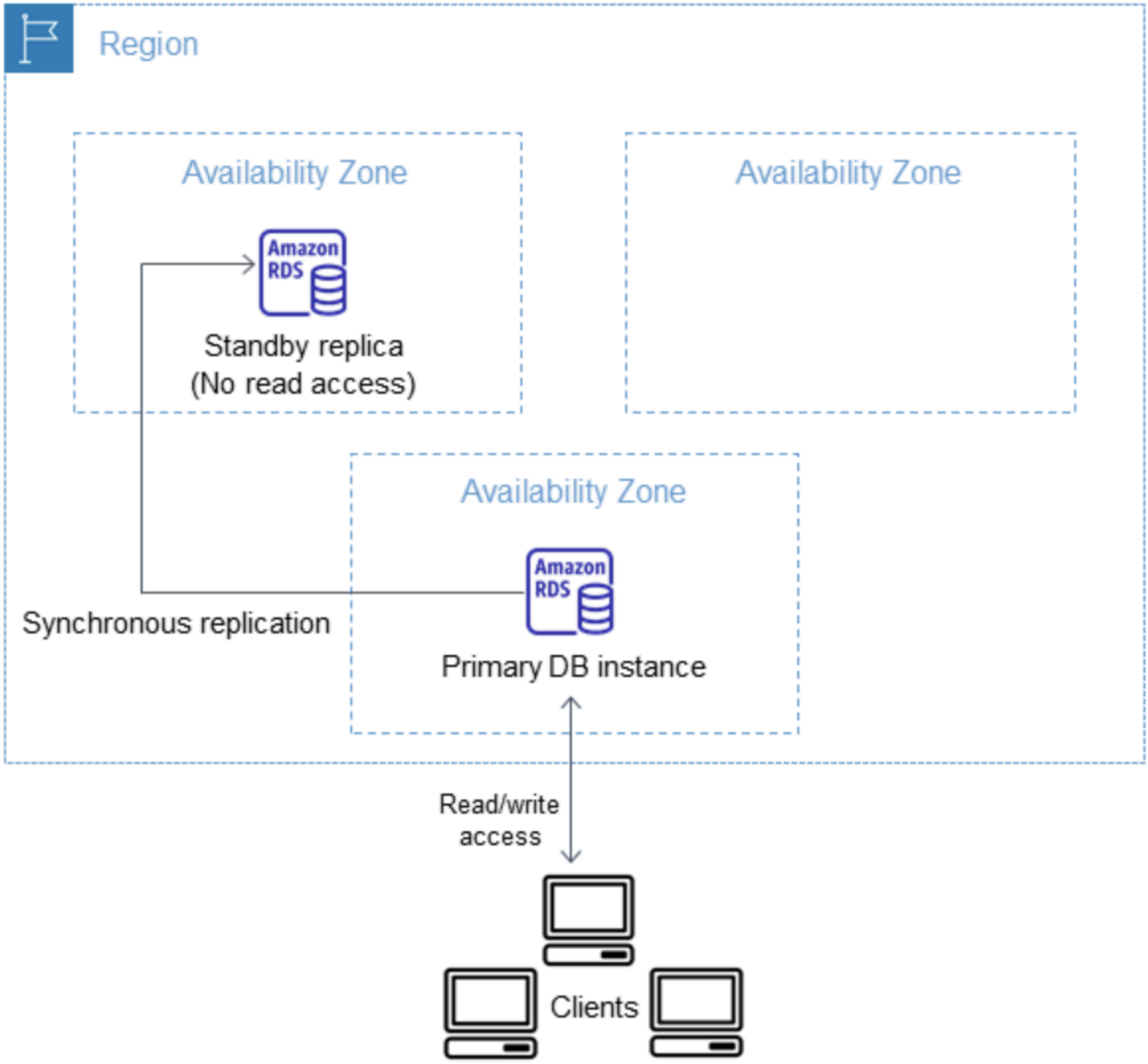
```
aws rds delete-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --skip-final-snapshot ^
    --no-delete-automated-backups
```

Managing a Multi-AZ deployment for RDS Custom for Oracle

In a Multi-AZ DB instance deployment for RDS Custom for Oracle, Amazon RDS automatically provisions and maintains a synchronous standby replica in a different Availability Zone (AZ). The primary DB instance is synchronously replicated across Availability Zones to a standby replica to provide data redundancy.

A Multi-AZ DB instance deployment enhances availability during planned system maintenance. If planned database maintenance or unplanned service disruption occurs, Amazon RDS automatically fails over to the up-to-date standby DB instance. Database operations can resume quickly without manual intervention. The primary and standby instances use the same endpoint, whose physical network address transitions to the standby replica as part of the failover process. You don't need to reconfigure your application when a failover occurs.

The following diagram shows the Multi-AZ architecture for RDS Custom for Oracle:



The Amazon RDS console shows the Availability Zone of the standby replica (the secondary AZ). You can also use the `describe-db-instances` CLI command or the `DescribeDBInstances` API operation to find the secondary AZ.

Region and version availability for a Multi-AZ deployment for RDS Custom for Oracle

Multi-AZ deployments for RDS Custom for Oracle are supported for the following releases for both Enterprise Edition (EE) and Standard Edition 2 (SE2):

- Oracle Database 19c
- Oracle Database 12c Release 2 (12.2)
- Oracle Database 12c Release 1 (12.1)

> **Note**
>
> Multi-AZ deployments for RDS Custom for Oracle aren't supported on Oracle Database 18c.

Multi-AZ deployments for RDS Custom for Oracle are available in all Regions where RDS Custom for Oracle is available. For more information on Region availability of Multi-AZ deployments for RDS Custom for Oracle, see the section called “Supported Regions and DB engines for RDS Custom for Oracle”.

Limitations for a Multi-AZ deployment in RDS Custom for Oracle

Multi-AZ deployments with RDS Custom for Oracle have the following limitations:

- If your DB instance was created before June 30, 2025, you can't convert from a Single-AZ to Multi-AZ deployment. Your underlying Custom Engine Version (CEV) was built with an older service-provided AMI that lacks Multi-AZ support. You must migrate your database to a new DB instance using a CEV that you create with a service-provided AMI after June 30, 2025. For details, see the section called “Migration for CEVs before 06.30.2025”.
- You can't create cross-Region Multi-AZ deployments.
- You can't configure the standby DB instance to accept database read activity.
- When you use a Custom Engine Version (CEV) with a Multi-AZ deployment, your standby DB instance uses the same CEV. The standby DB instance can't use a different CEV.
- You can't create a read replica on a Multi-AZ deployment instance and you also cannot modify the primary instance of the read replica to have a Multi-AZ deployment.

- RDS Custom for Oracle DB instances with Multi-AZ deployment can have increased write and commit latency compared to a Single-AZ deployment. This increase can happen because of the synchronous data replication between DB instances. You might have a change in latency if your deployment fails over to the standby replica, although AWS is engineered with low-latency network connectivity between Availability Zones.

Creating a Multi-AZ deployment in RDS Custom for Oracle

To create an RDS Custom for Oracle DB instance with a Multi-AZ deployment, follow the steps in the section called "Setting up your RDS Custom for Oracle environment" to set up your environment with the prerequisites.

> **Important**
>
> To simplify setup, we recommend that you use the latest CloudFormation template file provided in the network setup instructions. For more information, see the section called "Deploy with CloudFormation".

You can create an RDS Custom for Oracle instance with a Multi-AZ deployment by choosing the **Multi-AZ** option during the database instance creation in the Amazon RDS console. Alternatively, you can specify the `--multi-az` parameter in the Amazon RDS `create-db-instance` command in the AWS CLI.

Prerequisites for a Multi-AZ deployment in RDS Custom for Oracle

A Multi-AZ deployment for RDS Custom for Oracle is different from Multi-AZ for RDS for Oracle. Unlike Multi-AZ for RDS for Oracle, you must meet the prerequisites for RDS Custom for Oracle before you create your Multi-AZ DB instance. This is because RDS Custom runs inside your own account, which requires permissions. Also, you need to create the Multi-AZ instance on the CEV with the latest service-provided AMI, which supports the Multi-AZ deployments.

If you don't complete the prerequisites, your Multi-AZ DB instance might fail to run, or automatically revert to a Single-AZ DB instance. For more information about prerequisites, see the section called "Prerequisites".

RDS Custom for Oracle requires specific prerequisites when converting from Single-AZ to Multi-AZ deployment. Incomplete prerequisites cause Multi-AZ setup to fail. Use either manual setup

or the latest CloudFormation template file provided in the network setup instructions. For more information, see Step 3: Extract the CloudFormation templates for RDS Custom for Oracle.

To complete the prerequisites manually, follow the steps in the section called "Converting Single-AZ to Multi-AZ" and note the following:

- Make sure your RDS Custom for Oracle DB instance uses a CEV created after June 30, 2025 with the latest service-provided AMI.
- Update the Amazon RDS security group inbound and outbound rules to allow port 1120.
- Create new Amazon SQS VPC endpoints that allow the Amazon RDS RDS Custom for Oracle DB instance to communicate with Amazon SQS.
- Update the Amazon SQS permissions in the instance profile role.

> **Important**
>
> Don't reboot your Multi-AZ primary DB instance manually by logging into the instance without pausing RDS Custom for Oracle automation.

Migration steps for DB instances using CEVs created before June 30, 2025

If you created your RDS Custom for Oracle DB instance before June 30, 2025, you can't directly modify it to Multi-AZ deployment because the underlying CEV was built with an older service-provided AMI that lacks Multi-AZ support. You must migrate your database to a new instance using a CEV you create after June 30, 2025. Do the following for the migration.

1. **Create CEV from source** with the latest service-provided AMI (after June 30, 2025).
2. **Launch a new DB instance** with the new CEV.
3. **Migrate your database** from the existing DB instance that doesn't support Multi-AZ deployment to the newly created instance using one of following methods:
   - Physical migration of Oracle databases to Amazon RDS Custom using Data Guard
   - Physical migration of Oracle databases to Amazon RDS Custom using RMAN duplication

Converting a Single-AZ deployment to a Multi-AZ deployment in RDS Custom for Oracle

You can convert an existing multi-AZ compatible RDS Custom for Oracle instance from a Single-AZ deployment to a Multi-AZ deployment. When you modify the DB instance, Amazon RDS performs several actions:

- Takes a snapshot of the primary DB instance.
- Creates new volumes for the standby replica from the snapshot. These volumes initialize in the background, and maximum volume performance is achieved after the data is fully initialized.
- Turns on synchronous block-level replication between the primary and standby DB instances.

> **Important**
>
> We recommend that you avoid modifying your RDS Custom for Oracle DB instance from a Single-AZ to a Multi-AZ deployment on a production DB instance during periods of peak activity.

AWS uses a snapshot to create the standby instance to avoid downtime when you convert from Single-AZ to Multi-AZ, but performance might be impacted during and after converting to Multi-AZ. This impact can be significant for workloads that are sensitive to write latency. While this capability allows large volumes to quickly be restored from snapshots, it can cause increase in the latency of I/O operations because of the synchronous replication. This latency can impact your database performance.

Configuring prerequisites to modify a Single-AZ to a Multi-AZ deployment using CloudFormation

Follow Step 3: Extract the CloudFormation templates for RDS Custom for Oracle to setup your VPC and IAM profile again to add SQS VPC endpoint and SQS permission in IAM profile.

Configuring prerequisites to modify a Single-AZ to a Multi-AZ deployment manually

If you choose to configure the prerequisites manually, perform the following tasks.

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/
2. Choose **Endpoint**. The **Create Endpoint** page appears.

3. For **Service Category**, choose **AWS services**.
4. In **Services**, search for SQS.
5. In **VPC**, choose the VPC where your RDS Custom for Oracle DB instance is deployed.
6. In **Subnets**, choose the subnets where your RDS Custom for Oracle DB instance is deployed.
7. In **Security Groups**, choose the security group where your RDS Custom for Oracle DB instance is deployed.
8. For **Policy**, choose **Custom**.
9. In your custom policy, replace the `AWS partition`, `Region`, `accountId`, and `IAM-Instance-role` with your own values.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/AWSRDSCustom": "custom-oracle"
                }
            },
            "Action": [
                "SQS:SendMessage",
                "SQS:ReceiveMessage",
                "SQS:DeleteMessage",
                "SQS:GetQueueUrl"
            ],
            "Resource": "arn:aws:sqs:us-east-1:111122223333:do-not-delete-rds-
custom-*",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/{IAM-Instance-role}"
            }
        }
    ]
}
```

Update the **Instance profile** with permission to access Amazon SQS. Replace the `AWS partition`, `Region`, and `accountId` with your own values.

```
{
    "Sid": "13",
    "Effect": "Allow",
    "Action": [
        "SQS:SendMessage",
        "SQS:ReceiveMessage",
        "SQS:DeleteMessage",
        "SQS:GetQueueUrl"
    ],
    "Resource": [
        {
            "Fn::Sub": "arn:${AWS::Partition}:sqs:${AWS::Region}:${AWS::AccountId}:do-
not-delete-rds-custom-*"
        }
    ],
    "Condition": {
        "StringLike": {
            "aws:ResourceTag/AWSRDSCustom": "custom-oracle"
        }
    }
}
```

Update the Amazon RDS security group inbound and outbound rules to allow port 1120.

- In **Security Groups**, choose the group where your RDS Custom for Oracle DB instance is deployed.
- For **Inbound Rules**, create a **Custom TCP** rule to allow port `1120` from the source group.
- For **Outbound Rules**, create a **Custom TCP** rule to allow port `1120` to the destination group.

Modify using the RDS console, AWS CLI, or RDS API

After you've completed the prerequisites, you can modify an RDS Custom for Oracle DB instance from a Single-AZ to Multi-AZ deployment using the Amazon RDS console, AWS CLI, or Amazon RDS API.

Console

To modify an existing RDS Custom for Oracle Single-AZ to Multi-AZ deployment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the Amazon RDS console, choose **Databases**. The **Databases** pane appears.
3. Choose the RDS Custom for Oracle DB instance that you want to modify.
4. For **Actions**, choose **Convert to Multi-AZ deployment**.
5. On the **Confirmation** page, choose **Apply immediately** to apply the changes immediately. Choosing this option doesn't cause downtime, but there is a possible performance impact. Alternatively, you can choose to apply the update during the next maintenance window. For more information, see the section called "Scheduling modifications".
6. On the **Confirmation** page, choose **Convert to Multi-AZ**.

AWS CLI

To convert to a Multi-AZ DB instance deployment by using the AWS CLI, call the modify-db-instance command and set the `--multi-az` option. Specify the DB instance identifier and the values for other options that you want to modify. For information about each option, see the section called "Available settings".

Example

The following code modifies `mycustomdbinstance` by including the `--multi-az` option. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately. For more information, see the section called "Scheduling modifications".

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mycustomdbinstance \
    --multi-az \
    [--no-apply-immediately | --apply-immediately]
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mycustomdbinstance ^
    --multi-az ^
    [--no-apply-immediately | --apply-immediately]
```

RDS API

To convert to a Multi-AZ DB instance deployment with the Amazon RDS API, call the ModifyDBInstance operation and set the `MultiAZ` parameter to true.

Converting a Multi-AZ deployment to a Single-AZ deployment in RDS Custom for Oracle

You can modify an existing RDS Custom for Oracle DB instance from a Multi-AZ to a Single-AZ deployment.

Console

To modify an RDS Custom for Oracle DB instance from a Multi-AZ to Single-AZ deployment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the Amazon RDS console, choose **Databases**. The **Databases** pane appears.
3. Choose the RDS Custom for Oracle DB instance that you want to modify.
4. For **Multi-AZ deployment**, choose **No**.
5. On the **Confirmation** page, choose **Apply immediately** to apply the changes immediately. Choosing this option doesn't cause downtime, but there is a possible performance impact. Alternatively, you can choose to apply the update during the next maintenance window. For more information, see the section called “Scheduling modifications”.
6. On the **Confirmation** page, choose **Modify DB Instance**.

AWS CLI

To modify a Multi-AZ deployment to a Single-AZ deployment by using the AWS CLI, call the modify-db-instance command and include the `--no-multi-az` option. Specify the DB instance identifier and the values for other options that you want to modify. For information about each option, see the section called “Available settings”.

Example

The following code modifies `mycustomdbinstance` by including the `--no-multi-az` option. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately. For more information, see the section called "Scheduling modifications".

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mycustomdbinstance \
    --no-multi-az \
    [--no-apply-immediately | --apply-immediately]
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mycustomdbinstance ^
    --no-multi-az ^
    [--no-apply-immediately | --apply-immediately]
```

RDS API

To modify a Multi-AZ deployment to a Single-AZ deployment by using the Amazon RDS API, call the ModifyDBInstance operation and set the `MultiAZ` parameter to `false`.

Customizing the OS in an RDS Custom for Oracle Multi-AZ deployment

With RDS Custom for Oracle Multi-AZ deployments, you can customize the operating system and install third-party software on both primary and standby EC2 instances. Unlike Amazon RDS, RDS Custom for Oracle provides administrative access to the database environment and underlying operating system, allowing you to install monitoring tools, security agents, or custom applications in addition to Oracle databases.

When you customize the OS on a Multi-AZ deployment, you're responsible for ensuring that customizations exist on both primary and standby instances. This approach ensures application continuity during Multi-AZ failover and maintains consistent functionality across both instances.

Requirements for customizing the OS in an RDS Custom for Oracle Multi-AZ deployment

Before you customize the OS in a Multi-AZ deployment, be aware of the following requirements:

- Install third-party software only on the `/rdsdbdata` mount point. The data volume (`/rdsdbdata`) is the only data that is replicated in a Multi-AZ deployment. The root volume (`/`) is replaced during OS patching, and the binary volume (`/rdsdbbin`) is replaced during database patching. Software installed on the root and binary volumes is lost during patching.
- Ensure all customizations comply with AWS and Oracle licensing terms and conditions.
- Before you convert from Single-AZ to Multi-AZ, make sure the HugePages settings in `/etc/sysctl.conf` work correctly.

Identifying EC2 instances in an RDS Custom for Oracle Multi-AZ deployment

When customizing your Multi-AZ instances, identify which Amazon EC2 instances serve as the primary and standby for your RDS Custom for Oracle deployment.

To identify primary and standby EC2 instances

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose your Multi-AZ RDS Custom for Oracle DB instance.
4. In the **Configuration** section, note the **Resource ID** (format: db-*nnnnnnn*).
5. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
6. In the navigation pane, choose **Instances**.
7. In the search box, enter the Resource ID from step 4.
8. The search results show two instances: your primary and secondary. The instance with the active RDS Custom for Oracle database is the primary.

Customizing the OS before creating an RDS Custom for Oracle Multi-AZ deployment

In this scenario, your current deployment is a Single-AZ DB instance. You can customize the OS and then convert your DB instance to a Multi-AZ deployment. If you're installing third-party software, and you have modified files on multiple volumes, this technique is recommended.

To customize the OS before converting your Single-AZ to Multi-AZ

1. Connect to the EC2 instance in your Single-AZ deployment using AWS Systems Manager Session Manager or SSH.
2. Perform either of the following customizations:

- Install third-party software on the data volume (`/rdsdbdata`).
- Customize files on the root volume (`/`).

3. Test your software or root volume customizations to ensure they work correctly.
4. Convert the Single-AZ DB instance to a Multi-AZ deployment by following the instructions in Converting a Single-AZ deployment to a Multi-AZ deployment in RDS Custom for Oracle.
5. Verify that your customizations exist on both instances in the Multi-AZ deployment. For more information, see Identifying EC2 instances in an RDS Custom for Oracle Multi-AZ deployment.

Customizing the OS after creating an RDS Custom for Oracle Multi-AZ deployment

If you have an exiting Multi-AZ deployment, you can deploy your customizations using AWS Systems Manager or using manual techniques.

Customizing the OS in a Multi-AZ deployment using AWS Systems Manager

For existing Multi-AZ DB instances, we recommend Systems Manager as the most reliable way to apply customizations simultaneously to both primary and standby instances. This approach ensures consistency. For a general introduction to this service, see What is AWS Systems Manager?. To learn how to install software on both DB instances simultaneously, see Install or update Distributor packages.

Customizing the OS in a Multi-AZ deployment manually

In this scenario, your Multi-AZ deployment already exists, but you don't use AWS Systems Manager to deploy the customizations. You can customize your OS manually in either of the following ways:

Customize the OS on the primary instance and replicate the changes

Multi-AZ deployment automatically replicates the `rdsdbdata` volume. You can customize the OS in the following ways:

- Install third-party software directly on the `/rdsdbdata` mount point.
- To modify files on the root volume (`/`), create files on the data volume and then create a symbolic link from the root volume files to the data volume files.

Customize the OS on the primary and standby instances separately

In this approach, you customize the OS on the primary instance. Then you perform the same customizations on the standby instance.

To customize the OS on the primary instance so that they are replicated automatically

1. Identify the primary and standby DB instances using the procedure in Identifying EC2 instances in an RDS Custom for Oracle Multi-AZ deployment.
2. Connect to the primary EC2 instance using Session Manager or SSH.
3. Use either of the following techniques, depending on your business requirements:

   Install third-party software

   Install your software on the `/rdsdbdata` mount point.

   ```
   sudo mkdir -p /rdsdbdata/custom-software
   cd /rdsdbdata/custom-software
   # Install your software here
   ```

   Customize the root volume

   Create symbolic links from OS configuration files on the root volume to files on the data volume. For example, create file `/rdsdbdata/customizations/sysctl.conf`, and then create a symbolic link at `/etc/sysctl.conf` that points to `/rdsdbdata/customizations/sysctl.conf`.

   ```
   sudo mkdir -p /rdsdbdata/customizations
   sudo mv /etc/sysctl.conf /rdsdbdata/customizations/sysctl.conf
   sudo ln -sf /rdsdbdata/customizations/sysctl.conf /etc/sysctl.conf
   ```

4. Test your software or root volume customizations to ensure they work correctly.
5. Connect to the standby instance and verify that the synchronous replication has copied your software or root volume customizations to the `/rdsdbdata` directory.

Customizing the binary volume in an RDS Custom for Oracle Multi-AZ deployment

You can apply a database patch to the binary volume (`/rdsdbbin`) in an RDS Custom for Oracle Multi-AZ deployment. You must apply the patch to the primary and standby instances. Consider the following guidelines:

- When you perform a one-off patch, we recommend that you create a new CEV with the new one-off patch included in the manifest.

- To apply a one-off patch manually, make sure to unzip the one-off patch in both the primary and secondary EC2 instances. Applying the patch and running `datapatch` is required only on the primary Multi-AZ instance.
- If you patch the database using a different CEV, the binary volume is replaced. Make sure to include the one-off patch in the manifest of the new CEV.

Best practices for customizing the OS

Follow these best practices when customizing the OS on RDS Custom for Oracle Multi-AZ instances:

Test in non-production environments

Always test customizations in a non-production environment before applying them to production instances.

Document all changes

Maintain detailed documentation of all customizations for future reference and troubleshooting. We recommend that you store your customizations in a script that you can be applied anytime, just in case.

Verify on both instances

Regularly verify that customizations are present and functioning correctly on both primary and standby instances.

Use Systems Manager for consistency

Use Systems Manager for consistent application of changes across instances, especially for existing Multi-AZ deployments.

Deploying RDS Custom for Oracle with AWS CloudFormation

Automate your RDS Custom for Oracle deployment using the provided AWS CloudFormation template. Complete the following prerequisites before deploying the resources.

Prerequisites

Download required Oracle files

You need specific Oracle installation files before you can create the CloudFormationtemplate. Download these files before you deploy.

1. Navigate to Oracle Database 19c (19.3)
2. Locate and download the file `LINUX.X64_193000_db_home.zip`
3. Rename the file to`V982063-01.zip`
4. Download the remaining patches, selecting **Platform or Language** as `Linux x86-64`

Latest OPatch utility

Patch 6880880

January 2023 PSU Patches

Database PSU & RU Patches

- Patch 34765931
- Patch 34786990

Additional Required Patches

- Patch 35099667
- Patch 35099674
- Patch 28730253
- Patch 29213893
- Patch 35012866

April 2023 PSU Patches

Database PSU & RU Patches

- Patch 35042068
- Patch 35050341

Additional Required Patches

- Patch 28730253
- Patch 29213893
- Patch 33125873
- Patch 35220732
- Patch 35239280

Amazon S3 bucket setup

1. Create an Amazon S3 bucket in your AWS account, or choose an existing bucket.
2. Create a folder structure in the bucket similar to example below.

```
<bucket-name>/
### oracle_cev/
    ### V982063-01.zip
    ### p6880880_190000_Linux-x86-64.zip
    ### p34765931_190000_Linux-x86-64.zip
    ### p34786990_190000_Linux-x86-64.zip
    ### p35099667_190000_Linux-x86-64.zip
    ### p35099674_190000_Generic.zip
    ### p28730253_190000_Linux-x86-64.zip
    ### p29213893_1918000DBRU_Generic.zip
    ### p35012866_1918000DBRU_Linux-x86-64.zip
    ### p35042068_190000_Linux-x86-64.zip
    ### p35050341_190000_Linux-x86-64.zip
    ### p29213893_1919000DBRU_Generic.zip
    ### p33125873_1919000DBRU_Linux-x86-64.zip
    ### p35220732_190000_Linux-x86-64.zip
    ### p35239280_190000_Generic.zip
```

3. Upload all of Oracle files that you previously downloaded to the appropriate folders.

Deploy RDS Custom for Oracle using AWS CloudFormation

Step 1: Prepare the CloudFormation template

Before you can deploy RDS Custom for Oracle, you need to download and configure the CloudFormation template that creates the necessary prerequisites.

Copy and save the template

1. Go to Deploying RDS Custom for Oracle with single and multiple Availability Zones
2. Copy the template in your preferred format (YAML or JSON)
3. Save the file in YAML or JSON format. For example, `rds-custom-oracle-prereqs.yaml`

Launch the stack in the AWS console

1. Open the AWS Console and navigate to AWS CloudFormation
2. Choose **Create stack** > **With new resources (standard)**
3. Select **Choose an existing template**
4. Select **Upload a template file** > **Choose file**
5. Select the template file you previously downloaded
6. Keep the default parameter values
7. Select **Next** to create the stack

Alternative: Using AWS CLI

Instead of using the console, you can create the stack using the AWS CLI:

```
aws cloudformation create-stack \
  --stack-name rds-custom-oracle \
  --template-body file://rds-custom-oracle-prereqs.yaml \
  --capabilities CAPABILITY_NAMED_IAM
```

Step 2: Create the Custom Engine Versions (CEVs) and Amazon RDS instances

Copy and save the template

1. Go to Deploying RDS Custom for Oracle with single and multiple Availability Zones
2. Copy the template in your preferred format (YAML or JSON)
3. Update the following parameters in the template if needed:
   - `BucketName`
   - `CEVS3Prefix`
   - Database master password (replace **********)

4. Save the file in YAML or JSON format

Step 3: Deploy using the AWS console

1. Open the AWS Console and navigate to AWS CloudFormation
2. Choose **Create stack** > **With new resources (standard)**
3. Select **Choose an existing template**
4. Select **Upload a template file** > **Choose file**
5. Select the template file you previously downloaded
6. Leave the parameters as default values
7. Fill in the parameters as follows:

```
BucketName: rds-custom-id
CEVS3Prefix: oracle_cev
CEVCreation: Yes
```

8. Review the configuration and select **Next** to create the stack

Optional: Deploy Using AWS CLI

```
aws cloudformation create-stack \
  --stack-name rds-custom-oracle \
  --template-body file://rds-custom-oracle.yaml \
  --parameters \
    ParameterKey=BucketName,ParameterValue=rds-custom-id \
    ParameterKey=CEVS3Prefix,ParameterValue=oracle_cev \
    ParameterKey=CEVCreation,ParameterValue=Yes \
  --capabilities CAPABILITY_NAMED_IAM
```

Deployment resources created

The template creates the following resources:

- Amazon VPC with public and private subnets
- Security groups
- Amazon VPC endpoints
- IAM roles and policies

- AWS KMS key for encryption
- Custom Engine Versions (CEVs)
- RDS Custom for Oracle instances for both single-AZ and multi-AZ configurations

Monitor your deployment progress

After you create the CloudFormation stack, monitor its progress to ensure successful deployment. The deployment process includes creating Custom Engine Versions (CEVs) and RDS instances.

To monitor deployment progress:

1. Open the CloudFormation console.
2. Choose your stack name.
3. Choose the **Events** tab to view progress and identify any errors.

> **Note**
>
> CEV creation typically requires 2-3 hours. After CEV creation completes successfully, Amazon RDS automatically begins creating the Amazon RDS instance.

Post-Deployment

After the stack creation process completes, perform the following post-deployment verification and configuration steps:

1. From the Amazon RDS console page, navigate to **Custom engine versions** to verify CEV creation.
2. Confirm Amazon RDS instances are created and available
3. Test connectivity to the Amazon RDS instances
4. Set up monitoring and backup strategies as needed

Cleanup

To remove all resources, run the following AWS CLI command:

```
aws cloudformation delete-stack --stack-name rds-custom-oracle
```

Troubleshooting

If you encounter issues during deployment, use the following solutions to resolve common problems.

CEV creation fails

- Verify all required patches are uploaded to Amazon S3
- Check IAM permissions
- Verify the patch versions are correct; see the the section called “Prerequisites” for the list of required patches.

Amazon RDS instance creation fails

- Check VPC/subnet configurations
- Verify security group rules
- Confirm CEV is available

Working with Oracle replicas for RDS Custom for Oracle

You can create Oracle replicas for RDS Custom for Oracle DB instances that run Oracle Enterprise Edition. Both container databases (CDBs) and non-CDBs are supported. Standard Edition 2 doesn't support Oracle Data Guard.

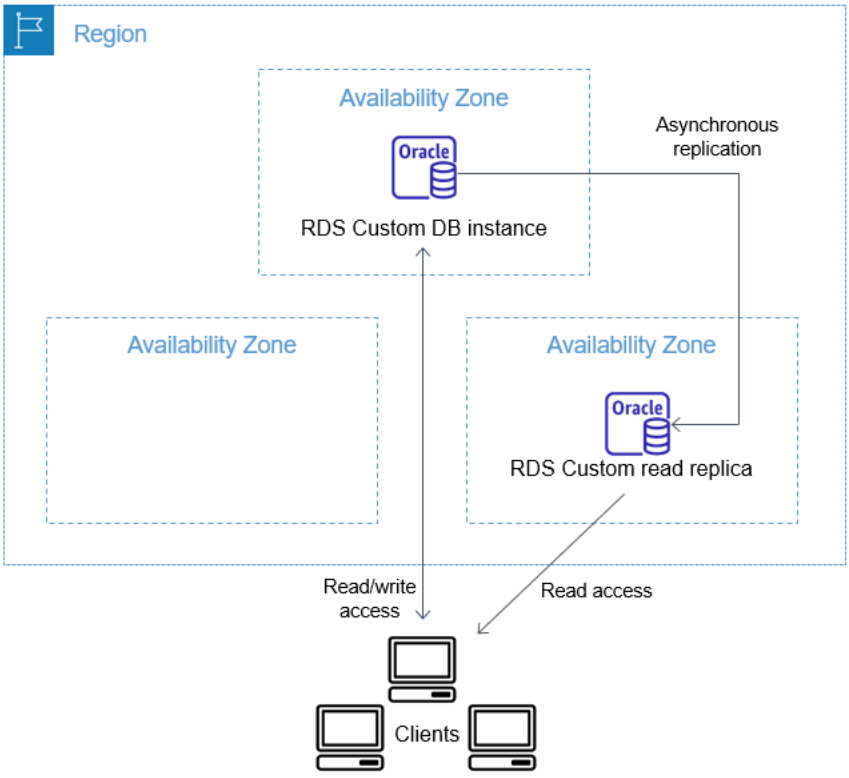
Creating an RDS Custom for Oracle replica is similar to creating an RDS for Oracle replica, but with important differences. For general information about creating and managing Oracle replicas, see Working with DB instance read replicas and Working with read replicas for Amazon RDS for Oracle.

Topics

- Overview of RDS Custom for Oracle replication
- Guidelines and limitations for RDS Custom for Oracle replication
- Promoting an RDS Custom for Oracle replica to a standalone DB instance
- Configuring a VPN tunnel between RDS Custom for Oracle primary and replica instances

Overview of RDS Custom for Oracle replication

The architecture of RDS Custom for Oracle replication is analogous to RDS for Oracle replication. A primary DB instance replicates asynchronously to one or more Oracle replicas.



Maximum number of replicas

As with RDS for Oracle, you can create up to five managed Oracle replicas of your RDS Custom for Oracle primary DB instance. You can also create your own manually configured (external) Oracle replicas. External replicas don't count toward your DB instance limit. They also lie outside the RDS Custom support perimeter. For more information about the support perimeter, see RDS Custom support perimeter.

Replica naming convention

Oracle replica names are based on the database unique name. The format is *DB_UNIQUE_NAME_X*, with letters appended sequentially. For example, if your database unique name is ORCL, the first two replicas are named ORCL_A and ORCL_B. The first six letters, A–F, are reserved for RDS Custom. RDS Custom copies database parameters from your primary DB instance to the replicas. For more information, see DB_UNIQUE_NAME in the Oracle documentation.

Replica backup retention

By default, RDS Custom Oracle replicas use the same backup retention period as your primary DB instance. You can modify the backup retention period to 1–35 days. RDS Custom supports backing up, restoring, and point-in-time recovery (PITR). For more information about backing up and restoring RDS Custom DB instances, see Backing up and restoring an Amazon RDS Custom for Oracle DB instance.

> **Note**
>
> While creating a Oracle replica, RDS Custom temporarily pauses the cleanup of redo log files. In this way, RDS Custom ensures that it can apply these logs to the new Oracle replica after it becomes available.

Replica promotion

You can promote managed Oracle replicas in RDS Custom for Oracle using the console, `promote-read-replica` AWS CLI command, or `PromoteReadReplica` API. If you delete your primary DB instance, and all replicas are healthy, RDS Custom for Oracle promotes your managed replicas to standalone instances automatically. If a replica has paused automation or is outside the support perimeter, you must fix the replica before RDS Custom can promote it automatically. You can only promote external Oracle replicas manually.

Guidelines and limitations for RDS Custom for Oracle replication

When you create RDS Custom for Oracle replicas, not all RDS Oracle replica options are supported.

Topics

- General guidelines for RDS Custom for Oracle replication
- General limitations for RDS Custom for Oracle replication
- Networking requirements and limitations for RDS Custom for Oracle replication
- External replica limitations for RDS Custom for Oracle

General guidelines for RDS Custom for Oracle replication

When working with RDS Custom for Oracle, follow these guidelines:

- You can use RDS Custom for Oracle replication only in Oracle Enterprise Edition. Standard Edition 2 isn't supported.
- We strongly recommend that you implement a VPN tunnel to encrypt communication between your primary and standby instances. For more information, see Configuring a VPN tunnel between RDS Custom for Oracle primary and replica instances.
- Don't modify the `RDS_DATAGUARD` user. This user is reserved for RDS Custom for Oracle automation. Modifying this user can result in undesired outcomes, such as an inability to create Oracle replicas for your RDS Custom for Oracle DB instance.
- Don't change the replication user password. It is required to administer the Oracle Data Guard configuration on the RDS Custom host. If you change the password, RDS Custom for Oracle might put your Oracle replica outside the support perimeter. For more information, see RDS Custom support perimeter.

  The password is stored in AWS Secrets Manager, tagged with the DB resource ID. Each Oracle replica has its own secret in Secrets Manager. The secret uses either of the following naming formats.

  ```
  do-not-delete-rds-custom-db-DB_resource_id-uuid-dg
  rds-custom!oracle-do-not-delete-DB_resource_id-uuid-dg
  ```

- Don't change the `DB_UNIQUE_NAME` for the primary DB instance. Changing the name causes any restore operation to become stuck.

- Don't specify the clause `STANDBYS=NONE` in a `CREATE PLUGGABLE DATABASE` command in an RDS Custom CDB. This way, if a failover occurs, your standby CDB contains all PDBs.

General limitations for RDS Custom for Oracle replication

RDS Custom for Oracle replicas have the following limitations:

- You can't create RDS Custom for Oracle replicas in read-only mode. However, you can manually change the mode of mounted replicas to read-only, and from read-only to mounted. For more information, see the documentation for the create-db-instance-read-replica AWS CLI command.
- You can't create cross-Region RDS Custom for Oracle replicas.
- You can't change the value of the Oracle Data Guard `CommunicationTimeout` parameter. This parameter is set to 15 seconds for RDS Custom for Oracle DB instances.

Networking requirements and limitations for RDS Custom for Oracle replication

Make sure that your network configuration supports RDS Custom for Oracle replicas. Consider the following:

- Make sure to enable port 1140 for both inbound and outbound communication within your virtual private cloud (VPC) for the primary DB instance and all of its replicas. This is required for Oracle Data Guard communication between read replicas.
- RDS Custom for Oracle validates the network while creating a Oracle replica. If the primary DB instance and the new replica can't connect over the network, RDS Custom for Oracle doesn't create the replica and places it in the `INCOMPATIBLE_NETWORK` state.
- For external Oracle replicas, such as those you create on Amazon EC2 or on-premises, use another port and listener for Oracle Data Guard replication. Trying to use port 1140 could cause conflicts with RDS Custom automation.
- The `/rdsdbdata/config/tnsnames.ora` file contains network service names mapped to listener protocol addresses. Note the following requirements and recommendations:
  - Entries in `tnsnames.ora` prefixed with `rds_custom_` are reserved for RDS Custom when handling Oracle replica operations.

    When creating manual entries in `tnsnames.ora`, don't use this prefix.
  - In some cases, you might want to switch over or fail over manually, or use failover technologies such as Fast-Start Failover (FSFO). If so, make sure to manually synchronize

`tnsnames.ora` entries from the primary DB instance to all of the standby instances. This recommendation applies to both Oracle replicas managed by RDS Custom and to external Oracle replicas.

RDS Custom automation updates `tnsnames.ora` entries on only the primary DB instance. Make sure also to synchronize when you add or remove a Oracle replica.

If you don't synchronize the `tnsnames.ora` files and switch over or fail over manually, Oracle Data Guard on the primary DB instance might not be able to communicate with the Oracle replicas.

External replica limitations for RDS Custom for Oracle

RDS Custom for Oracle external replicas, which include on-premises replicas, have the following limitations:

- RDS Custom for Oracle doesn't detect instance role changes upon manual failover, such as FSFO, for external Oracle replicas.

  RDS Custom for Oracle does detect changes for managed replicas. The role change is noted in the event log. You can also see the new state by using the describe-db-instances AWS CLI command.
- RDS Custom for Oracle doesn't detect high replication lag for external Oracle replicas.

  RDS Custom for Oracle does detect lag for managed replicas. High replication lag produces the `Replication has stopped` event. You can also see the replication status by using the describe-db-instances AWS CLI command, but there might be a delay for it to be updated.
- RDS Custom for Oracle doesn't promote external Oracle replicas automatically if you delete your primary DB instance.

  The automatic promotion feature is available only for managed Oracle replicas. For information about promoting Oracle replicas manually, see the white paper Enabling high availability with Data Guard on Amazon RDS Custom for Oracle.

Promoting an RDS Custom for Oracle replica to a standalone DB instance

Just as with RDS for Oracle, you can promote an RDS Custom for Oracle replica to a standalone DB instance. When you promote a Oracle replica, RDS Custom for Oracle reboots the DB instance

before it becomes available. For more information about promoting Oracle replicas, see Promoting a read replica to be a standalone DB instance.

When promoting a replica, note the following guidelines:

- Don't initiate a failover while RDS Custom for Oracle is promoting your replica. Otherwise, the promotion workflow could become stuck.
- Don't switch over your primary DB instance while RDS Custom for Oracle is promoting your Oracle replica. Otherwise, the promotion workflow could become stuck.
- Don't shut down your primary DB instance while RDS Custom for Oracle is promoting your Oracle replica. Otherwise, the promotion workflow could become stuck.
- Don't try to restart replication with your newly promoted DB instance as a target. After RDS Custom for Oracle promotes your Oracle replica, it becomes a standalone DB instance and no longer has the replica role.

Note the following limitations for RDS Custom for Oracle replica promotion:

- You can't promote a replica while RDS Custom for Oracle is backing it up.
- You can't change the backup retention period to `0` when you promote your Oracle replica.
- You can't promote your replica when it isn't in a healthy state.

  If you issue `delete-db-instance` on the primary DB instance, RDS Custom for Oracle validates that each managed Oracle replica is healthy and available for promotion. A replica might be ineligible for promotion because automation is paused or it is outside the support perimeter. In such cases, RDS Custom for Oracle publishes an event explaining the issue so that you can repair your Oracle replica manually.

The following steps show the general process for promoting a Oracle replica to a DB instance:

1. Stop any transactions from being written to the primary DB instance.
2. Wait for RDS Custom for Oracle to apply all updates to your Oracle replica.
3. Promote your Oracle replica by choosing the **Promote** option on the Amazon RDS console, the AWS CLI command `promote-read-replica`, or the `PromoteReadReplica` Amazon RDS API operation.

Promoting a Oracle replica takes a few minutes to complete. During the process, RDS Custom for Oracle stops replication and reboots your replica. When the reboot completes, the Oracle replica is available as a standalone DB instance. For information about troubleshooting replica promotion, see Troubleshooting replica promotion for RDS Custom for Oracle.

Console

To promote an RDS Custom for Oracle replica to a standalone DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the Amazon RDS console, choose **Databases**.

   The **Databases** pane appears. Each Oracle replica shows **Replica** in the **Role** column.
3. Choose the RDS Custom for Oracle replica that you want to promote.
4. For **Actions**, choose **Promote**.
5. On the **Promote Oracle replica** page, enter the backup retention period and the backup window for the newly promoted DB instance. You can't set this value to **0**.
6. When the settings are as you want them, choose **Promote Oracle replica**.

AWS CLI

To promote your RDS Custom for Oracle replica to a standalone DB instance, use the AWS CLI `promote-read-replica` command.

Example

For Linux, macOS, or Unix:

```
aws rds promote-read-replica \
--db-instance-identifier my-custom-read-replica \
--backup-retention-period 2 \
--preferred-backup-window 23:00-24:00
```

For Windows:

```
aws rds promote-read-replica ^
--db-instance-identifier my-custom-read-replica ^
```

```
--backup-retention-period 2 ^
--preferred-backup-window 23:00-24:00
```

RDS API

To promote your RDS Custom for Oracle replica to be a standalone DB instance, call the Amazon RDS API `PromoteReadReplica` operation with the required parameter `DBInstanceIdentifier`.

Configuring a VPN tunnel between RDS Custom for Oracle primary and replica instances

A VPN tunnel is an encrypted connection between two or more devices over a network. To ensure the highest level of security for your Oracle Data Guard instances in RDS Custom for Oracle, we strongly recommend that you implement a VPN tunnel to encrypt communication between your primary and standby instances. The tunnel serves as a safeguard for sensitive data as it travels the network between instances. While this configuration is optional, we recommend it as a best practice to achieve data security and regulatory compliance.

Make sure you meet the following prerequisites:

- You have root access to the primary and standby hosts.
- You have the technical expertise to run the `ipsec` command.

To configure a VPN tunnel between a primary and replica in RDS Custom for Oracle

1. Add the security groups for both the primary instance and standby instance to the allow list using the following rules:

```
ACTION FLOW SOURCE PROTO PORT

ALLOW ingress this-SG 50 (ESP) all (N/A)
ALLOW egress this-SG 50 (ESP) all (N/A)

ALLOW ingress this-SG 17 (UDP) 500 (IKE)
ALLOW egress this-SG 17 (UDP) 500 (IKE)
```

2. Switch to the root user.

```
$ sudo su - root
```

3. Run the following commands on both the primary instance and the standby instance to initialize the Network Security Services (NSS) database under the user `root`.

```
ipsec initnss --nssdir /etc/ipsec.d
```

4. Generate RSA keys as follows:

   a. On the primary instance, generate the keys using either of the following `ipsec` commands, depending on your OS version.

```
ipsec newhostkey --nssdir /etc/ipsec.d       ## for Oracle Linux Version 8
ipsec newhostkey --output /etc/ipsec.secrets ## for Oracle Linux version 7.9
```

   b. Obtain the public key, which you need to create the configuration. In the following example, the primary instance is `left` because in `ipsec` parlance, `left` refers to the device you are currently configuring, and `right` refers to the device at the other end of the tunnel.

```
ipsec showhostkey --left --ckaid ckaid-returned-in-last-statement
```

   c. On the standby instance, generate keys for the standby instance.

```
ipsec newhostkey --nssdir /etc/ipsec.d       ## for Oracle Linux Version 8
ipsec newhostkey --output /etc/ipsec.secrets ## for Oracle Linux version 7.9
```

   d. Obtain the public key for the standby instance, which you need to create the configuration. In the following example, the standby instance is `right` because it refers to the device at the other end of the tunnel.

```
ipsec showhostkey --right --ckaid ckaid-returned-in-last-statement
```

5. Based on the RSA keys that you obtained, generate the configuration. The configuration is identical for both the primary instance and the standby instance. You can find the primary instance IPv4 address and standby instance IPv4 address in the AWS console.

   On both the primary instance and the standby instance, save the following configuration to the file `/etc/ipsec.d/custom-fb-tunnel.conf`.

```
conn custom-db-tunnel
 type=transport
```

```
 auto=add
 authby=rsasig
 left=IPV4-for-primary
 leftrsasigkey=RSA-key-generated-on-primary
 right=IPV4-for-standby
 rightrsasigkey=RSA-key-generated-on-standby
```

6. On both the primary instance and the standby instance, start the `ipsec` daemon on both hosts.

```
ipsec setup start
```

7. Start the tunnel on either the primary instance or the standby instance. The output should look similar to the following.

```
[root@ip-172-31-6-81 ~]# ipsec auto --up custom-db-tunnel
181 "custom-db-tunnel" #1: initiating IKEv2 connection
181 "custom-db-tunnel" #1: sent IKE_SA_INIT request to 172.31.32.196:500
182 "custom-db-tunnel" #1: sent IKE_AUTH request {cipher=AES_GCM_16_256 integ=n/a
 prf=HMAC_SHA2_512 group=DH19}
003 "custom-db-tunnel" #1: initiator established IKE SA; authenticated peer '3584-
bit PKCS#1 1.5 RSA with SHA1' signature using preloaded certificate '172.31.32.196'
004 "custom-db-tunnel" #2: initiator established Child SA using #1; IPsec transport
 [172.31.6.81-172.31.6.81:0-65535 0] -> [172.31.32.196-172.31.32.196:0-65535 0]
 {ESP/ESN=>0xda9c4815 <0xb742ca42 xfrm=AES_GCM_16_256-NONE DPD=passive}
[root@ip-172-31-6-81 ~]#
```

Backing up and restoring an Amazon RDS Custom for Oracle DB instance

Like Amazon RDS, RDS Custom creates and saves automated backups of your RDS Custom for Oracle DB instance during the backup window of your DB instance. You can also back up your DB instance manually.

The procedure is identical to taking a snapshot of an Amazon RDS DB instance. The first snapshot of an RDS Custom DB instance contains the data for the full DB instance. Subsequent snapshots are incremental.

Restore DB snapshots using either the AWS Management Console or the AWS CLI.

Topics

- Creating an RDS Custom for Oracle snapshot
- Restoring from an RDS Custom for Oracle DB snapshot
- Restoring an RDS Custom for Oracle instance to a point in time
- Deleting an RDS Custom for Oracle snapshot
- Deleting RDS Custom for Oracle automated backups

Creating an RDS Custom for Oracle snapshot

RDS Custom for Oracle creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. When your DB instance contains a container database (CDB), the snapshot of the instance includes the root CDB and all PDBs.

When you create an RDS Custom for Oracle snapshot, specify which RDS Custom DB instance to back up. Give your snapshot a name so you can restore from it later.

When you create a snapshot, RDS Custom for Oracle creates an Amazon EBS snapshot for every volume attached to the DB instance. RDS Custom for Oracle uses the EBS snapshot of the root volume to register a new Amazon Machine Image (AMI). To make snapshots easy to associate with a specific DB instance, they're tagged with `DBSnapshotIdentifier`, `DbiResourceId`, and `VolumeType`.

Creating a DB snapshot results in a brief I/O suspension. This suspension can last from a few seconds to a few minutes, depending on the size and class of your DB instance. The snapshot creation time varies with the size of your database. Because the snapshot includes the entire

storage volume, the size of files, such as temporary files, also affects snapshot creation time. To learn more about creating snapshots, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Create an RDS Custom for Oracle snapshot using the console or the AWS CLI.

Console

To create an RDS Custom snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. In the list of RDS Custom DB instances, choose the instance for which you want to take a snapshot.
4. For **Actions**, choose **Take snapshot**.

   The **Take DB snapshot** window appears.
5. For **Snapshot name**, enter the name of the snapshot.
6. Choose **Take snapshot**.

AWS CLI

You create a snapshot of an RDS Custom DB instance by using the create-db-snapshot AWS CLI command.

Specify the following options:

- `--db-instance-identifier` – Identifies which RDS Custom DB instance you are going to back up
- `--db-snapshot-identifier` – Names your RDS Custom snapshot so you can restore from it later

In this example, you create a DB snapshot called *`my-custom-snapshot`* for an RDS Custom DB instance called *`my-custom-instance`*.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-snapshot \
    --db-instance-identifier my-custom-instance \
    --db-snapshot-identifier my-custom-snapshot
```

For Windows:

```
aws rds create-db-snapshot ^
    --db-instance-identifier my-custom-instance ^
    --db-snapshot-identifier my-custom-snapshot
```

Restoring from an RDS Custom for Oracle DB snapshot

When you restore an RDS Custom for Oracle DB instance, you provide the name of the DB snapshot and a name for the new instance. You can't restore from a snapshot to an existing RDS Custom DB instance. A new RDS Custom for Oracle DB instance is created when you restore.

The restore process differs in the following ways from restore in Amazon RDS:

- Before restoring a snapshot, RDS Custom for Oracle backs up existing configuration files. These files are available on the restored instance in the directory `/rdsdbdata/config/backup`. RDS Custom for Oracle restores the DB snapshot with default parameters and overwrites the previous database configuration files with existing ones. Thus, the restored instance doesn't preserve custom parameters and changes to database configuration files.
- The restored database has the same name as in the snapshot. You can't specify a different name. (For RDS Custom for Oracle, the default is `ORCL`.)

Console

To restore an RDS Custom DB instance from a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.
5. On the **Restore DB instance** page, for **DB instance identifier**, enter the name for your restored RDS Custom DB instance.

6. Choose **Restore DB instance**.

AWS CLI

You restore an RDS Custom DB snapshot by using the restore-db-instance-from-db-snapshot AWS CLI command.

If the snapshot you are restoring from is for a private DB instance, make sure to specify both the correct `db-subnet-group-name` and `no-publicly-accessible`. Otherwise, the DB instance defaults to publicly accessible. The following options are required:

- `db-snapshot-identifier` – Identifies the snapshot from which to restore
- `db-instance-identifier` – Specifies the name of the RDS Custom DB instance to create from the DB snapshot
- `custom-iam-instance-profile` – Specifies the instance profile associated with the underlying Amazon EC2 instance of an RDS Custom DB instance.

The following code restores the snapshot named `my-custom-snapshot` for `my-custom-instance`.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \
  --db-snapshot-identifier my-custom-snapshot \
  --db-instance-identifier my-custom-instance \
  --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance \
  --no-publicly-accessible
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^
  --db-snapshot-identifier my-custom-snapshot ^
  --db-instance-identifier my-custom-instance ^
  --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance ^
  --no-publicly-accessible
```

Restoring an RDS Custom for Oracle instance to a point in time

You can restore a DB instance to a specific point in time (PITR), creating a new DB instance. To support PITR, your DB instances must have backup retention set to a nonzero value.

The latest restorable time for an RDS Custom for Oracle DB instance depends on several factors, but is typically within 5 minutes of the current time. To see the latest restorable time for a DB instance, use the AWS CLI describe-db-instances command and look at the value returned in the `LatestRestorableTime` field for the DB instance. To see the latest restorable time for each DB instance in the Amazon RDS console, choose **Automated backups**.

You can restore to any point in time within your backup retention period. To see the earliest restorable time for each DB instance, choose **Automated backups** in the Amazon RDS console.

For general information about PITR, see Restoring a DB instance to a specified time for Amazon RDS.

Topics

- PITR considerations for RDS Custom for Oracle

PITR considerations for RDS Custom for Oracle

In RDS Custom for Oracle, PITR differs in the following important ways from PITR in Amazon RDS:

- The restored database has the same name as in the source DB instance. You can't specify a different name. The default is `ORCL`.
- `AWSRDSCustomIamRolePolicy` requires new permissions. For more information, see Step 2: Add an access policy to AWSRDSCustomInstanceRoleForRdsCustomInstance.
- All RDS Custom for Oracle DB instances must have backup retention set to a nonzero value.
- If you change the operating system or DB instance time zone, PITR might not work. For information about changing time zones, see Oracle time zone.
- If you set automation to `ALL_PAUSED`, RDS Custom pauses the upload of archived redo log files, including logs created before the latest restorable time (LRT). We recommend that you pause automation for a brief period.

  To illustrate, assume that your LRT is 10 minutes ago. You pause automation. During the pause, RDS Custom doesn't upload archived redo logs. If your DB instance crashes, you can only recover

to a time before the LRT that existed when you paused. When you resume automation, RDS Custom resumes uploading logs. The LRT advances. Normal PITR rules apply.

- In RDS Custom, you can manually specify an arbitrary number of hours to retain archived redo logs before RDS Custom deletes them after upload. Specify the number of hours as follows:

  1. Create a text file named `/opt/aws/rdscustomagent/config/redo_logs_custom_configuration.json`.
  2. Add a JSON object in the following format: `{"archivedLogRetentionHours" : "num_of_hours"}`. The number must be an integer in the range 1–840.

- Assume that you plug a non-CDB into a container database (CDB) as a PDB and then attempt PITR. The operation succeeds only if you previously backed up the PDB. After you create or modify a PDB, we recommend that you always back it up.
- We recommend that you don't customize database initialization parameters. For example, modifying the following parameters affects PITR:
  - `CONTROL_FILE_RECORD_KEEP_TIME` affects the rules for uploading and deleting logs.
  - `LOG_ARCHIVE_DEST_`*n* doesn't support multiple destinations.
  - `ARCHIVE_LAG_TARGET` affects the latest restorable time. `ARCHIVE_LAG_TARGET` is set to `300` because the recovery point objective (RPO) is 5 minutes. To honor this objective, RDS switches the online redo log every 5 minutes and stores it in an Amazon S3 bucket. If the frequency of the log switch causes a performance issue for your RDS Custom for Oracle database, you can scale your DB instance and storage to one with higher IOPS and throughput. If necessary for your recovery plan, you can adjust the setting of the `ARCHIVE_LAG_TARGET` initialization parameter to a value from 60–7200.
- If you customize database initialization parameters, we strongly recommend that you customize only the following:
  - `COMPATIBLE`
  - `MAX_STRING_SIZE`
  - `DB_FILES`
  - `UNDO_TABLESPACE`
  - `ENABLE_PLUGGABLE_DATABASE`
  - `CONTROL_FILES`
  - `AUDIT_TRAIL`
  - `AUDIT_TRAIL_DEST`

For all other initialization parameters, RDS Custom restores the default values. If you modify a parameter that isn't in the preceding list, it might have an adverse effect on PITR and lead to unpredictable results. For example, `CONTROL_FILE_RECORD_KEEP_TIME` affects the rules for uploading and deleting logs.

You can restore an RDS Custom DB instance to a point in time using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To restore an RDS Custom DB instance to a specified time

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Automated backups**.
3. Choose the RDS Custom DB instance that you want to restore.
4. For **Actions**, choose **Restore to point in time**.

   The **Restore to point in time** window appears.
5. Choose **Latest restorable time** to restore to the latest possible time, or choose **Custom** to choose a time.

   If you chose **Custom**, enter the date and time to which you want to restore the instance.

   Times are shown in your local time zone, which is indicated by an offset from Coordinated Universal Time (UTC). For example, UTC-5 is Eastern Standard Time/Central Daylight Time.
6. For **DB instance identifier**, enter the name of the target restored RDS Custom DB instance. The name must be unique.
7. Choose other options as needed, such as DB instance class.
8. Choose **Restore to point in time**.

AWS CLI

You restore a DB instance to a specified time by using the restore-db-instance-to-point-in-time AWS CLI command to create a new RDS Custom DB instance.

Use one of the following options to specify the backup to restore from:

- `--source-db-instance-identifier mysourcedbinstance`
- `--source-dbi-resource-id dbinstanceresourceID`
- `--source-db-instance-automated-backups-arn backupARN`

The `custom-iam-instance-profile` option is required.

The following example restores `my-custom-db-instance` to a new DB instance named `my-restored-custom-db-instance`, as of the specified time.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-to-point-in-time \
    --source-db-instance-identifier my-custom-db-instance\
    --target-db-instance-identifier my-restored-custom-db-instance \
    --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance \
    --restore-time 2022-10-14T23:45:00.000Z
```

For Windows:

```
aws rds restore-db-instance-to-point-in-time ^
    --source-db-instance-identifier my-custom-db-instance ^
    --target-db-instance-identifier my-restored-custom-db-instance ^
    --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance ^
    --restore-time 2022-10-14T23:45:00.000Z
```

Deleting an RDS Custom for Oracle snapshot

You can delete DB snapshots managed by RDS Custom for Oracle when you no longer need them. The deletion procedure is the same for both Amazon RDS and RDS Custom DB instances.

The Amazon EBS snapshots for the binary and root volumes remain in your account for a longer time because they might be linked to some instances running in your account or to other RDS Custom for Oracle snapshots. These EBS snapshots are automatically deleted after they're no longer related to any existing RDS Custom for Oracle resources (DB instances or backups).

Console

To delete a snapshot of an RDS Custom DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB snapshot that you want to delete.
4. For **Actions**, choose **Delete snapshot**.
5. Choose **Delete** on the confirmation page.

AWS CLI

To delete an RDS Custom snapshot, use the AWS CLI command delete-db-snapshot.

The following option is required:

- `--db-snapshot-identifier` – The snapshot to be deleted

The following example deletes the `my-custom-snapshot` DB snapshot.

Example

For Linux, macOS, or Unix:

```
aws rds delete-db-snapshot \
  --db-snapshot-identifier my-custom-snapshot
```

For Windows:

```
aws rds delete-db-snapshot ^
  --db-snapshot-identifier my-custom-snapshot
```

Deleting RDS Custom for Oracle automated backups

You can delete retained automated backups for RDS Custom for Oracle when they are no longer needed. The procedure is the same as the procedure for deleting Amazon RDS backups.

Console

To delete a retained automated backup

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Automated backups**.
3. Choose **Retained**.
4. Choose the retained automated backup that you want to delete.
5. For **Actions**, choose **Delete**.
6. On the confirmation page, enter **`delete me`** and choose **Delete**.

AWS CLI

You can delete a retained automated backup by using the AWS CLI command delete-db-instance-automated-backup.

The following option is used to delete a retained automated backup:

- `--dbi-resource-id` – The resource identifier for the source RDS Custom DB instance.

  You can find the resource identifier for the source DB instance of a retained automated backup by using the AWS CLI command describe-db-instance-automated-backups.

The following example deletes the retained automated backup with source DB instance resource identifier `custom-db-123ABCEXAMPLE`.

Example

For Linux, macOS, or Unix:

```
aws rds delete-db-instance-automated-backup \
    --dbi-resource-id custom-db-123ABCEXAMPLE
```

For Windows:

```
aws rds delete-db-instance-automated-backup ^
    --dbi-resource-id custom-db-123ABCEXAMPLE
```

Working with option groups in RDS Custom for Oracle

RDS Custom uses option groups to enable and configure additional features. An *option group* specifies features, called options, that are available for an RDS Custom for Oracle DB instance. Options can have settings that specify how the option works. When you associate an RDS Custom for Oracle DB instance with an option group, the specified options and option settings are enabled for this instance. For general information about option groups in Amazon RDS, see Working with option groups.

Topics

- Overview of option groups in RDS Custom for Oracle
- Oracle time zone

Overview of option groups in RDS Custom for Oracle

To enable options for your Oracle database, add them to an option group, and then associate the option group with your DB instance. For more information, see Working with option groups.

Topics

- Summary of RDS Custom for Oracle options
- Basic steps for adding an option to an RDS Custom for Oracle DB instance
- Creating an option group for in RDS Custom for Oracle
- Associating an option group with an RDS Custom for Oracle DB instance

Summary of RDS Custom for Oracle options

RDS Custom for Oracle supports the following options for a DB instance.

Option	Option ID	Description
Oracle time zone	`Timezone`	The time zone used by your RDS Custom for Oracle DB instance.

Basic steps for adding an option to an RDS Custom for Oracle DB instance

The general procedure for adding an option to your RDS Custom for Oracle DB instance is the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with your DB instance when you create or modify it.

Creating an option group for in RDS Custom for Oracle

You can create a new option group that derives its settings from the default option group. You then add one or more options to the new option group. Or, if you already have an existing option group, you can copy that option group with all of its options to a new option group. To learn how to copy an option group, see Copying an option group.

The default option groups for RDS Custom for Oracle are the following:

- `default:custom-oracle-ee`
- `default:custom-oracle-se2`
- `default:custom-oracle-ee-cdb`
- `default:custom-oracle-se2-cdb`

When you create an option group, the settings are derived from the default option group. After you have added the `TIME_ZONE` option, you can then associate the option group with your DB instance.

Console

One way of creating an option group is by using the AWS Management Console.

To create a new option group by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** window, do the following:

a. For **Name**, type a name for the option group that is unique within your AWS account. The name can contain only letters, digits, and hyphens.

b. For **Description**, type a brief description of the option group. The description is used for display purposes.

c. For **Engine**, choose any of the following RDS Custom for Oracle DB engines:

- **custom-oracle-ee**
- **custom-oracle-se2**
- **custom-oracle-ee-cdb**
- **custom-oracle-se2-cdb**

d. For **Major engine version**, choose a major engine version supported by RDS Custom for Oracle. For more information, see Supported Regions and DB engines for RDS Custom for Oracle.

5. To continue, choose **Create**. To cancel the operation instead, choose **Cancel**.

AWS CLI

To create an option group, use the AWS CLI `create-option-group` command with the following required parameters.

- `--option-group-name`
- `--engine-name`
- `--major-engine-version`
- `--option-group-description`

Example

The following example creates an option group named `testoptiongroup`, which is associated with the Oracle Enterprise Edition DB engine. The description is enclosed in quotation marks.

For Linux, macOS, or Unix:

```
aws rds create-option-group \
    --option-group-name testoptiongroup \
    --engine-name custom-oracle-ee-cdb \
    --major-engine-version 19 \
```

```
    --option-group-description "Test option group for a Custom Oracle CDB"
```

For Windows:

```
aws rds create-option-group ^
    --option-group-name testoptiongroup ^
    --engine-name custom-oracle-ee-cdb ^
    --major-engine-version 19 ^
    --option-group-description "Test option group for a Custom Oracle CDB"
```

RDS API

To create an option group, call the Amazon RDS API `CreateOptionGroup` operation.

Associating an option group with an RDS Custom for Oracle DB instance

You can associate your option group with a new or existing DB instance:

- For a new DB instance, apply the option group when you create the instance. For more information, see Creating an RDS Custom for Oracle DB instance.
- For an existing DB instance, apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying your RDS Custom for Oracle DB instance.

Oracle time zone

To change the system time zone used by your RDS Custom for Oracle DB instance, use the time zone option. For example, you might change the time zone of a DB instance to be compatible with an on-premises environment, or a legacy application. The time zone option changes the time zone at the host level. Changing the time zone impacts all date columns and values, including `SYSDATE` and `SYSTIMESTAMP`.

Topics

- Time zone option settings in RDS Custom for Oracle
- Available time zones in RDS Custom for Oracle
- Considerations for setting the time zone in RDS Custom for Oracle
- Limitations for the time zone setting in RDS Custom for Oracle
- Adding the time zone option to an option group
- Removing the time zone option

Time zone option settings in RDS Custom for Oracle

Amazon RDS supports the following settings for the time zone option.

Option setting	Valid values	Description
TIME_ZONE	One of the available time zones. For the full list, see Available time zones in RDS Custom for Oracle.	The new time zone for your DB instance.

Available time zones in RDS Custom for Oracle

You can use the following values for the time zone option.

Zone	Time zone
Africa	Africa/Cairo, Africa/Casablanca, Africa/Harare, Africa/Lagos, Africa/Luanda, Africa/Monrovia, Africa/Nairobi, Africa/Tripoli, Africa/Windhoek
America	America/Araguaina, America/Argentina/Buenos_Aires, America/Asuncion, America/Bogota, America/Caracas, America/Chicago, America/Chihuahua, America/Cuiaba, America/Denver, America/Detroit, America/Fortaleza, America/Godthab, America/Guatemala, America/Halifax, America/Lima, America/Los_Angeles, America/Manaus, America/Matamoros, America/Mexico_City, America/Monterrey, America/Montevideo, America/New_York, America/Phoenix, America/Santiago, America/Sao_Paulo, America/Tijuana, America/Toronto
Asia	Asia/Amman, Asia/Ashgabat, Asia/Baghdad, Asia/Baku, Asia/Bangkok, Asia/Beirut, Asia/Calcutta, Asia/Damascus, Asia/Dhaka, Asia/Hong_Kong, Asia/Irkutsk, Asia/Jakarta, Asia/Jerusalem, Asia/Kabul, Asia/Karachi, Asia/Kathmandu, Asia/Kolkata, Asia/Krasnoyarsk, Asia/Magadan, Asia/Manila, Asia/Muscat, Asia/Novosibirsk, Asia/Rangoon, Asia/Riyadh, Asia/Seoul, Asia/Shanghai, Asia/Singapore, Asia/Taipei, Asia/Tehran, Asia/Tokyo, Asia/Ulaanbaatar, Asia/Vladivostok, Asia/Yakutsk, Asia/Yerevan
Atlantic	Atlantic/Azores, Atlantic/Cape_Verde

Zone	Time zone
Australia	Australia/Adelaide, Australia/Brisbane, Australia/Darwin, Australia/Eucla, Australia/Hobart, Australia/Lord_Howe, Australia/Perth, Australia/Sydney
Brazil	Brazil/DeNoronha, Brazil/East
Canada	Canada/Newfoundland, Canada/Saskatchewan
Etc	Etc/GMT-3
Europe	Europe/Amsterdam, Europe/Athens, Europe/Berlin, Europe/Dublin, Europe/Helsinki, Europe/Kaliningrad, Europe/London, Europe/Madrid, Europe/Moscow, Europe/Paris, Europe/Prague, Europe/Rome, Europe/Sarajevo
Pacific	Pacific/Apia, Pacific/Auckland, Pacific/Chatham, Pacific/Fiji, Pacific/Guam, Pacific/Honolulu, Pacific/Kiritimati, Pacific/Marquesas, Pacific/Samoa, Pacific/Tongatapu, Pacific/Wake
US	US/Alaska, US/Central, US/East-Indiana, US/Eastern, US/Pacific
UTC	UTC

Considerations for setting the time zone in RDS Custom for Oracle

If you choose to set the time zone for your DB instance, consider the following:

- When you add the time zone option, a brief outage occurs while your DB instance is automatically restarted.
- If you accidentally set the time zone incorrectly, you must recover your DB instance to its previous time zone setting. For this reason, we strongly suggest that you to use one of the following strategies before you add the time zone option to your instance:
  - If your RDS Custom for Oracle DB instance uses the default option group, take a snapshot of your DB instance. For more information, see Creating an RDS Custom for Oracle snapshot.
  - If your DB instance currently uses a nondefault option group, take a snapshot of your DB instance, and then create a new option group with the time zone option.
- We strongly recommend that you back up your DB instance manually after applying the `Timezone` option.

- We strongly recommend that you to test the time zone option on a test DB instance before you add it to a production DB instance. Adding the time zone option can cause problems with tables that use system date to add dates or times. We recommend that you analyze your data and applications to assess the impact of changing the time zone.

Limitations for the time zone setting in RDS Custom for Oracle

Note the following limitations:

- You can't change your timezone directly on your host without moving it outside the support perimeter. To change your database timezone, you must create an option group.
- Because the time zone option is a persistent option (but not a permanent option), you can't do the following:
  - Remove the option from an option group after you add the option.
  - Modify the time zone setting of the option to a different time zone.
- You can't associate multiple option groups with your RDS Custom for Oracle DB instance.
- You can't set the time zone for individual PDBs within a CDB.

Adding the time zone option to an option group

The default option groups for RDS Custom for Oracle are the following:

- `default:custom-oracle-ee`
- `default:custom-oracle-se2`
- `default:custom-oracle-ee-cdb`
- `default:custom-oracle-se2-cdb`

When you create an option group, the settings are derived from the default option group. For general information about option groups in Amazon RDS, see Working with option groups.

Console

To add the time zone option to an option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Option groups**.
3. Choose the option group that you want to modify, and then choose **Add option**.
4. In the **Add option** window, do the following:
   a. Choose **Timezone**.
   b. In **Option settings**, choose a time zone.
   c. To enable the option on all associated RDS Custom for Oracle DB instances as soon as you add it, for **Apply Immediately**, choose **Yes**. If you choose **No** (the default), the option is enabled for each associated DB instances during its next maintenance window.
   d.

> **Important**
>
> If you add the time zone option to an existing option group that is already attached to one or more DB instances, a brief outage occurs while all the DB instances are automatically restarted.

5. When the settings are as you want them, choose **Add option**.
6. Back up the RDS Custom for Oracle DB instances whose time zones were updated. For more information, see Creating an RDS Custom for Oracle snapshot.

AWS CLI

The following example uses the AWS CLI add-option-to-option-group command to add the `Timezone` option and the `TIME_ZONE` option setting to an option group called `testoptiongroup`. The time zone is set to `America/Los_Angeles`.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name "testoptiongroup" \
    --options "OptionName=Timezone,OptionSettings=[{Name=TIME_ZONE,Value=America/
Los_Angeles}]" \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name "testoptiongroup" ^
```

```
    --options "OptionName=Timezone,OptionSettings=[{Name=TIME_ZONE,Value=America/
Los_Angeles}]" ^
    --apply-immediately
```

Removing the time zone option

The time zone option is a persistent option, but not a permanent option. You can't remove the option from an option group after you add it. To disassociate the old option group from your DB instance:

1. Create a new option group with an updated `Timezone` option.
2. Associate the new option group with your DB instance when you modify the instance.

Migrating an on-premises database to RDS Custom for Oracle

Before you migrate an on-premises Oracle database to RDS Custom for Oracle, consider the following factors:

- The amount of downtime the application can afford
- The size of the source database
- Network connectivity
- A requirement for a fallback plan
- The source and target Oracle database version and DB instance OS types
- Available replication tools, such as AWS Database Migration Service, Oracle GoldenGate, or third-party replication tools

Based on these factors, you can choose physical migration, logical migration, or a combination. If you choose physical migration, you can use the following techniques:

RMAN duplication

Active database duplication doesn't require a backup of your source database. It duplicates the live source database to the destination host by copying database files over the network to the auxiliary instance. The RMAN `DUPLICATE` command copies the required files as image copies or backup sets. To learn this technique, see the AWS blog post Physical migration of Oracle databases to Amazon RDS Custom using RMAN duplication.

Oracle Data Guard

In this technique, you back up a primary on-premises database and copy the backups to an Amazon S3 bucket. You then copy the backups to your RDS Custom for Oracle standby DB instance. After performing the necessary configuration, you manually switch over your primary database to your RDS Custom for Oracle standby database. To learn this technique, see the AWS blog post Physical migration of Oracle databases to Amazon RDS Custom using Data Guard.

For general information about logically importing data into RDS for Oracle, see Importing data into Oracle on Amazon RDS.

Upgrading a DB instance for Amazon RDS Custom for Oracle

You can upgrade an Amazon RDS Custom DB instance by modifying it to use a new custom engine version (CEV). For general information about upgrades, see Upgrading a DB instance engine version.

Topics

- Overview of upgrades in RDS Custom for Oracle
- Requirements for RDS Custom for Oracle upgrades
- Considerations for RDS Custom for Oracle database upgrades
- Considerations for RDS Custom for Oracle OS upgrades
- Viewing valid CEV upgrade targets for RDS Custom for Oracle DB instances
- Upgrading an RDS Custom for Oracle DB instance
- Viewing pending database upgrades for RDS Custom DB instances
- Troubleshooting an upgrade failure for an RDS Custom for Oracle DB instance

Overview of upgrades in RDS Custom for Oracle

With RDS Custom for Oracle, you can patch either your Oracle database or your DB instance operating system (OS) by creating new CEVs and then modifying your instance to use the new CEV.

Topics

- CEV upgrade options
- Patching without CEVs
- General steps for patching your DB instance with a CEV

CEV upgrade options

When you create a CEV for an upgrade, you have the following mutually exclusive options:

Database only

Reuse the Amazon Machine Image (AMI) currently in use by your DB instance, but specify different database binaries. RDS Custom allocates a new binary volume and then attaches it to

the existing Amazon EC2 instance. RDS Custom replaces the entire database volume with a new volume that uses your target database version.

OS only

Reuse the database binaries currently in use by your DB instance, but specify a different AMI. RDS Custom allocates a new Amazon EC2 instance, and then attaches the existing binary volume to the new instance. The existing database volume is retained.

If you want to upgrade both the OS and database, you must upgrade the CEV twice. You can either upgrade the OS and then the database or upgrade the database and then the OS.

> **Warning**
>
> When you patch your OS, you lose your root volume data and any existing OS customization. Thus, we strongly recommend that you don't use the root volume for installations or for storing permanent data or files. We also recommend that you back up your data before the upgrade.

Patching without CEVs

We strongly recommend that you upgrade your RDS Custom for Oracle DB instance using CEVs. RDS Custom for Oracle automation synchronizes the patch metadata with the database binary on your DB instance.

In special circumstances, RDS Custom supports applying a "one-off" database patch directly to the underlying Amazon EC2 instance directly using the OPatch utility. A valid use case might be a database patch that you want to apply immediately, but the RDS Custom team is upgrading the CEV feature, causing a delay. To apply a database patch manually, perform the following steps:

1. Pause RDS Custom automation.
2. Apply your patch to the database binaries on the Amazon EC2 instance.
3. Resume RDS Custom automation.

A disadvantage of the preceding technique is that you must apply the database patch manually to every instance that you want to upgrade. In contrast, when you create a new CEV, you can create or upgrade multiple DB instances using the same CEV.

General steps for patching your DB instance with a CEV

Whether you patch the OS or your database, perform the following basic steps:

1. Create a CEV that contains either of the following, depending on whether you're patching the database or OS:
   - The Oracle Database RU that you want to apply to your DB instance
   - A different AMI–either the latest available or one that you specify–and an existing CEV to use as a source

   Follow the steps in Creating a CEV.
2. (Optional for database patching) Check available engine version upgrades by running `describe-db-engine-versions`.
3. Start the patching process by running `modify-db-instance`.

   The status of the instance being patched differs as follows:
   - While RDS is patching the database, the status of the DB instance changes to **Upgrading**.
   - While RDS is patching the OS, the status of the DB instance changes to **Modifying**.

   When the DB instance has the status **Available**, patching is complete.
4. Confirm that your DB instance uses the new CEV by running `describe-db-instances`.

Requirements for RDS Custom for Oracle upgrades

When upgrading your RDS Custom for Oracle DB instance to a target CEV, make sure you meet the following requirements:

- The target CEV to which you are upgrading must exist.
- You must upgrade either the OS or the database in a single operation. Upgrading both the OS and the database in a single API call isn't supported.
- The target CEV must use the installation parameter settings that are in the manifest of the current CEV. For example, you can't upgrade a database that uses the default Oracle home to a CEV that uses a nondefault Oracle home.
- For database upgrades, the target CEV must use a new minor database version, not a new major version. For example, you can't upgrade from an Oracle Database 12c CEV to an Oracle Database 19c CEV. But you can upgrade from version 21.0.0.0.ru-2023-04.rur-2023-04.r1 to version 21.0.0.0.ru-2023-07.rur-2023-07.r1.

- For OS upgrades, the target CEV must use a different AMI but have the same major version.

Considerations for RDS Custom for Oracle database upgrades

If you plan to upgrade your database, consider the following:

- The currently supported operating system (OS) version is Oracle Linux 8. To continue receiving the latest security updates and patches from RDS Custom for Oracle, upgrade your DB instances to Oracle Linux 8 by specifying a CEV based on this OS. Oracle Database 12c Release 1 (12.1), Oracle Database Release 2 (12.2), and Oracle Database 19c are the only releases that support Oracle Linux 8. To migrate to the latest Oracle Linux 8 AMI, upgrade your OS to the latest AMI. For more information, see Upgrading an RDS Custom for Oracle DB instance.

  Oracle Linux 7.9 ended support on Dec 31, 2024. To continue running Oracle Linux 7 after the end of support, purchase an Oracle Extended Support license. You're responsible for security updates and must patch your RDS Custom for Oracle instances manually. For more information, see Lifetime Support Policy: Coverage for Oracle Open Source Service Offerings.
- When you upgrade the database binaries in your primary DB instance, RDS Custom for Oracle upgrades your read replicas automatically. When you upgrade the OS, however, you must upgrade the read replicas manually.
- When you upgrade a container database (CDB) to a new database version, RDS Custom for Oracle checks that all PDBs are open or could be opened. If these conditions aren't met, RDS Custom stops the check and returns the database to its original state without attempting the upgrade. If the conditions are met, RDS Custom patches the CDB root first, and then patches all other PDBs (including `PDB$SEED`) in parallel.

  After patching completes, RDS Custom attempts to open all PDBs. If any PDBs fail to open, you receive the following event: `The following PDBs failed to open: list-of-PDBs`. If RDS Custom fails to patch the CDB root or any PDBs, the instance is put into the `PATCH_DB_FAILED` state.
- You might want to perform a major database version upgrade and a conversion of non-CDB to CDB at the same time. In this case, we recommend that you proceed as follows:

  1. Create a new RDS Custom for Oracle DB instance that uses the Oracle multitenant architecture.
  2. Plug in a non-CDB into your CDB root, creating it as a PDB. Make sure that the non-CDB is the same major version as your CDB.

3. Convert your PDB by running the `noncdb_to_pdb.sql` Oracle SQL script.
4. Validate your CDB instance.
5. Upgrade your CDB instance.

Considerations for RDS Custom for Oracle OS upgrades

When you plan an OS upgrade, consider the following:

- You can't provide your own AMI for use in an RDS Custom for Oracle CEV. You can specify either the default AMI, which uses Oracle Linux 8, or an AMI that has been previously used by an RDS Custom for Oracle CEV.

  > **Note**
  >
  > RDS Custom for Oracle releases a new default AMI when common vulnerabilities and exposures are discovered. No fixed schedule is available or guaranteed. RDS Custom for Oracle tends to publish a new default AMI every 30 days.

- When you upgrade the OS in your primary DB instance, you must upgrade its associated read replicas manually.
- Reserve sufficient Amazon EC2 compute capacity for your instance type in your AZ before you begin patching the OS.

  When you create a Capacity Reservation, you specify the AZ, number of instances, and instance attributes (including instance type). For example, if your DB instance uses the underlying EC2 instance type r5.large, we recommend that you reserve EC2 capacity for r5.large in your AZ. During OS patching, RDS Custom creates one new host of type db.r5.large, which can become stuck if the AZ lacks EC2 capacity for this instance type. If you reserve EC2 capacity, you lower the risk of blocked patching caused by capacity constraints. For more information, see On-Demand Capacity Reservations in the *Amazon EC2 User Guide*.
- Back up your DB instance before you upgrade its OS. The upgrade removes your root volume data and any existing OS customizations.
- In the shared responsibility model, you're responsible for keeping your OS up to date. RDS Custom for Oracle doesn't mandate which patches you apply to your OS. If your RDS Custom for Oracle is functional, you can use the AMI associated with this CEV indefinitely.

Viewing valid CEV upgrade targets for RDS Custom for Oracle DB instances

You can see existing CEVs on the **Custom engine versions** page in the AWS Management Console.

You can also use the describe-db-engine-versions AWS CLI command to find valid CEVs to use when you upgrade your DB instances, as shown in the following example. This example assumes that you created a DB instance using the engine version `19.my_cev1`, and that the upgrade versions `19.my_cev2` and `19.my_cev` exist.

```
aws rds describe-db-engine-versions --engine custom-oracle-ee --engine-version
 19.my_cev1
```

The output resembles the following. The `ImageId` field shows the AMI ID.

```
{
    "DBEngineVersions": [
        {
            "Engine": "custom-oracle-ee",
            "EngineVersion": "19.my_cev1",
            ...
            "Image": {
                "ImageId": "ami-2345",
                "Status": "active"
            },
            "DBEngineVersionArn": "arn:aws:rds:us-west-2:123456789012:cev:custom-
oracle-ee/19.my_cev1/12a34b5c-67d8-90e1-2f34-gh56ijk78lm9"
            "ValidUpgradeTarget": [
                {
                    "Engine": "custom-oracle-ee",
                    "EngineVersion": "19.my_cev2",
                    "Description": "19.my_cev2 description",
                    "AutoUpgrade": false,
                    "IsMajorVersionUpgrade": false
                },
                {
                    "Engine": "custom-oracle-ee",
                    "EngineVersion": "19.my_cev3",
                    "Description": "19.my_cev3 description",
                    "AutoUpgrade": false,
                    "IsMajorVersionUpgrade": false
                }
            ]
```

```
...
```

Upgrading an RDS Custom for Oracle DB instance

To upgrade your RDS Custom for Oracle DB instance, modify it to use a new CEV. This CEV can contain either new database binaries or a new AMI. For example, to upgrade your Oracle Linux 7.9 DB instance to Oracle Linux 8, specify the latest AMI, which uses Oracle Linux 8. To upgrade the database and OS, you must perform two separate upgrades.

> **Note**
>
> If you upgrade the database, RDS Custom automatically upgrades read replicas after it upgrades the primary DB instance. If you upgrade the OS, you must upgrade the replicas manually.

Before you begin, review Requirements for RDS Custom for Oracle upgrades and Considerations for RDS Custom for Oracle database upgrades.

Console

To upgrade an RDS Custom for Oracle DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom for Oracle DB instance that you want to upgrade.
3. Choose **Modify**. The **Modify DB instance** page appears.
4. For **DB engine version**, choose a new CEV. Do the following:
   - If you are patching the database, make sure that the CEV specifies database binaries that are different from those used by your DB instance, and doesn't specify an AMI that is different from the AMI currently used by your DB instance.
   - If you are patching the OS, make sure that the CEV specifies an AMI that is different from the AMI currently used by your DB instance, and doesn't specify different database binaries.

> **Warning**
>
> When you patch your OS, you lose your root volume data and any existing OS customization.

5. Choose **Continue** to check the summary of modifications.

   Choose **Apply immediately** to apply the changes immediately.
6. If your changes are correct, choose **Modify DB instance**. Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

The following examples show possible upgrade scenarios. The examples assume that you created an RDS Custom for Oracle DB instance with the following characteristics:

- DB instance named `my-custom-instance`
- CEV named `19.my_cev1`
- Oracle Database 19c using the non-CDB architecture
- Oracle Linux 8 using AMI `ami-1234`

The latest service-provided AMI is `ami-2345`. You can find AMIs by running the CLI command `describe-db-engine-versions`.

Topics

- Upgrading the OS
- Upgrading the database

Upgrading the OS

In this example, you want to upgrade `ami-1234` to `ami-2345`, which is the latest service-provided AMI. Because you are upgrading the OS, the database binaries for `ami-1234` and `ami-2345` must be the same. You create a new CEV named `19.my_cev2` based on `19.my_cev1`.

Example

For Linux, macOS, or Unix:

```
aws rds create-custom-db-engine-version \
    --engine custom-oracle-ee \
    --engine-version 19.my_cev2 \
    --description "Non-CDB CEV based on ami-2345" \
    --kms-key-id key-name \
    --source-custom-db-engine-version-identifer arn:aws:rds:us-
west-2:123456789012:cev:custom-oracle-ee/19.my_cev1/12345678-ab12-1234-cde1-
abcde123456789 \
    --image-id ami-2345
```

For Windows:

```
aws rds create-custom-db-engine-version ^
    --engine custom-oracle-ee ^
    --engine-version 19.my_cev2 ^
    --description "Non-CDB CEV based on ami-2345" ^
    --kms-key-id key-name ^
    --source-custom-db-engine-version-identifer arn:aws:rds:us-
west-2:123456789012:cev:custom-oracle-ee/19.my_cev1/12345678-ab12-1234-cde1-
abcde123456789 ^
    --image-id ami-2345
```

To upgrade an RDS Custom DB instance, use the modify-db-instance AWS CLI command with the following parameters:

- `--db-instance-identifier` – Specify the RDS Custom for Oracle DB instance to be upgraded.
- `--engine-version` – Specify the CEV that has the new AMI.
- `--no-apply-immediately` | `--apply-immediately` – Specify whether to perform the upgrade immediately or wait until the scheduled maintenance window.

The following example upgrades `my-custom-instance` to version `19.my_cev2`. Only the OS is upgraded.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
```

```
    --engine-version 19.my_cev2 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --engine-version 19.my_cev2 ^
    --apply-immediately
```

Upgrading the database

In this example, you want to apply Oracle patch p35042068 to your RDS for Oracle DB instance. Because you upgraded your OS in Upgrading the OS, your DB instance is currently using `19.my_cev2`, which is based on `ami-2345`. You create a new CEV named `19.my_cev3` that also uses `ami-2345`, but you specify a new JSON manifest in the `$MANIFEST` environment variable. Thus, only the database binaries different in your new CEV and the CEV that your instance is currently using.

Example

For Linux, macOS, or Unix:

```
aws rds create-custom-db-engine-version \
    --engine custom-oracle-ee \
    --engine-version 19.my_cev3 \
    --description "Non-CDB CEV with p35042068 based on ami-2345" \
    --kms-key-id key-name \
    --image-id ami-2345 \
    --manifest $MANIFEST
```

For Windows:

```
aws rds create-custom-db-engine-version ^
    --engine custom-oracle-ee ^
    --engine-version 19.my_cev3 ^
    --description "Non-CDB CEV with p35042068 based on ami-2345" ^
    --kms-key-id key-name ^
    --image-id ami-2345 ^
    --manifest $MANIFEST
```

The following example upgrades `my-custom-instance` to engine version `19.my_cev3`. Only the database is upgraded.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --engine-version 19.my_cev3 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --engine-version 19.my_cev3 ^
    --apply-immediately
```

Viewing pending database upgrades for RDS Custom DB instances

You can see pending database upgrades for your Amazon RDS Custom DB instances by using the describe-db-instances or describe-pending-maintenance-actions AWS CLI command.

However, this approach doesn't work if you used the `--apply-immediately` option or if the upgrade is in progress.

The following `describe-db-instances` command shows pending database upgrades for `my-custom-instance`.

```
aws rds describe-db-instances --db-instance-identifier my-custom-instance
```

The output resembles the following.

```
{
    "DBInstances": [
        {
           "DBInstanceIdentifier": "my-custom-instance",
            "EngineVersion": "19.my_cev1",
            ...
```

```
            "PendingModifiedValues": {
                "EngineVersion": "19.my_cev3"
            ...
            }
        }
    ]
}
```

Troubleshooting an upgrade failure for an RDS Custom for Oracle DB instance

If an RDS Custom DB instance upgrade fails, an RDS event is generated and the DB instance status becomes `upgrade-failed`.

You can see this status by using the describe-db-instances AWS CLI command, as shown in the following example.

```
aws rds describe-db-instances --db-instance-identifier my-custom-instance
```

The output resembles the following.

```
{
    "DBInstances": [
        {
           "DBInstanceIdentifier": "my-custom-instance",
            "EngineVersion": "19.my_cev1",
            ...
            "PendingModifiedValues": {
                "EngineVersion": "19.my_cev3"
            ...
            }
            "DBInstanceStatus": "upgrade-failed"
        }
    ]
}
```

After an upgrade failure, all database actions are blocked except for modifying the DB instance to perform the following tasks:

- Retrying the same upgrade
- Pausing and resuming RDS Custom automation
- Point-in-time recovery (PITR)

- Deleting the DB instance

> **Note**
>
> If automation has been paused for the RDS Custom DB instance, you can't retry the upgrade until you resume automation.
> The same actions apply to an upgrade failure for an RDS-managed read replica as for the primary.

For more information, see Troubleshooting upgrades for RDS Custom for Oracle.

Troubleshooting DB issues for Amazon RDS Custom for Oracle

The shared responsibility model of RDS Custom provides OS shell–level access and database administrator access. RDS Custom runs resources in your account, unlike Amazon RDS, which runs resources in a system account. With greater access comes greater responsibility. In the following sections, you can learn how to troubleshoot issues with Amazon RDS Custom DB instances.

> **Note**
>
> This section explains how to troubleshoot RDS Custom for Oracle. For troubleshooting RDS Custom for SQL Server, see Troubleshooting DB issues for Amazon RDS Custom for SQL Server.

Topics

- Viewing RDS Custom events
- Subscribing to RDS Custom events
- Troubleshooting DB instance creation issues
- Troubleshooting custom engine version creation for RDS Custom for Oracle
- Fixing unsupported configurations in RDS Custom for Oracle
- Troubleshooting upgrades for RDS Custom for Oracle
- Troubleshooting replica promotion for RDS Custom for Oracle

Viewing RDS Custom events

The procedure for viewing events is the same for RDS Custom and Amazon RDS DB instances. For more information, see Viewing Amazon RDS events.

To view RDS Custom event notification using the AWS CLI, use the `describe-events` command. RDS Custom introduces several new events. The event categories are the same as for Amazon RDS. For the list of events, see Amazon RDS event categories and event messages.

The following example retrieves details for the events that have occurred for the specified RDS Custom DB instance.

```
aws rds describe-events \
```

```
    --source-identifier my-custom-instance \
    --source-type db-instance
```

Subscribing to RDS Custom events

The procedure for subscribing to events is the same for RDS Custom and Amazon RDS DB instances. For more information, see Subscribing to Amazon RDS event notification.

To subscribe to RDS Custom event notification using the CLI, use the `create-event-subscription` command. Include the following required parameters:

- `--subscription-name`
- `--sns-topic-arn`

The following example creates a subscription for backup and recovery events for an RDS Custom DB instance in the current AWS account. Notifications are sent to an Amazon Simple Notification Service (Amazon SNS) topic, specified by `--sns-topic-arn`.

```
aws rds create-event-subscription \
    --subscription-name my-instance-events \
    --source-type db-instance \
    --event-categories '["backup","recovery"]' \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:interesting-events
```

Troubleshooting DB instance creation issues

If your environment isn't properly configured or required permissions are missing, you can't create or restore RDS Custom for Oracle DB instances. When you attempt to create or restore a DB instance, Amazon RDS validates your environment and returns specific error messages if it detects any issues.

After you resolve all issues, try again to create or restore your RDS Custom for Oracle DB instance.

Common permissions issues

When you create or restore a RDS Custom for Oracle instance, Amazon RDS validates that your environment has the required permissions. If permissions are missing or denied, the operation fails with a specific error message.

Issue type	Error message	Action
IAM role access policy	You can't create the DB instance because of incompatible resources . The host environment validation failed for the following permissions: <permission> on resource: <resource > due to permissions issue with message: User: <user> is not authorize d to perform: <permission> on resource: <resource> because no identity-based policy allows the <permission> action.	Ensure that the listed required permissions are present and set to `Allow` in the access policy with the appropriate resources included.
Permission boundary	You can't create the DB instance because of incompatible resources . The host environment validation failed for the following permissions: <permission> on resource: <resource > due to permissions issue with message: User: <user> is not authorize d to perform: <permission> on resource: <resource> with an explicit deny in a permissions boundary.	Verify that the permissions boundary attached to the instance role isn't restricting the listed required permissions and resources.
Service control policy	You can't create the DB instance because of incompatible resources . The host environment validation failed for the following permissions: <permission> on resource: <resource > due to permissions issue with message: User: <user> is not authorize d to perform: <permission> on resource: <resource> with an explicit deny in a service control policy.	Contact your AWS Organizations administrator and verify that the service control policy attached to your account isn't restricting the listed required permissions and resources.

Issue type	Error message	Action
Resource control policy	You can't create the DB instance because of incompatible resources . The host environment validation failed for the following permissions: <permission> on resource: <resource > due to permissions issue with message: User: <user> is not authorize d to perform: <permission> on resource: <resource> with an explicit deny in a resource control policy.	Contact your AWS Organizations administrator and verify that the resource control policy attached to your account isn't restricting the listed required permissions and resources.
VPC endpoint policy	You can't create the DB instance because of incompatible resources . The host environment validation failed for the following permissions: <permission> on resource: <resource > due to permissions issue with message: User: <user> is not authorize d to perform: <permission> on resource: <resource> with an explicit deny in a VPC endpoint policy.	Ensure that the required VPC endpoints exist and the policies attached to them aren't restricting the listed required permissions and resources.

Networking issues

In addition to reviewing Step 6: Configure your VPC for RDS Custom for Oracle, verify that the following are configured correctly and not restricting access to the required AWS services:

Security group attached to the Amazon EC2 instance

Ensure that the security group allows all necessary inbound and outbound traffic for RDS Custom operations.

Security group attached to your VPC

Verify that VPC security groups permit traffic to and from the required AWS services.

VPC endpoints

Confirm that all required VPC endpoints are properly configured and accessible.

Networking access control lists

Check that network ACLs aren't blocking traffic needed for RDS Custom functionality.

Troubleshooting custom engine version creation for RDS Custom for Oracle

When CEV creation fails, RDS Custom issues `RDS-EVENT-0198` with the message `Creation failed for custom engine version` *`major-engine-version.cev_name`*, and includes details about the failure. For example, the event prints missing files.

CEV creation might fail because of the following issues:

- The Amazon S3 bucket containing your installation files isn't in the same AWS Region as your CEV.
- When you request CEV creation in an AWS Region for the first time, RDS Custom creates an S3 bucket for storing RDS Custom resources (such as CEV artifacts, AWS CloudTrail logs, and transaction logs).

  CEV creation fails if RDS Custom can't create the S3 bucket. Either the caller doesn't have S3 permissions as described in Step 5: Grant required permissions to your IAM user or role, or the number of S3 buckets has reached the limit.
- The caller doesn't have permissions to get files from your S3 bucket that contains the installation media files. These permissions are described in Step 7: Add necessary IAM permissions.
- Your IAM policy has an `aws:SourceIp` condition. Make sure to follow the recommendations in AWS Denies access to AWS based on the source IP in the *AWS Identity and Access Management User Guide*. Also make sure that the caller has the S3 permissions described in Step 5: Grant required permissions to your IAM user or role.
- Installation media files listed in the CEV manifest aren't in your S3 bucket.
- The SHA-256 checksums of the installation files are unknown to RDS Custom.

  Confirm that the SHA-256 checksums of the provided files match the SHA-256 checksum on the Oracle website. If the checksums match, contact AWS Support and provide the failed CEV name, file name, and checksum.

- The OPatch version is incompatible with your patch files. You might get the following message: `OPatch is lower than minimum required version. Check that the version meets the requirements for all patches, and try again`. To apply an Oracle patch, you must use a compatible version of the OPatch utility. You can find the required version of the Opatch utility in the readme file for the patch. Download the most recent OPatch utility from My Oracle Support, and try creating your CEV again.
- The patches specified in the CEV manifest are in the wrong order.

You can view RDS events either on the RDS console (in the navigation pane, choose **Events**) or by using the `describe-events` AWS CLI command. The default duration is 60 minutes. If no events are returned, specify a longer duration, as shown in the following example.

```
aws rds describe-events --duration 360
```

Currently, the MediaImport service that imports files from Amazon S3 to create CEVs isn't integrated with AWS CloudTrail. Therefore, if you turn on data logging for Amazon RDS in CloudTrail, calls to the MediaImport service such as the `CreateCustomDbEngineVersion` event aren't logged.

However, you might see calls from the API gateway that accesses your Amazon S3 bucket. These calls come from the MediaImport service for the `CreateCustomDbEngineVersion` event.

Fixing unsupported configurations in RDS Custom for Oracle

In the shared responsibility model, it's your responsibility to fix configuration issues that put your RDS Custom for Oracle DB instance into the `unsupported-configuration` state. If the issue is with the AWS infrastructure, use the console or the AWS CLI to fix it. If the issue is with the operating system or the database configuration, log in to the host to fix it.

> **Note**
>
> This section explains how to fix unsupported configurations in RDS Custom for Oracle. For information about RDS Custom for SQL Server, see Fixing unsupported configurations in RDS Custom for SQL Server.

The following tables includes descriptions of the notifications and events that the support perimeter sends and how to fix them. These notifications and the support perimeter are subject to

change. For background on the support perimeter, see RDS Custom support perimeter. For event descriptions, see Amazon RDS event categories and event messages.

Event ID	Configuration	RDS event message	Action
SP-00000	Manual unsupported configuration	The RDS Custom DB instance status is set to [Unsupported configuration] because of: *reason*.	To resolve this issue, create an Support case.

AWS resources (infrastructure)

Event ID	Configuration	RDS event message	Action
SP-O100	Amazon Elastic Block Store (Amazon EBS) volumes	The following EBS volumes were added to EC2 instance *ec2_id*: *volume_id* . To resolve the issue, detach the specified volumes from the instance.	RDS Custom creates two types of EBS volume, besides the root volume created from the Amazon Machine Image (AMI), and associates them with the EC2 instance: • The binary volume where the database software binaries are located • The data volumes where database files are located When you create your DB instance, the storage configurations that you specify configure the data volumes. The support perimeter monitors the following: • The initial EBS volumes created with the DB instance are still associated with the instance.

Event ID	Configuration	RDS event message	Action

			• The initial EBS volumes still have the same configurations as initially set: storage type, size, Provisioned IOPS, and storage throughput.
			• No additional EBS volumes are attached to the DB instance.
			
			Use the following CLI command to compare the volume type of the EBS volume details and the RDS Custom for Oracle DB instance details:
			
			`aws rds describe-db-instances \`
			`--db-instance-identifier db-instance-name | grep StorageType`

Event ID	Configuration	RDS event message	Action
SP-O100	Amazon Elastic Block Store (Amazon EBS) volumes	EBS volume *volume_id* has been detached from EC2 instance [*ec2_id*]. You can't detach the original volume from this instance. To resolve the issue, re-attach *volume_id* to *ec2_id*.	RDS Custom creates two types of EBS volume, besides the root volume created from the Amazon Machine Image (AMI), and associates them with the EC2 instance: • The binary volume where the database software binaries are located • The data volumes where database files are located When you create your DB instance, the storage configurations that you specify configure the data volumes. The support perimeter monitors the following: • The initial EBS volumes created with the DB instance are still associated with the instance. • The initial EBS volumes still have the same configurations as initially set: storage type, size, Provisioned IOPS, and storage throughput. • No additional EBS volumes are attached to the DB instance. Use the following CLI command to compare the volume type of the EBS volume details and the RDS Custom for Oracle DB instance details: `aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep StorageType`

Event ID	Configuration	RDS event message	Action
SP-O100	Amazon Elastic Block Store (Amazon EBS) volumes	The original EBS volume *volume_id* attached to EC2 instance *ec2_id* has been modified as follows: size [*X*] to [*Y*], type [*N*] to [*M*], or IOPS [*J*] to [*K*]. To resolve the issue, revert the modification.	RDS Custom creates two types of EBS volume, besides the root volume created from the Amazon Machine Image (AMI), and associates them with the EC2 instance: • The binary volume where the database software binaries are located • The data volumes where database files are located When you create your DB instance, the storage configurations that you specify configure the data volumes. The support perimeter monitors the following: • The initial EBS volumes created with the DB instance are still associated with the instance. • The initial EBS volumes still have the same configurations as initially set: storage type, size, Provisioned IOPS, and storage throughput. • No additional EBS volumes are attached to the DB instance. Use the following CLI command to compare the volume type of the EBS volume details and the RDS Custom for Oracle DB instance details: `aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep StorageType`

Event ID	Configuration	RDS event message	Action
SP-O1004	Amazon EC2 instance state	Automated recovery left EC2 instance [*ec2_id*] in an impaired state. To resolve the issue, see Troubleshooting instance recovery failures.	To check the status of a DB instance, use the console or run the following AWS CLI command: `aws rds describe-db-instances \ --db-instance-identifier *db-instance-name* |grep DBInstanceStatus`
SP-O1005	Amazon EC2 instance attributes	EC2 instance [*ec2_id*] was modified as follows: attribute [*att1*] changed from [*val-old*] to [*val-new*], attribute [*att2*] changed from [*val-old*] to [*val-new*]. To resolve the issue, revert to the original value.	

Event ID	Configuration	RDS event message	Action
SP-O1006	Amazon EC2 instance state	EC2 instance [*ec2_id*] was terminated or can't be found. To resolve the issue, delete the RDS Custom DB instance.	The support perimeter monitors EC2 instance state-change notifications. The EC2 instance must always be running. **To delete your DB instance** 1. To check the status of a DB instance, use the console or run the following AWS CLI command: `aws rds describe-db-instances \ --db-instance-identifier *db-instance-name* |grep DBInstanceStatus` 2. Delete your RDS Custom for Oracle DB instance.
SP-O1007	Amazon EC2 instance state	EC2 instance [*ec2_id*] was stopped. To resolve the issue, start the instance.	The support perimeter monitors EC2 instance state-change notifications. The EC2 instance must always be running. **To restart your DB instance** 1. To check the status of a DB instance, use the console or run the following AWS CLI command: `aws rds describe-db-instances \ --db-instance-identifier *db-instance-name* |grep DBInstanceStatus` 2. Start your DB instance. 3. Remount the binary and data volumes.

Event ID	Configuration	RDS event message	Action

SP-10	Amazon SQS permission	Permissions are missing for Amazon SQS. Check the permissions for the IAM instance profile, VPC endpoint policy, and dependent service connections, and then try again.	You can resolve this by making sure the IAM profile associated with the host has the following permissions:
			`"SQS:SendMessage"`
			`"SQS:ReceiveMessage"`
			`"SQS:DeleteMessage"`
			`"SQS:GetQueueUrl"`

SP-10	Amazon Simple Queue Service (Amazon SQS)	The SQS queue [%s] was deleted and couldn't be recovered. To resolve this issue, recreate the queue.	Recreate the Amazon SQS queue.

Operating system

Event ID	Configuration	RDS event message	Action
SP-O200	RDS Custom agent status	The RDS Custom agent isn't running on EC2 instance [*ec2_id*]. Make sure the agent is running on [*ec2_id*].	On RDS Custom for Oracle, the DB instance goes outside the support perimeter if the RDS Custom agent stops. The agent publishes the `IamAlive` metric to Amazon CloudWatch every 30 seconds. An alarm is triggered if the metric hasn't been published for 30 seconds. The support perimeter also monitors the RDS Custom agent process state on the host every 30 minutes.

Event ID	Configuration	RDS event message	Action

			To restart the RDS Custom agent
			
			1. Log in to your host and make sure that the RDS Custom agent is running.
			2. Run the following command to find the status of the agent.
			
			`service rdscustomagent status`
			
			3. Use the following command to start the agent.
			
			`service rdscustomagent start`
			
			When the RDS Custom agent is running again, the `IamAlive` metric is published to Amazon CloudWatch, and the alarm switches to the OK state. This switch notifies the support perimeter that the agent is running.

Event ID	Configuration	RDS event message	Action
SP-02002	AWS Systems Manager agent (SSM agent) status	The Systems Manager agent on EC2 instance [*ec2_id*] is unreachable. Make sure that you that have correctly configured the network, agent, and IAM permissions.	SSM Agent must always be running. The RDS Custom agent is responsible for making sure that the Systems Manager agent is running. If SSM Agent was terminated and then restarted, the RDS Custom agent publishes the metric `SSM_Agent_Restarted_Or_NotFound` to CloudWatch. The RDS Custom agent has an alarm on the metric `do-not-delete-rds-custom-ssm-agent-restarted-or-notfound-` *ec2-id* configured to trigger when there has been a restart in each of the previous three minutes. The support perimeter also monitors the process state of SSM Agent on the host every 30 minutes. For more information, see Troubleshooting SSM Agent.
SP-02003	AWS Systems Manager agent (SSM agent) status	The Systems Manager agent on EC2 instance [*ec2_id*] crashed multiple times. For more information, see the SSM Agent troubleshooting documentation.	For more information, see Troubleshooting SSM Agent.

Event ID	Configuration	RDS event message	Action
SP-O200	OS time zone	The time zone on EC2 instance [*ec2_id*] was changed. To resolve this issue, revert the timezone to the previous setting of [*previous-time-zone*]. Then use an RDS options group to change the time zone.	RDS automation detected that the time zone on the host was changed without the use of an option group. This host-level change can cause RDS automation failures, so the EC2 instance is placed in the `unsupported-configuration` state. **To fix the time zone setting** 1. Log in to your EC2 host and check the OS time zone as follows: `timedatectl` 2. Pause RDS Custom automation. For more information, see Pausing and resuming your RDS Custom DB instance. 3. Stop the DB instance. 4. Revert the time zone change on the operating system. 5. Start the DB instance. 6. Resume RDS Custom automation. Your DB instance becomes available within 30 minutes. To prevent moving out of perimeter in the future, modify your timezone through an options group. For more information, see Oracle time zone.

Event ID	Configuration	RDS event message	Action
SP-O200	sudo configurations	The sudo configurations on EC2 instance [*ec2_id*] lack necessary permissions. To resolve this issue, revert the recent changes to the sudo configurations.	The support perimeter verifies that certain OS users are allowed to run certain commands on the host. It monitors sudo configurations and compares them to the supported state. If the sudo configurations aren't supported, RDS Custom tries to overwrite them and return to the previous supported state. If the attempt is successful, RDS Custom sends the following notification: RDS Custom successfully overwrote your configuration. If the overwrite isn't successful, your DB instance remains in the unsupported configuration state. To resolve this problem, either revert the changes within the `sudoers.d/` file or fix the permissions. **To investigate changes to the sudo configurations** 1. Log in to your host. 2. Run the following command. `visudo -c -f /etc/sudoers.d/ individual_sudo_files` 3. Modify the sudo configurations as necessary. After the support perimeter determines that the sudo configurations are supported, your RDS Custom for Oracle DB instance becomes available within 30 minutes.

Event ID	Configuration	RDS event message	Action
SP-O2006	S3 bucket accessibility	RDS Custom automation can't download files from the S3 bucket on EC2 instance [*ec2_id*]. Check your networking configuration and make sure the instance allows connections to and from S3.	
SP-20	High Availability Software Solution Version	The HA solution of your instance differs from the expected version. To resolve this issue, create an AWS Support case.	Create an AWS Support case.

Database

Event ID	Configuration	RDS event message	Action
SP-O300	Database archive lag target	The ARCHIVE_LAG_TARGET parameter on EC2 instance [*ec2_id*] is out of the recommended range *value_ran*	The support perimeter monitors the `ARCHIVE_LAG_TARGET` database parameter to verify that the latest restorable time of the DB instance is within reasonable bounds. **To change the lag target for archived redo logs** 1. Log in to your EC2 host

Event ID	Configuration	RDS event message	Action
		ge . To resolve the issue, set the parameter to a value within value_range.	2. Connect to your RDS Custom for Oracle DB instance 3. Change the ARCHIVE_LAG_TARGET parameter to a value from 60–7200. For example, use the following SQL statement. `ALTER SYSTEM SET ARCHIVE_LAG_TARGET=300 SCOPE=BOTH;` Your DB instance becomes available within 30 minutes.

Event ID	Configuration	RDS event message	Action

SP-O300	Oracle Data Guard role	The database role [*role_name*] isn't supported for Oracle Data Guard on EC2 instance [*ec2_id*]. To resolve the issue, set the DATABASE_ROLE parameter to either PRIMARY or PHYSICAL STANDBY.	The support perimeter monitors the current database role every 15 seconds and sends a CloudWatch notification if the database role has changed. The Oracle Data Guard DATABASE_ROLE parameter must be either PRIMARY or PHYSICAL STANDBY.
			
			To restore your Oracle Data Guard database role to a supported value
			
			1. Check the Oracle Data Guard role by running the following statement:
			
			`SELECT DATABASE_ROLE FROM V$DATABASE;`
			
			2. If your DB instance is standalone, use either of the following statements to change it back to the PRIMARY role:
			
			`ALTER DATABASE COMMIT TO SWITCHOVER PRIMARY;`
			`ALTER DATABASE ACTIVATE STANDBY DATABASE;`
			
			If your DB instance is a replica, use the following statement to change it back to the PHYSICAL STANDBY role:
			
			`ALTER DATABASE CONVERT TO PHYSICAL STANDBY;`
			
			After the support perimeter determines that the database role is supported, your RDS Custom for Oracle DB instance becomes available within 15 seconds.

Event ID	Configuration	RDS event message	Action
SP-O300	Database health	The SMON process of the Oracle database is in a zombie state. To resolve the issue, manually recover the database on EC2 instance [*ec2_id*], open the database, and then immediately back it up. For more help, contact Support.	The support perimeter monitors the DB instance state. It also monitors how many restarts occurred during the previous hour and day. You're notified when the instance is in a state where it still exists, but you can't interact with it. **To make the support perimeter evaluate your instance state** 1. Log in to your host and determine the database state. `ps -eo pid,state,command | grep smon` 2. If necessary, restart your DB instance. If the restart fails, proceed to the next step. 3. If necessary, restart your EC2 host. After your DB instance restarts, the RDS Custom agent detects that your DB instance is no longer in an unresponsive state. It then notifies the support perimeter to reevaluate your DB instance state.

Event ID	Configuration	RDS event message	Action

SP-O3004	Database log mode	The database log mode on EC2 instance [*ec2_id*] was changed to [*value_b*]. To resolve the issue, set the log mode to [*value_a*].	**To change your DB instance log mode to ARCHIVELOG**
			1. Log in to your EC2 host.
			2. Connect to your database and run the following statement:
			`SELECT LOG_MODE FROM V$DATABASE;`
			Or you can run the follow command in SQL*Plus:
			`ARCHIVE LOG LIST`
			3. Run the following SQL*Plus command to initiate a consistent shutdown.
			`SHUTDOWN IMMEDIATE`
			The RDS Custom agent automatically restarts your DB instance and sets the log mode to ARCHIVELOG . Your DB instance becomes available within 30 minutes.

SP-O3005	Oracle home path	The Oracle home on EC2 instance [*ec2_id*] was changed to *new_path*. To resolve the issue, revert the setting to *old_path*.	

Event ID	Configuration	RDS event message	Action

SP-O3006	Database unique name	The database unique name on EC2 instance [*ec2_id*] was changed to *new_value* . To resolve the issue, revert the name to *old_value* .	**To change the database unique name for your DB instance**
			1. Log in to your EC2 host.
			2. Connect to the database and run the following statement:
			`SELECT DB_UNIQUE_NAME FROM V$DATABASE;`
			3. Specify the original database unique name using the command `ALTER SYSTEM SET DB_UNIQUE_NAME` .
			4. Run the following SQL statement to initiate a consistent shutdown.
			`SHUTDOWN IMMEDIATE;`
			The RDS Custom agent automatically restarts your DB instance and sets the log mode to `ARCHIVELOG` . Your DB instance becomes available within 30 minutes.

Troubleshooting upgrades for RDS Custom for Oracle

Your upgrade of an RDS Custom for Oracle instance might fail. Following, you can find techniques that you can use during upgrades of RDS Custom DB for Oracle DB instances:

- Examine the upgrade output log files in the `/tmp` directory on your DB instance. The names of the logs depend on your DB engine version. For example, you might see logs that contain the strings `catupgrd` or `catup`.
- Examine the `alert.log` file located in the `/rdsdbdata/log/trace` directory.

- Run the following `grep` command in the `root` directory to track the upgrade OS process. This command shows where the log files are being written and determine the state of the upgrade process.

```
ps -aux | grep upg
```

The following shows sample output.

```
root     18884  0.0  0.0 235428  8172 ?        S<   17:03   0:00 /usr/bin/
sudo -u rdsdb /rdsdbbin/scripts/oracle-control ORCL op_apply_upgrade_sh RDS-
UPGRADE/2.upgrade.sh
rdsdb    18886  0.0  0.0 153968 12164 ?        S<   17:03   0:00 /usr/bin/perl -T -
w /rdsdbbin/scripts/oracle-control ORCL op_apply_upgrade_sh RDS-UPGRADE/2.upgrade.sh
rdsdb    18887  0.0  0.0 113196  3032 ?        S<   17:03   0:00 /bin/sh /rdsdbbin/
oracle/rdbms/admin/RDS-UPGRADE/2.upgrade.sh
rdsdb    18900  0.0  0.0 113196  1812 ?        S<   17:03   0:00 /bin/sh /rdsdbbin/
oracle/rdbms/admin/RDS-UPGRADE/2.upgrade.sh
rdsdb    18901  0.1  0.0 167652 20620 ?        S<   17:03   0:07 /rdsdbbin/oracle/
perl/bin/perl catctl.pl -n 4 -d /rdsdbbin/oracle/rdbms/admin -l /tmp catupgrd.sql
root     29944  0.0  0.0 112724  2316 pts/0    S+   18:43   0:00 grep --color=auto
 upg
```

- Run the following SQL query to verify the current state of the components to find the database version and the options installed on the DB instance.

```
SET LINESIZE 180
COLUMN COMP_ID FORMAT A15
COLUMN COMP_NAME FORMAT A40 TRUNC
COLUMN STATUS FORMAT A15 TRUNC
SELECT COMP_ID, COMP_NAME, VERSION, STATUS FROM DBA_REGISTRY ORDER BY 1;
```

The output resembles the following.

```
COMP_NAME                                STATUS               PROCEDURE
---------------------------------------- --------------------
 ------------------------------------------------
Oracle Database Catalog Views            VALID
 DBMS_REGISTRY_SYS.VALIDATE_CATALOG
Oracle Database Packages and Types       VALID
 DBMS_REGISTRY_SYS.VALIDATE_CATPROC
Oracle Text                              VALID                VALIDATE_CONTEXT
```

```
Oracle XML Database                        VALID                DBMS_REGXDB.VALIDATEXDB

4 rows selected.
```

- Run the following SQL query to check for invalid objects that might interfere with the upgrade process.

```
SET PAGES 1000 LINES 2000
COL OBJECT FOR A40
SELECT SUBSTR(OWNER,1,12) OWNER,
       SUBSTR(OBJECT_NAME,1,30) OBJECT,
       SUBSTR(OBJECT_TYPE,1,30) TYPE, STATUS,
       CREATED
FROM   DBA_OBJECTS
WHERE  STATUS <>'VALID'
AND    OWNER IN ('SYS','SYSTEM','RDSADMIN','XDB');
```

Troubleshooting replica promotion for RDS Custom for Oracle

You can promote managed Oracle replicas in RDS Custom for Oracle using the console, `promote-read-replica` AWS CLI command, or `PromoteReadReplica` API. If you delete your primary DB instance, and all replicas are healthy, RDS Custom for Oracle promotes your managed replicas to standalone instances automatically. If a replica has paused automation or is outside the support perimeter, you must fix the replica before RDS Custom can promote it automatically. For more information, see Promoting an RDS Custom for Oracle replica to a standalone DB instance.

The replica promotion workflow might become stuck in the following situation:

- The primary DB instance is in the state `STORAGE_FULL`.
- The primary database can't archive all of its online redo logs.
- A gap exists between the archived redo log files on your Oracle replica and the primary database.

To respond to the stuck workflow

1. Synchronize the redo log gap on your Oracle replica DB instance.
2. Force the promotion of your read replica to the latest applied redo log. Run the following commands in SQL*Plus:

```
ALTER DATABASE ACTIVATE STANDBY DATABASE;
```

```
SHUTDOWN IMMEDIATE
STARTUP
```

3. Contact Support and ask them to move your DB instance to `available` status.

Known issues for Amazon RDS Custom for Oracle

When working with RDS Custom for Oracle, note the following issues for DB instances:

- Resizing the root or dbbin volumes isn't supported.

  > **Warning**
  >
  > We strongly recommend that you don't resize the root or dbbin volumes manually. We recommend that you store all configurations in the data volume, which persists after patching, and that you resize the volume using only the RDS scale storage API.

- Some RDS APIs can be blocked when a database instance is on an older AMI, for example, an AMI that uses Oracle Linux 7. To resolve this issue, patch your DB instance to the latest AMI using OS patching. For more information, see CEV upgrade options.
- Before you perform RDS operations, make sure that your AWS account has enough quota for compute and storage.
- If the database is in the creation state, and you actively log in to the database or Amazon EC2 host and run commands, database creation might not complete.
- Control file multiplexing isn't currently supported because of a read replica issue. Before you create a read replica, make sure to specify only one file name in the `CONTROL_FILES` initialization parameter on the source database.
- You can't change the database mode from `PHYSICAL STANDBY` (mounted or read-only) to `SNAPSHOT STANDBY` (converting to read/write).
- If an AWS account is part of an AWS Organization with a service control policy (SCP), and the SCP contains a condition key, an RDS Custom for Oracle DB instance might fail to create with the following error:

```
You can't create the DB instance because of incompatible resources.
The IAM instance profile role [AWSRDSCustomInstanceRole1-us-east-1] is missing the
 following permissions:
EFFECT [Allow] on ACTION(S) [ssm:DescribeAssociation, ssm:DescribeDocument,
 ssm:GetConnectionStatus,
  ssm:GetDeployablePatchSnapshotForInstance, ssmmessages:OpenControlChannel,
 ssm:GetParameters,
  ssm:ListInstanceAssociations, ssm:PutConfigurePackageResult,
 ssmmessages:CreateControlChannel,
```

```
 ssm:GetParameter, ssm:UpdateAssociationStatus, ssm:GetManifest,
ssmmessages:CreateDataChannel,
 ssm:PutInventory, ssm:UpdateInstanceInformation, ssm:DescribeInstanceInformation,
 ssmmessages:OpenDataChannel, ssm:GetDocument, ssm:ListAssociations,
ssm:PutComplianceItems,
 ssm:UpdateInstanceAssociationStatus] for RESOURCE(S) [], EFFECT [Allow] on
ACTION(S) [ec2messages:DeleteMessage,
 ec2messages:FailMessage, ec2messages:GetEndpoint, ec2messages:AcknowledgeMessage,
ec2messages:GetMessages,
 ec2messages:SendReply] for RESOURCE(S) [], EFFECT [Allow] on ACTION(S)
[logs:CreateLogStream,
 logs:DescribeLogStreams, logs:PutRetentionPolicy, logs:PutLogEvents]
```

To resolve this issue, create a ticket with Support.

Known issues with database user accounts

Note the following issues:

- Don't remove database user accounts that begin with the string RDS, such as RDSADMIN and RDS_DATAGUARD. RDS Custom for Oracle uses the RDS account for automation. If you remove this user account, RDS Custom moves the instance to the unsupported configuration state.
- You can't change the master username for your RDS Custom for Oracle DB instance using the ModifyDBInstance API.
- RDS Custom for Oracle rotates user account credentials on all DB instances. For more information, see Rotating RDS Custom for Oracle credentials for compliance programs. If you use an on-premises primary/standby configuration, credential rotation might affect the following resources:
  - Manually created standby RDS Custom for Oracle instances

    To resolve this issue, drop the manual standby databases, and then create an Oracle read replica using an API call. Manage the secrets manually for the manual standby databases so that they match the source DB instance.
  - Manually created cross-Region read replicas

    To resolve this issue, manually keep the secrets so that they match the primary DB instance.

Known issues with parameter and configuration files

- You must configure the `crontab` file after scale compute, OS upgrades, and other operations where RDS Custom replaces the root volume. We highly recommend that you keep a backup of `crontab`.
- Note the following guidelines when you configure the `listener.ora` file:
  - Make sure that every entry in the file is on a single line. This approach avoids issues with indentation during instance creation.
  - Make sure that `GLOBAL_DBNAME` is equal to the value of `SID_NAME`.
  - Make sure the value for `LISTENER` follows the naming convention `L_`*`dbname`*`_001`.
  - Make sure the `listener.ora` file maintains a connection to the database name. RDS Custom uses this connection to verify database startup. If you modify this file incorrectly, operations such as scale compute or patching might fail.

    The following example shows a `listener.ora` that is configured correctly:

    ```
    ADR_BASE_L_ORCL_001=/rdsdbdata/log/
    USE_SID_AS_SERVICE_L_ORCL_001=ON
    SID_LIST_L_ORCL_001=(SID_LIST = (SID_DESC = (SID_NAME = ORCL)(GLOBAL_DBNAME = ORCL)
     (ORACLE_HOME = /rdsdbbin/oracle.19.custom.r1.EE.1)))
    SUBSCRIBE_FOR_NODE_DOWN_EVENT_L_ORCL_001=OFF
    L_ORCL_001=(DESCRIPTION_LIST = (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(PORT =
     XXXX)(HOST = x.x.x.x))) (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(PORT = XXXX)
    (HOST = 127.0.0.1))))
    ```
- Comments aren't supported in a server parameter file or initialization parameter file.
- You must declare the following initialization parameters in the server parameter file (`/rdsdbdata/config/oracle_pfile`):
  - `MEMORY_MAX_TARGET`
  - `MEMORY_TARGET`
  - `PGA_AGGREGATE_TARGET`
  - `PROCESSES`
  - `SGA_TARGET`
  - `USE_LARGE_PAGES`

If the preceding parameters aren't declared in `/rdsdbdata/config/oracle_pfile`, read replica creation and scale compute might fail.

- You can't delete the symbolic links for configuration files such as the server parameter file, audit files, `listener.ora`, `tnsnames.ora`, or `sqlnet.ora`. You also can't modify the directory structure for these files. RDS Custom automation expects these files to exist in a specific directory structure.

  To create a server parameter file from an initialization parameter file, use the following syntax.

```
CREATE SPFILE='/rdsdbdata/admin/$ORACLE_SID/pfile/spfile$ORACLE_SID.ora'
    FROM PFILE='/rdsdbdata/config/oracle_pfile';
```

Working with RDS Custom for SQL Server

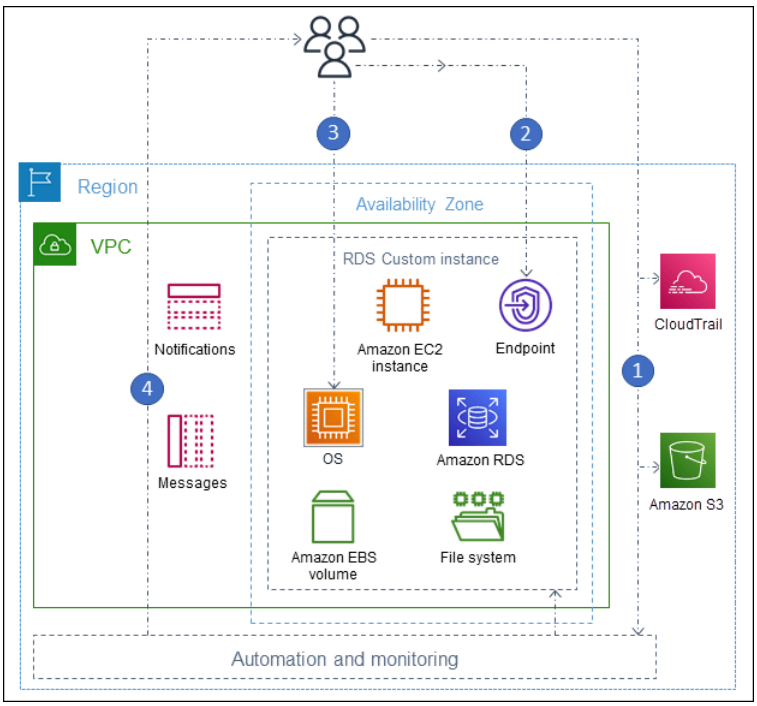
Following, you can find instructions for creating, managing, and maintaining your RDS Custom for SQL Server DB instances.

Topics

- RDS Custom for SQL Server workflow
- Requirements and limitations for Amazon RDS Custom for SQL Server
- Setting up your environment for Amazon RDS Custom for SQL Server
- Bring Your Own Media with RDS Custom for SQL Server
- Working with custom engine versions for RDS Custom for SQL Server
- Creating and connecting to a DB instance for Amazon RDS Custom for SQL Server
- Managing an Amazon RDS Custom for SQL Server DB instance
- Working with Microsoft Active Directory with RDS Custom for SQL Server
- Managing a Multi-AZ deployment for RDS Custom for SQL Server
- Backing up and restoring an Amazon RDS Custom for SQL Server DB instance
- Copying an Amazon RDS Custom for SQL Server DB snapshot
- Migrating an on-premises database to Amazon RDS Custom for SQL Server
- RDS Custom for SQL Server Operating system updates
- Upgrading an Amazon RDS Custom for SQL Server DB instance
- Troubleshooting DB issues for Amazon RDS Custom for SQL Server

RDS Custom for SQL Server workflow

The following diagram shows the typical workflow for RDS Custom for SQL Server.



The steps are as follows:

1. Create an RDS Custom for SQL Server DB instance from an engine version offered by RDS Custom.

   For more information, see Creating an RDS Custom for SQL Server DB instance.

2. Connect your application to the RDS Custom DB instance endpoint.

   For more information, see Connecting to your RDS Custom DB instance using AWS Systems Manager and Connecting to your RDS Custom DB instance using RDP.

3. (Optional) Access the host to customize your software.

4. Monitor notifications and messages generated by RDS Custom automation.

Creating a DB instance for RDS Custom

You create your RDS Custom DB instance using the `create-db-instance` command. The procedure is similar to creating an Amazon RDS instance. However, some of the parameters are different. For more information, see Creating and connecting to a DB instance for Amazon RDS Custom for SQL Server.

Database connection

Like an Amazon RDS DB instance, your RDS Custom for SQL Server DB instance resides in a VPC. Your application connects to the RDS Custom instance using a client such as SQL Server Management Suite (SSMS), just as in RDS for SQL Server.

RDS Custom customization

You can access the RDS Custom host to install or customize software. To avoid conflicts between your changes and the RDS Custom automation, you can pause the automation for a specified period. During this period, RDS Custom doesn't perform monitoring or instance recovery. At the end of the period, RDS Custom resumes full automation. For more information, see Pausing and resuming RDS Custom automation.

Requirements and limitations for Amazon RDS Custom for SQL Server

Following, you can find a summary of the Amazon RDS Custom for SQL Server requirements and limitations for quick reference. Requirements and limitations also appear in the relevant sections.

Topics

- Region and version availability
- General requirements for RDS Custom for SQL Server
- DB instance class support for RDS Custom for SQL Server
- Limitations for RDS Custom for SQL Server
- Setting character sets and collations for RDS Custom for SQL Server DB instances
- Local time zone for RDS Custom for SQL Server DB instances
- Using a Service Master Key with RDS Custom for SQL Server
- Change data capture (CDC) support with RDS Custom for SQL Server

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability of Amazon RDS with Amazon RDS Custom for SQL Server, see Supported Regions and DB engines for RDS Custom for SQL Server.

General requirements for RDS Custom for SQL Server

Make sure to follow these requirements for Amazon RDS Custom for SQL Server:

- Use the instance classes shown in DB instance class support for RDS Custom for SQL Server. The only storage types supported are solid state drives (SSD) of types gp2, gp3, io1, and io2 Block Express. The maximum storage limit for io1, gp2, and gp3 is 16 TiB while io2 supports 64 TiB.
- Make sure that you have a symmetric encryption AWS KMS key to create an RDS Custom DB instance. For more information, see Make sure that you have a symmetric encryption AWS KMS key.
- Make sure that you create an AWS Identity and Access Management (IAM) role and instance profile. For more information, see Creating your IAM role and instance profile manually and Automated instance profile creation using the AWS Management Console.

- Make sure to supply a networking configuration that RDS Custom can use to access other AWS services. For specific requirements, see Step 2: Configure networking, instance profile, and encryption.
- The combined number of RDS Custom and Amazon RDS DB instances can't exceed your quota limit. For example, if your quota is 40 DB instances, you can have 20 RDS Custom for SQL Server DB instances and 20 Amazon RDS DB instances.
- RDS Custom automatically creates an AWS CloudTrail trail whose name begins with `do-not-delete-rds-custom-`. The RDS Custom support perimeter relies on the events from CloudTrail to determine whether your actions affect RDS Custom automation. RDS Custom creates the trail when you create your first DB instance. To use an already existing CloudTrail, contact AWS Support. For more information, see AWS CloudTrail.

DB instance class support for RDS Custom for SQL Server

Check if the DB instance class is supported in your Region by using the describe-orderable-db-instance-options command.

RDS Custom for SQL Server supports the DB instance classes shown in the following table:

SQL Server edition	RDS Custom support
Enterprise Edition	db.r5.xlarge–db.r5.24xlarge db.r5b.xlarge–db.r5b.24xlarge db.m5.xlarge–db.m5.24xlarge db.r6i.xlarge–db.r6i.32xlarge db.m6i.xlarge–db.m6i.32xlarge db.x2iedn.xlarge–db.x2iedn.32xlarge
Standard Edition	db.r5.large–db.r5.24xlarge db.r5b.large–db.r5b.8xlarge

SQL Server edition	RDS Custom support
	db.m5.large–db.m5.24xlarge db.r6i.large–db.r6i.8xlarge db.m6i.large–db.m6i.8xlarge db.x2iedn.xlarge–db.x2iedn.8xlarge
Developer Edition	db.r5.xlarge–db.r5.24xlarge db.r5b.xlarge–db.r5b.24xlarge db.m5.xlarge–db.m5.24xlarge db.r6i.xlarge–db.r6i.32xlarge db.m6i.xlarge–db.m6i.32xlarge db.x2iedn.xlarge–db.x2iedn.32xlarge
Web Edition	db.r5.large–db.r5.4xlarge db.m5.large–db.m5.4xlarge db.r6i.large–db.r6i.4xlarge db.m6i.large–db.m6i.4xlarge db.r5b.large–db.r5b.4xlarge

The following recommendations apply to db.x2iedn class types:

- At creation, local storage is a raw and unallocated device. Before using a DB instance with this instance class, you must mount and format the local storage. Afterward, configure `tempdb` on it to ensure optimal performance. For more information, see Optimize tempdb performance in Amazon RDS Custom for SQL Server using local instance storage.

- Local storage reverts to its raw and unallocated state when you run DB instance operations such as scale compute, instance replacement, snapshot restore, or point-in-time recovery (PITR). In these situations, you must remount, reformat, and reconfigure the drive and `tempdb` to restore functionality.
- For Multi-AZ instances, we recommend that you perform the configuration on a standby DB instance. This way, if a failover occurs, the system continues to operate without issues because the configuration is already in place on the standby instance.

Limitations for RDS Custom for SQL Server

The following limitations apply to RDS Custom for SQL Server:

- You can't create read replicas in Amazon RDS for RDS Custom for SQL Server DB instances. However, you can configure high availability automatically with a Multi-AZ deployment. For more information, see Managing a Multi-AZ deployment for RDS Custom for SQL Server.
- You can't modify the DB instance identifier of an existing RDS Custom for SQL Server DB instance.
- For an RDS Custom for SQL Server DB instance that wasn't created with a custom engine version (CEV), changes to the Microsoft Windows operating system aren't guaranteed to persist. For example, you lose these changes when you initiate a snapshot or point-in-time restore operation. If the RDS Custom for SQL Server DB instance was created with a CEV, then those changes are persisted.
- Not all options are supported. For example, when you create an RDS Custom for SQL Server DB instance, you can't do the following:
  - Change the number of CPU cores and threads per core on the DB instance class.
  - Turn on storage autoscaling.
  - Specify your own DB parameter group, option group, or character set.
  - Turn on Performance Insights or Database Insights.
  - Turn on automatic minor version upgrade.
- The maximum DB instance storage is 64 TiB.
- You can't use RDS Proxy with RDS Custom for SQL Server.
- You can't use the `describe-reserved-db-instances` API for RDS Custom for SQL Server DB instances.

Setting character sets and collations for RDS Custom for SQL Server DB instances

Overview

With RDS Custom for SQL Server DB instances, you can configure the character set and collation settings that determine how data is stored and sorted. Character sets define which characters are allowed, while collations specify the rules for sorting and comparing data. It't important to set the appropriate character sets and collations for applications that work with multilingual data or have specific sorting requirements. For example, you might need to handle accented characters and define language-specific sorting rules, or maintain data integrity across different locales. The following sections provide information on character set and collation support for your RDS Custom for SQL Server DB instances.

RDS Custom for SQL Server supports a wide range of server collations, both in traditional and UTF-8 encoding, for the SQL_Latin, Japanese, German, and Arabic locales. The default server collation is `SQL_Latin1_General_CP1_CI_AS`, however, you can select another supported collation to use. You can select a collation using the same procedure that RDS for SQL Server uses. For more information, see Managing collations and character sets for Amazon RDS for Microsoft SQL Server.

Considerations

The following requirements and limitations apply when working with server collations on RDS Custom for SQL Server:

- You can set the server collation when you create an RDS Custom for SQL Server DB instance. You can't modify the server-level collation after the DB instance is created.
- You can't modify the server level collation when restoring from a DB snapshot or during a point in time recovery (PITR).
- When you create a DB instance from an RDS Custom for SQL Server CEV, the DB instance doesn't inherit the server collation from the CEV. Instead, the default server collation of `SQL_Latin1_General_CP1_CI_AS` is used. If you've configured a non-default server collation on a RDS Custom for SQL Server CEV and want to use that same server collation on a new DB instance, be sure to select that same collation when you create the DB instance from the CEV.

> **Note**
>
> If the collation you select while creating the DB instance is different from the collation of the CEV, the Microsoft SQL Server system databases on the new RDS Custom for SQL

Server DB instance will be rebuilt to use the updated collation. The rebuild process is only performed on the new RDS Custom for SQL Server DB instance and has no impact on the CEV itself. Any previous modifications that you made to the system databases on the CEV will not be retained on the new RDS Custom for SQL Server DB instance once the system databases are rebuilt. Examples of some modifications include user-defined objects in the `master` database, scheduled jobs in the `msdb` database, or changes to default database settings in the `model` database on your CEV. You can manually recreate your modifications once the new RDS Custom for SQL Server DB instance is created.

- When you create a DB instance from an RDS Custom for SQL Server custom engine version (CEV) and select a different collation from that of the CEV, ensure that your golden image (AMI) used for CEV creation meets the following requirements so the Microsoft SQL Server system databases on the new DB instance can be rebuilt:
  - For SQL Server 2022, ensure the `setup.exe` file is located in the following path: `C:\Program Files\Microsoft SQL Server\160\Setup Bootstrap\SQL2022\setup.exe`
  - For SQL Server 2019, ensure the `setup.exe` file is located in the following path: `C:\Program Files\Microsoft SQL Server\150\Setup Bootstrap\SQL2019\setup.exe`
  - Copies of the data and log templates for the `master`, `model`, and `msdb` databases must exist in their default locations. For more information, see Rebuild system databases in the Microsoft public documentation.
  - Ensure your SQL Server Database Engine uses `NT Service\MSSQLSERVER` or `NT AUTHORITY\NETWORK SERVICE` as the service account. Any other account will not have the required permissions on the `C:\` drive when configuring a non-default server collation for the DB instance.
- If the server collation selected for a new DB instance is the same as that configured on your CEV, the Microsoft SQL Server system databases on the new RDS Custom for SQL Server DB instance do not undergo the rebuild process. Any previous modifications that you made to the system databases on the CEV will automatically persist to the new RDS Custom for SQL Server DB instance.

Supported collations

You can set your collation to one of the values listed in the following table.

Collation	Description
Arabic_100_BIN	Arabic-100, binary sort
Arabic_100_BIN2	Arabic-100, binary code point comparison sort
Arabic_100_CI_AI	Arabic-100, case-insensitive, accent-insensitive , kanatype-insensitive, width-insensitive
Arabic_100_CI_AI_KS	Arabic-100, case-insensitive, accent-insensitive , kanatype-sensitive, width-insensitive
Arabic_100_CI_AI_KS_SC	Arabic-100, case-insensitive, accent-insensitive , kanatype-sensitive, width-insensitive, supplementary characters
Arabic_100_CI_AI_KS_SC_UTF8	Arabic-100, case-insensitive, accent-insensitive , kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Arabic_100_CI_AI_KS_WS	Arabic-100, case-insensitive, accent-insensitive , kanatype-sensitive, width-sensitive
Arabic_100_CI_AI_KS_WS_SC	Arabic-100, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters
Arabic_100_CI_AI_KS_WS_SC_UTF8	Arabic-100, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Arabic_100_CI_AI_SC	Arabic-100, case-insensitive, accent-insensitive , kanatype-insensitive, width-insensitive, supplementary characters
Arabic_100_CI_AI_SC_UTF8	Arabic-100, case-insensitive, accent-insensitive , kanatype-insensitive, width-insensitive, supplementary characters, UTF8

Collation	Description
Arabic_100_CI_AI_WS	Arabic-100, case-insensitive, accent-insensitive , kanatype-insensitive, width-sensitive
Arabic_100_CI_AI_WS_SC	Arabic-100, case-insensitive, accent-insensitive , kanatype-insensitive, width-sensitive, supplementary characters
Arabic_100_CI_AI_WS_SC_UTF8	Arabic-100, case-insensitive, accent-insensitive , kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Arabic_100_CI_AS	Arabic-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Arabic_100_CI_AS_KS	Arabic-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Arabic_100_CI_AS_KS_SC	Arabic-100, case-insensitive, accent-se nsitive, kanatype-sensitive, width-insensitive, supplementary characters
Arabic_100_CI_AS_KS_SC_UTF8	Arabic-100, case-insensitive, accent-se nsitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Arabic_100_CI_AS_KS_WS	Arabic-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Arabic_100_CI_AS_KS_WS_SC	Arabic-100, case-insensitive, accent-se nsitive, kanatype-sensitive, width-sensitive, supplementary characters
Arabic_100_CI_AS_KS_WS_SC_UTF8	Arabic-100, case-insensitive, accent-se nsitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8

Collation	Description
Arabic_100_CI_AS_SC	Arabic-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
Arabic_100_CI_AS_SC_UTF8	Arabic-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Arabic_100_CI_AS_WS	Arabic-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Arabic_100_CI_AS_WS_SC	Arabic-100, case-insensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters
Arabic_100_CI_AS_WS_SC_UTF8	Arabic-100, case-insensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Arabic_100_CS_AI	Arabic-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Arabic_100_CS_AI_KS	Arabic-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Arabic_100_CS_AI_KS_SC	Arabic-100, case-sensitive, accent-insensitive , kanatype-sensitive, width-insensitive, supplementary characters
Arabic_100_CS_AI_KS_SC_UTF8	Arabic-100, case-sensitive, accent-insensitive , kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Arabic_100_CS_AI_KS_WS	Arabic-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive

Collation	Description
Arabic_100_CS_AI_KS_WS_SC	Arabic-100, case-sensitive, accent-insensitive , kanatype-sensitive, width-sensitive, supplementary characters
Arabic_100_CS_AI_KS_WS_SC_UTF8	Arabic-100, case-sensitive, accent-insensitive , kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Arabic_100_CS_AI_SC	Arabic-100, case-sensitive, accent-insensitive , kanatype-insensitive, width-insensitive, supplementary characters
Arabic_100_CS_AI_SC_UTF8	Arabic-100, case-sensitive, accent-insensitive , kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Arabic_100_CS_AI_WS	Arabic-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Arabic_100_CS_AI_WS_SC	Arabic-100, case-sensitive, accent-insensitive , kanatype-insensitive, width-sensitive, supplementary characters
Arabic_100_CS_AI_WS_SC_UTF8	Arabic-100, case-sensitive, accent-insensitive , kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Arabic_100_CS_AS	Arabic-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Arabic_100_CS_AS_KS	Arabic-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Arabic_100_CS_AS_KS_SC	Arabic-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters

Collation	Description
Arabic_100_CS_AS_KS_SC_UTF8	Arabic-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Arabic_100_CS_AS_KS_WS	Arabic-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Arabic_100_CS_AS_KS_WS_SC	Arabic-100, case-sensitive, accent-se nsitive, kanatype-sensitive, width-sensitive, supplementary characters
Arabic_100_CS_AS_KS_WS_SC_UTF8	Arabic-100, case-sensitive, accent-se nsitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Arabic_100_CS_AS_SC	Arabic-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
Arabic_100_CS_AS_SC_UTF8	Arabic-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Arabic_100_CS_AS_WS	Arabic-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Arabic_100_CS_AS_WS_SC	Arabic-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters
Arabic_100_CS_AS_WS_SC_UTF8	Arabic-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Arabic_BIN	Arabic, binary sort
Arabic_BIN2	Arabic, binary code point comparison sort

Collation	Description
Arabic_CI_AI	Arabic, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Arabic_CI_AI_KS	Arabic, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Arabic_CI_AI_KS_WS	Arabic, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive
Arabic_CI_AI_WS	Arabic, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Arabic_CI_AS	Arabic, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Arabic_CI_AS_KS	Arabic, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Arabic_CI_AS_KS_WS	Arabic, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Arabic_CI_AS_WS	Arabic, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Arabic_CS_AI	Arabic, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Arabic_CS_AI_KS	Arabic, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Arabic_CS_AI_KS_WS	Arabic, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive
Arabic_CS_AI_WS	Arabic, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Arabic_CS_AS	Arabic, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive

Collation	Description
Arabic_CS_AS_KS	Arabic, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Arabic_CS_AS_KS_WS	Arabic, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Arabic_CS_AS_WS	Arabic, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Chinese_PRC_BIN2	Chinese-PRC, binary code point comparison sort
Chinese_PRC_CI_AS	Chinese-PRC, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Chinese_Taiwan_Stroke_CI_AS	Chinese-Taiwan-Stroke, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Danish_Norwegian_CI_AS	Danish-Norwegian, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Finnish_Swedish_CI_AS	Finnish-Swedish, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
French_CI_AS	French, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
German_PhoneBook_100_BIN	German-PhoneBook-100, binary sort
German_PhoneBook_100_BIN2	German-PhoneBook-100, binary code point comparison sort
German_PhoneBook_100_CI_AI	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive

Collation	Description
German_PhoneBook_100_CI_AI_KS	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive
German_PhoneBook_100_CI_AI_KS_SC	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters
German_PhoneBook_100_CI_AI_KS_SC_UTF8	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
German_PhoneBook_100_CI_AI_KS_WS	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive
German_PhoneBook_100_CI_AI_KS_WS_SC	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters
German_PhoneBook_100_CI_AI_KS_WS_SC_UTF8	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
German_PhoneBook_100_CI_AI_SC	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters
German_PhoneBook_100_CI_AI_SC_UTF8	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
German_PhoneBook_100_CI_AI_WS	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive

Collation	Description
German_PhoneBook_100_CI_AI_WS_SC	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters
German_PhoneBook_100_CI_AI_WS_SC_UTF8	German-PhoneBook-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
German_PhoneBook_100_CI_AS	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
German_PhoneBook_100_CI_AS_KS	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive
German_PhoneBook_100_CI_AS_KS_SC	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters
German_PhoneBook_100_CI_AS_KS_SC_UTF8	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
German_PhoneBook_100_CI_AS_KS_WS	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
German_PhoneBook_100_CI_AS_KS_WS_SC	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters
German_PhoneBook_100_CI_AS_KS_WS_SC_UTF8	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8

Collation	Description
German_PhoneBook_100_CI_AS_SC	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
German_PhoneBook_100_CI_AS_SC_UTF8	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
German_PhoneBook_100_CI_AS_WS	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive
German_PhoneBook_100_CI_AS_WS_SC	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters
German_PhoneBook_100_CI_AS_WS_SC_UTF8	German-PhoneBook-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
German_PhoneBook_100_CS_AI	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
German_PhoneBook_100_CS_AI_KS	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive
German_PhoneBook_100_CS_AI_KS_SC	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters
German_PhoneBook_100_CS_AI_KS_SC_UTF8	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8

Collation	Description
German_PhoneBook_100_CS_AI_KS_WS	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive
German_PhoneBook_100_CS_AI_KS_WS_SC	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters
German_PhoneBook_100_CS_AI_KS_WS_SC_UTF8	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
German_PhoneBook_100_CS_AI_SC	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters
German_PhoneBook_100_CS_AI_SC_UTF8	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
German_PhoneBook_100_CS_AI_WS	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive
German_PhoneBook_100_CS_AI_WS_SC	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters
German_PhoneBook_100_CS_AI_WS_SC_UTF8	German-PhoneBook-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
German_PhoneBook_100_CS_AS	German-PhoneBook-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive

Collation	Description
German_PhoneBook_100_CS_AS_KS	German-PhoneBook-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive
German_PhoneBook_100_CS_AS_KS_SC	German-PhoneBook-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters
German_PhoneBook_100_CS_AS_KS_SC_UTF8	German-PhoneBook-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
German_PhoneBook_100_CS_AS_KS_WS	German-PhoneBook-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive
German_PhoneBook_100_CS_AS_KS_WS_SC	German-PhoneBook-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters
German_PhoneBook_100_CS_AS_KS_WS_SC_UTF8	German-PhoneBook-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
German_PhoneBook_BIN	German-PhoneBook, binary sort
German_PhoneBook_BIN2	German-PhoneBook, binary code point comparison sort
German_PhoneBook_CI_AI	German-PhoneBook, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive
German_PhoneBook_CI_AI_KS	German-PhoneBook, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive

Collation	Description
German_PhoneBook_CI_AI_KS_WS	German-PhoneBook, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive
German_PhoneBook_CI_AI_WS	German-PhoneBook, case-insensitive, accent-insensitive, kanatype-insensitive, width-sen sitive
German_PhoneBook_CI_AS	German-PhoneBook, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive
German_PhoneBook_CI_AS_KS	German-PhoneBook, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive
German_PhoneBook_CI_AS_KS_WS	German-PhoneBook, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
German_PhoneBook_CI_AS_WS	German-PhoneBook, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive
German_PhoneBook_CS_AI	German-PhoneBook, case-sensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive
German_PhoneBook_CS_AI_KS	German-PhoneBook, case-sensitive, accent-in sensitive, kanatype-sensitive, width-insensitive
German_PhoneBook_CS_AI_KS_WS	German-PhoneBook, case-sensitive, accent-in sensitive, kanatype-sensitive, width-sensitive
German_PhoneBook_CS_AI_WS	German-PhoneBook, case-sensitive, accent-in sensitive, kanatype-insensitive, width-sensitive
German_PhoneBook_CS_AS	German-PhoneBook, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
German_PhoneBook_CS_AS_KS	German-PhoneBook, case-sensitive, accent-se nsitive, kanatype-sensitive, width-insensitive

Collation	Description
German_PhoneBook_CS_AS_KS_WS	German-PhoneBook, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive
German_PhoneBook_CS_AS_WS	German-PhoneBook, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Hebrew_BIN	Hebrew, binary sort
Hebrew_CI_AS	Hebrew, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_90_BIN	Japanese-90, binary sort
Japanese_90_BIN2	Japanese-90, binary code point comparison sort
Japanese_90_CI_AI	Japanese-90, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_90_CI_AI_KS	Japanese-90, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Japanese_90_CI_AI_KS_SC	Japanese-90, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_90_CI_AI_KS_SC_UTF8	Japanese-90, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_90_CI_AI_KS_WS	Japanese-90, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive
Japanese_90_CI_AI_KS_WS_SC	Japanese-90, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters

Collation	Description
Japanese_90_CI_AI_KS_WS_SC_UTF8	Japanese-90, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_90_CI_AI_SC	Japanese-90, case-insensitive, accent-in sensitive, kanatype-insensitive, width-ins ensitive, supplementary characters
Japanese_90_CI_AI_SC_UTF8	Japanese-90, case-insensitive, accent-in sensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, UTF8
Japanese_90_CI_AI_WS	Japanese-90, case-insensitive, accent-in sensitive, kanatype-insensitive, width-sensitive
Japanese_90_CI_AI_WS_SC	Japanese-90, case-insensitive, accent-in sensitive, kanatype-insensitive, width-sen sitive, supplementary characters
Japanese_90_CI_AI_WS_SC_UTF8	Japanese-90, case-insensitive, accent-in sensitive, kanatype-insensitive, width-sen sitive, supplementary characters, UTF8
Japanese_90_CI_AS	Japanese-90, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Japanese_90_CI_AS_KS	Japanese-90, case-insensitive, accent-se nsitive, kanatype-sensitive, width-insensitive
Japanese_90_CI_AS_KS_SC	Japanese-90, case-insensitive, accent-se nsitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_90_CI_AS_KS_SC_UTF8	Japanese-90, case-insensitive, accent-se nsitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8

Collation	Description
Japanese_90_CI_AS_KS_WS	Japanese-90, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Japanese_90_CI_AS_KS_WS_SC	Japanese-90, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_90_CI_AS_KS_WS_SC_UTF8	Japanese-90, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_90_CI_AS_SC	Japanese-90, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
Japanese_90_CI_AS_SC_UTF8	Japanese-90, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_90_CI_AS_WS	Japanese-90, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Japanese_90_CI_AS_WS_SC	Japanese-90, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters
Japanese_90_CI_AS_WS_SC_UTF8	Japanese-90, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Japanese_90_CS_AI	Japanese-90, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_90_CS_AI_KS	Japanese-90, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive

Collation	Description
Japanese_90_CS_AI_KS_SC	Japanese-90, case-sensitive, accent-insensitive , kanatype-sensitive, width-insensitive, supplementary characters
Japanese_90_CS_AI_KS_SC_UTF8	Japanese-90, case-sensitive, accent-insensitive , kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_90_CS_AI_KS_WS	Japanese-90, case-sensitive, accent-insensitive , kanatype-sensitive, width-sensitive
Japanese_90_CS_AI_KS_WS_SC	Japanese-90, case-sensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_90_CS_AI_KS_WS_SC_UTF8	Japanese-90, case-sensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_90_CS_AI_SC	Japanese-90, case-sensitive, accent-insensitive , kanatype-insensitive, width-insensitive, supplementary characters
Japanese_90_CS_AI_SC_UTF8	Japanese-90, case-sensitive, accent-insensitive , kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_90_CS_AI_WS	Japanese-90, case-sensitive, accent-insensitive , kanatype-insensitive, width-sensitive
Japanese_90_CS_AI_WS_SC	Japanese-90, case-sensitive, accent-insensitive , kanatype-insensitive, width-sensitive, supplementary characters
Japanese_90_CS_AI_WS_SC_UTF8	Japanese-90, case-sensitive, accent-insensitive , kanatype-insensitive, width-sensitive, supplementary characters, UTF8

Collation	Description
Japanese_90_CS_AS	Japanese-90, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_90_CS_AS_KS	Japanese-90, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Japanese_90_CS_AS_KS_SC	Japanese-90, case-sensitive, accent-se nsitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_90_CS_AS_KS_SC_UTF8	Japanese-90, case-sensitive, accent-se nsitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_90_CS_AS_KS_WS	Japanese-90, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Japanese_90_CS_AS_KS_WS_SC	Japanese-90, case-sensitive, accent-se nsitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_90_CS_AS_KS_WS_SC_UTF8	Japanese-90, case-sensitive, accent-se nsitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_90_CS_AS_SC	Japanese-90, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
Japanese_90_CS_AS_SC_UTF8	Japanese-90, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_90_CS_AS_WS	Japanese-90, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive

Collation	Description
Japanese_90_CS_AS_WS_SC	Japanese-90, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters
Japanese_90_CS_AS_WS_SC_UTF8	Japanese-90, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Japanese_BIN	Japanese, binary sort
Japanese_BIN2	Japanese, binary code point comparison sort
Japanese_Bushu_Kakusu_100_BIN	Japanese-Bushu-Kakusu-100, binary sort
Japanese_Bushu_Kakusu_100_BIN2	Japanese-Bushu-Kakusu-100, binary code point comparison sort
Japanese_Bushu_Kakusu_100_CI_AI	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_Bushu_Kakusu_100_CI_AI_KS	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Japanese_Bushu_Kakusu_100_CI_AI_KS_SC	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CI_AI_KS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CI_AI_KS_WS	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive

Collation	Description
Japanese_Bushu_Kakusu_100_CI_AI_KS_WS_SC	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CI_AI_KS_WS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CI_AI_SC	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CI_AI_SC_UTF8	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CI_AI_WS	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Japanese_Bushu_Kakusu_100_CI_AI_WS_SC	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CI_AI_WS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CI_AS	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_Bushu_Kakusu_100_CI_AS_KS	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive

Collation	Description
Japanese_Bushu_Kakusu_100_CI_AS_KS_SC	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CI_AS_KS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CI_AS_KS_WS	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Japanese_Bushu_Kakusu_100_CI_AS_KS_WS_SC	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CI_AS_KS_WS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CI_AS_SC	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CI_AS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CI_AS_WS	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Japanese_Bushu_Kakusu_100_CI_AS_WS_SC	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters

Collation	Description
Japanese_Bushu_Kakusu_100_CI_AS_WS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CS_AI	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_Bushu_Kakusu_100_CS_AI_KS	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Japanese_Bushu_Kakusu_100_CS_AI_KS_SC	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CS_AI_KS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CS_AI_KS_WS	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive
Japanese_Bushu_Kakusu_100_CS_AI_KS_WS_SC	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CS_AI_KS_WS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CS_AI_SC	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters

Collation	Description
Japanese_Bushu_Kakusu_100_CS_AI_SC_UTF8	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CS_AI_WS	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Japanese_Bushu_Kakusu_100_CS_AI_WS_SC	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CS_AI_WS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CS_AS	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_Bushu_Kakusu_100_CS_AS_KS	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Japanese_Bushu_Kakusu_100_CS_AS_KS_SC	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CS_AS_KS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CS_AS_KS_WS	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive

Collation	Description
Japanese_Bushu_Kakusu_100_CS_AS_KS_WS_SC	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CS_AS_KS_WS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CS_AS_SC	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CS_AS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_100_CS_AS_WS	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Japanese_Bushu_Kakusu_100_CS_AS_WS_SC	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters
Japanese_Bushu_Kakusu_100_CS_AS_WS_SC_UTF8	Japanese-Bushu-Kakusu-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Japanese_Bushu_Kakusu_140_BIN	Japanese-Bushu-Kakusu-140, binary sort
Japanese_Bushu_Kakusu_140_BIN2	Japanese-Bushu-Kakusu-140, binary code point comparison sort
Japanese_Bushu_Kakusu_140_CI_AI	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive

Collation	Description
Japanese_Bushu_Kakusu_140_CI_AI_KS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CI_AI_KS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AI_KS_VSS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CI_AI_KS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AI_KS_WS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CI_AI_KS_WS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplement ary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AI_KS_WS_VSS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive

Collation	Description
Japanese_Bushu_Kakusu_140_CI_AI_KS_WS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AI_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AI_VSS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CI_AI_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AI_WS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CI_AI_WS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AI_WS_VSS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive

Collation	Description
Japanese_Bushu_Kakusu_140_CI_AI_WS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CI_AS_KS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CI_AS_KS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AS_KS_VSS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CI_AS_KS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AS_KS_WS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive

Collation	Description
Japanese_Bushu_Kakusu_140_CI_AS_KS_WS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AS_KS_WS_VSS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CI_AS_KS_WS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AS_VSS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CI_AS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AS_WS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive

Collation	Description
Japanese_Bushu_Kakusu_140_CI_AS_WS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CI_AS_WS_VSS	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CI_AS_WS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AI	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CS_AI_KS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CS_AI_KS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AI_KS_VSS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive

Collation	Description
Japanese_Bushu_Kakusu_140_CS_AI_KS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AI_KS_WS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CS_AI_KS_WS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AI_KS_WS_VSS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CS_AI_KS_WS_VSS_UTF8	Japanese-Bushu-Kaku su-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AI_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AI_VSS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive

Collation	Description
Japanese_Bushu_Kakusu_140_CS_AI_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AI_WS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CS_AI_WS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AI_WS_VSS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CS_AI_WS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CS_AS_KS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive

Collation	Description
Japanese_Bushu_Kakusu_140_CS_AS_KS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AS_KS_VSS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CS_AS_KS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AS_KS_WS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CS_AS_KS_WS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AS_KS_WS_VSS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CS_AS_KS_WS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8

Collation	Description
Japanese_Bushu_Kakusu_140_CS_AS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AS_VSS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CS_AS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AS_WS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_Bushu_Kakusu_140_CS_AS_WS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_Bushu_Kakusu_140_CS_AS_WS_VSS	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_Bushu_Kakusu_140_CS_AS_WS_VSS_UTF8	Japanese-Bushu-Kakusu-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8

Collation	Description
Japanese_CI_AI	Japanese, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_CI_AI_KS	Japanese, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Japanese_CI_AI_KS_WS	Japanese, case-insensitive, accent-insensitive, kanatype-sensitive, width-sensitive
Japanese_CI_AI_WS	Japanese, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Japanese_CI_AS	Japanese, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_CI_AS_KS	Japanese, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Japanese_CI_AS_KS_WS	Japanese, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Japanese_CI_AS_WS	Japanese, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Japanese_CS_AI	Japanese, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_CS_AI_KS	Japanese, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Japanese_CS_AI_KS_WS	Japanese, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive
Japanese_CS_AI_WS	Japanese, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Japanese_CS_AS	Japanese, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive

Collation	Description
Japanese_CS_AS_KS	Japanese, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Japanese_CS_AS_KS_WS	Japanese, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Japanese_CS_AS_WS	Japanese, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Japanese_Unicode_BIN	Japanese-Unicode, binary sort
Japanese_Unicode_BIN2	Japanese-Unicode, binary code point comparison sort
Japanese_Unicode_CI_AI	Japanese-Unicode, case-insensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive
Japanese_Unicode_CI_AI_KS	Japanese-Unicode, case-insensitive, accent-in sensitive, kanatype-sensitive, width-insensitive
Japanese_Unicode_CI_AI_KS_WS	Japanese-Unicode, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive
Japanese_Unicode_CI_AI_WS	Japanese-Unicode, case-insensitive, accent-in sensitive, kanatype-insensitive, width-sensitive
Japanese_Unicode_CI_AS	Japanese-Unicode, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Japanese_Unicode_CI_AS_KS	Japanese-Unicode, case-insensitive, accent-se nsitive, kanatype-sensitive, width-insensitive
Japanese_Unicode_CI_AS_KS_WS	Japanese-Unicode, case-insensitive, accent-se nsitive, kanatype-sensitive, width-sensitive
Japanese_Unicode_CI_AS_WS	Japanese-Unicode, case-insensitive, accent-se nsitive, kanatype-insensitive, width-sensitive

Collation	Description
Japanese_Unicode_CS_AI	Japanese-Unicode, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_Unicode_CS_AI_KS	Japanese-Unicode, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Japanese_Unicode_CS_AI_KS_WS	Japanese-Unicode, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive
Japanese_Unicode_CS_AI_WS	Japanese-Unicode, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Japanese_Unicode_CS_AS	Japanese-Unicode, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_Unicode_CS_AS_KS	Japanese-Unicode, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Japanese_Unicode_CS_AS_KS_WS	Japanese-Unicode, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Japanese_Unicode_CS_AS_WS	Japanese-Unicode, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Japanese_XJIS_100_BIN	Japanese-XJIS-100, binary sort
Japanese_XJIS_100_BIN2	Japanese-XJIS-100, binary code point comparison sort
Japanese_XJIS_100_CI_AI	Japanese-XJIS-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_XJIS_100_CI_AI_KS	Japanese-XJIS-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive

Collation	Description
Japanese_XJIS_100_CI_AI_KS_SC	Japanese-XJIS-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-ins ensitive, supplementary characters
Japanese_XJIS_100_CI_AI_KS_SC_UTF8	Japanese-XJIS-100, case-insensitive, accent-insensitive, kanatype-sensitive, width-ins ensitive, supplementary characters, UTF8
Japanese_XJIS_100_CI_AI_KS_WS	Japanese-XJIS-100, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive
Japanese_XJIS_100_CI_AI_KS_WS_SC	Japanese-XJIS-100, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_XJIS_100_CI_AI_KS_WS_SC_UTF8	Japanese-XJIS-100, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_XJIS_100_CI_AI_SC	Japanese-XJIS-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive, supplementary characters
Japanese_XJIS_100_CI_AI_SC_UTF8	Japanese-XJIS-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, UTF8
Japanese_XJIS_100_CI_AI_WS	Japanese-XJIS-100, case-insensitive, accent-in sensitive, kanatype-insensitive, width-sensitive
Japanese_XJIS_100_CI_AI_WS_SC	Japanese-XJIS-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-sen sitive, supplementary characters
Japanese_XJIS_100_CI_AI_WS_SC_UTF8	Japanese-XJIS-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-sen sitive, supplementary characters, UTF8

Collation	Description
Japanese_XJIS_100_CI_AS	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_XJIS_100_CI_AS_KS	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Japanese_XJIS_100_CI_AS_KS_SC	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_XJIS_100_CI_AS_KS_SC_UTF8	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_XJIS_100_CI_AS_KS_WS	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Japanese_XJIS_100_CI_AS_KS_WS_SC	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_XJIS_100_CI_AS_KS_WS_SC_UTF8	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_XJIS_100_CI_AS_SC	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
Japanese_XJIS_100_CI_AS_SC_UTF8	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_XJIS_100_CI_AS_WS	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive

Collation	Description
Japanese_XJIS_100_CI_AS_WS_SC	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters
Japanese_XJIS_100_CI_AS_WS_SC_UTF8	Japanese-XJIS-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Japanese_XJIS_100_CS_AI	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Japanese_XJIS_100_CS_AI_KS	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Japanese_XJIS_100_CS_AI_KS_SC	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_XJIS_100_CS_AI_KS_SC_UTF8	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_XJIS_100_CS_AI_KS_WS	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive
Japanese_XJIS_100_CS_AI_KS_WS_SC	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters
Japanese_XJIS_100_CS_AI_KS_WS_SC_UTF8	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_XJIS_100_CS_AI_SC	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters

Collation	Description
Japanese_XJIS_100_CS_AI_SC_UTF8	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_XJIS_100_CS_AI_WS	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive
Japanese_XJIS_100_CS_AI_WS_SC	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters
Japanese_XJIS_100_CS_AI_WS_SC_UTF8	Japanese-XJIS-100, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, UTF8
Japanese_XJIS_100_CS_AS	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_XJIS_100_CS_AS_KS	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Japanese_XJIS_100_CS_AS_KS_SC	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters
Japanese_XJIS_100_CS_AS_KS_SC_UTF8	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, UTF8
Japanese_XJIS_100_CS_AS_KS_WS	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Japanese_XJIS_100_CS_AS_KS_WS_SC	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters

Collation	Description
Japanese_XJIS_100_CS_AS_KS_WS_SC_UTF8	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, UTF8
Japanese_XJIS_100_CS_AS_SC	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters
Japanese_XJIS_100_CS_AS_SC_UTF8	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, UTF8
Japanese_XJIS_100_CS_AS_WS	Japanese-XJIS-100, case-sensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Japanese_XJIS_140_BIN	Japanese-XJIS-140, binary sort
Japanese_XJIS_140_BIN2	Japanese-XJIS-140, binary code point comparison sort
Japanese_XJIS_140_CI_AI	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CI_AI_KS	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CI_AI_KS_UTF8	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8

Collation	Description
Japanese_XJIS_140_CI_AI_KS_VSS	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-ins ensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CI_AI_KS_VSS_UTF8	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-sensitive, width-ins ensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CI_AI_KS_WS	Japanese-XJIS-140, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CI_AI_KS_WS_UTF8	Japanese-XJIS-140, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CI_AI_KS_WS_VSS	Japanese-XJIS-140, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CI_AI_KS_WS_VSS_UTF8	Japanese-XJIS-140, case-insensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CI_AI_UTF8	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, variation selector insensitive, UTF8

Collation	Description
Japanese_XJIS_140_CI_AI_VSS	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CI_AI_VSS_UTF8	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CI_AI_WS	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CI_AI_WS_UTF8	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CI_AI_WS_VSS	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CI_AI_WS_VSS_UTF8	Japanese-XJIS-140, case-insensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CI_AS	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive

Collation	Description
Japanese_XJIS_140_CI_AS_KS	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CI_AS_KS_UTF8	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CI_AS_KS_VSS	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CI_AS_KS_VSS_UTF8	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CI_AS_KS_WS	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CI_AS_KS_WS_UTF8	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CI_AS_KS_WS_VSS	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive

Collation	Description
Japanese_XJIS_140_CI_AS_KS_WS_VSS_UTF8	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CI_AS_UTF8	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CI_AS_VSS	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CI_AS_VSS_UTF8	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CI_AS_WS	Japanese-XJIS-140, case-insensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CI_AS_WS_UTF8	Japanese-XJIS-140, case-insensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CI_AS_WS_VSS	Japanese-XJIS-140, case-insensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive

Collation	Description
Japanese_XJIS_140_CI_AS_WS_VSS_UTF8	Japanese-XJIS-140, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CS_AI	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CS_AI_KS	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CS_AI_KS_UTF8	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CS_AI_KS_VSS	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CS_AI_KS_VSS_UTF8	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CS_AI_KS_WS	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive

Collation	Description
Japanese_XJIS_140_CS_AI_KS_WS_UTF8	Japanese-XJIS-140, case-sensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CS_AI_KS_WS_VSS	Japanese-XJIS-140, case-sensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CS_AI_KS_WS_VSS_UTF8	Japanese-XJIS-140, case-sensitive, accent-in sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CS_AI_UTF8	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CS_AI_VSS	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CS_AI_VSS_UTF8	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CS_AI_WS	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-sen sitive, supplementary characters, variation selector insensitive

Collation	Description
Japanese_XJIS_140_CS_AI_WS_UTF8	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CS_AI_WS_VSS	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CS_AI_WS_VSS_UTF8	Japanese-XJIS-140, case-sensitive, accent-insensitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CS_AS	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CS_AS_KS	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CS_AS_KS_UTF8	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CS_AS_KS_VSS	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive

Collation	Description
Japanese_XJIS_140_CS_AS_KS_VSS_UTF8	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CS_AS_KS_WS	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CS_AS_KS_WS_UTF8	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CS_AS_KS_WS_VSS	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CS_AS_KS_WS_VSS_UTF8	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CS_AS_UTF8	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CS_AS_VSS	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive

Collation	Description
Japanese_XJIS_140_CS_AS_VSS_UTF8	Japanese-XJIS-140, case-sensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, variation selector sensitive, UTF8
Japanese_XJIS_140_CS_AS_WS	Japanese-XJIS-140, case-sensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CS_AS_WS_UTF8	Japanese-XJIS-140, case-sensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector insensitive, UTF8
Japanese_XJIS_140_CS_AS_WS_VSS	Japanese-XJIS-140, case-sensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CS_AS_WS_VSS_UTF8	Japanese-XJIS-140, case-sensitive, accent-se nsitive, kanatype-insensitive, width-sensitive, supplementary characters, variation selector sensitive, UTF8
Korean_Wansung_CI_AS	Korean-Wansung, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Latin1_General_100_BIN	Latin1-General-100, binary sort
Latin1_General_100_BIN2	Latin1-General-100, binary code point comparison sort
Latin1_General_100_BIN2_UTF8	Latin1-General-100, binary code point comparison sort, UTF8

Collation	Description
Latin1_General_100_CI_AS	Latin1-General-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive
Latin1_General_100_CI_AS_SC_UTF8	Latin1-General-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive, supplementary characters, UTF8
Latin1_General_BIN	Latin1-General, binary sort
Latin1_General_BIN2	Latin1-General, binary code point comparison sort
Latin1_General_CI_AI	Latin1-General, case-insensitive, accent-in sensitive, kanatype-insensitive, width-ins ensitive
Latin1_General_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Latin1_General_CI_AS_KS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-sensitive, width-insensitive
Latin1_General_CS_AS	Latin1-General, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Modern_Spanish_CI_AS	Modern-Spanish, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
SQL_1xCompat_CP850_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 49 on Code Page 850 for non-Unicode Data

Collation	Description
SQL_Latin1_General_CP1_CI_AI	Latin1-General, case-insensitive, accent-in sensitive, kanatype-insensitive, width-ins ensitive for Unicode Data, SQL Server Sort Order 54 on Code Page 1252 for non-Unicode Data
SQL_Latin1_General_CP1_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 52 on Code Page 1252 for non-Unicode Data
SQL_Latin1_General_CP1_CS_AS	Latin1-General, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 51 on Code Page 1252 for non-Unicode Data
SQL_Latin1_General_CP1250_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 82 on Code Page 1250 for non-Unicode Data
SQL_Latin1_General_CP1250_CS_AS	Latin1-General, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 81 on Code Page 1250 for non-Unicode Data
SQL_Latin1_General_CP1251_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 106 on Code Page 1251 for non-Unicode Data
SQL_Latin1_General_CP1251_CS_AS	Latin1-General, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 105 on Code Page 1251 for non-Unicode Data

Collation	Description
SQL_Latin1_General_CP1253_CI_AI	Latin1-General, case-insensitive, accent-in sensitive, kanatype-insensitive, width-ins ensitive for Unicode Data, SQL Server Sort Order 124 on Code Page 1253 for non-Unico de Data
SQL_Latin1_General_CP1253_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 114 on Code Page 1253 for non-Unicode Data
SQL_Latin1_General_CP1253_CS_AS	Latin1-General, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 113 on Code Page 1253 for non-Unicode Data
SQL_Latin1_General_CP1254_CI_AS	Turkish, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 130 on Code Page 1254 for non-Unicode Data
SQL_Latin1_General_CP1254_CS_AS	Turkish, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 129 on Code Page 1254 for non-Unicode Data
SQL_Latin1_General_CP1255_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 138 on Code Page 1255 for non-Unicode Data
SQL_Latin1_General_CP1255_CS_AS	Latin1-General, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 137 on Code Page 1255 for non-Unicode Data

Collation	Description
SQL_Latin1_General_CP1256_CI_AS	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 146 on Code Page 1256 for non-Unicode Data
SQL_Latin1_General_CP1256_CS_AS	Latin1-General, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 145 on Code Page 1256 for non-Unicode Data
SQL_Latin1_General_CP1257_CI_AS	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 154 on Code Page 1257 for non-Unicode Data
SQL_Latin1_General_CP1257_CS_AS	Latin1-General, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 153 on Code Page 1257 for non-Unicode Data
SQL_Latin1_General_CP437_BIN	Latin1-General, binary sort for Unicode Data, SQL Server Sort Order 30 on Code Page 437 for non-Unicode Data
SQL_Latin1_General_CP437_BIN2	Latin1-General, binary code point comparison sort for Unicode Data, SQL Server Sort Order 30 on Code Page 437 for non-Unicode Data
SQL_Latin1_General_CP437_CI_AI	Latin1-General, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 34 on Code Page 437 for non-Unicode Data

Collation	Description
SQL_Latin1_General_CP437_CI_AS	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 32 on Code Page 437 for non-Unicode Data
SQL_Latin1_General_CP437_CS_AS	Latin1-General, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 31 on Code Page 437 for non-Unicode Data
SQL_Latin1_General_CP850_BIN	Latin1-General, binary sort for Unicode Data, SQL Server Sort Order 40 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP850_BIN2	Latin1-General, binary code point comparison sort for Unicode Data, SQL Server Sort Order 40 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP850_CI_AI	Latin1-General, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 44 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP850_CI_AS	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 42 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP850_CS_AS	Latin1-General, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 41 on Code Page 850 for non-Unicode Data

Collation	Description
SQL_Latin1_General_Pref_CP1_CI_AS	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 53 on Code Page 1252 for non-Unicode Data
SQL_Latin1_General_Pref_CP437_CI_AS	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 33 on Code Page 437 for non-Unicode Data
SQL_Latin1_General_Pref_CP850_CI_AS	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 43 on Code Page 850 for non-Unicode Data
Thai_CI_AS	Thai, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive

Local time zone for RDS Custom for SQL Server DB instances

The time zone of an RDS Custom for SQL Server DB instance is set by default. The current default is Coordinated Universal Time (UTC). You can set the time zone of your DB instance to a local time zone instead, to match the time zone of your applications.

You set the time zone when you first create your DB instance. You can create your DB instance by using the AWS Management Console, the Amazon RDS API CreateDBInstance action, or the AWS CLI create-db-instance command.

If your DB instance is part of a Multi-AZ deployment, then when you fail over, your time zone remains the local time zone that you set.

When you request a point-in-time restore, you specify the time to restore to. The time is shown in your local time zone. For more information, see Restoring a DB instance to a specified time for Amazon RDS.

The following are limitations to setting the local time zone on your DB instance:

- You can configure the time zone for a DB instance during instance creation, but you can't modify the time zone of an existing RDS Custom for SQL Server DB instance.
- If the time zone is modified for an existing RDS Custom for SQL Server DB instance, RDS Custom changes the DB instance status to `unsupported-configuration`, and sends event notifications.
- You can't restore a snapshot from a DB instance in one time zone to a DB instance in a different time zone.
- We strongly recommend that you don't restore a backup file from one time zone to a different time zone. If you restore a backup file from one time zone to a different time zone, you must audit your queries and applications for the effects of the time zone change. For more information, see Importing and exporting SQL Server databases using native backup and restore.

Supported time zones

You can set your local time zone to one of the values listed in the following table.

Time zone	Standard time offset	Description	Notes
Afghanistan Standard Time	(UTC+04:30)	Kabul	This time zone doesn't observe daylight saving time.
Alaskan Standard Time	(UTC–09:00)	Alaska	
Aleutian Standard Time	(UTC–10:00)	Aleutian Islands	
Altai Standard Time	(UTC+07:00)	Barnaul, Gorno-Altaysk	
Arab Standard Time	(UTC+03:00)	Kuwait, Riyadh	This time zone doesn't observe daylight saving time.
Arabian Standard Time	(UTC+04:00)	Abu Dhabi, Muscat	

Time zone	Standard time offset	Description	Notes
Arabic Standard Time	(UTC+03:00)	Baghdad	This time zone doesn't observe daylight saving time.
Argentina Standard Time	(UTC–03:00)	City of Buenos Aires	This time zone doesn't observe daylight saving time.
Astrakhan Standard Time	(UTC+04:00)	Astrakhan, Ulyanovsk	
Atlantic Standard Time	(UTC–04:00)	Atlantic Time (Canada)	
AUS Central Standard Time	(UTC+09:30)	Darwin	This time zone doesn't observe daylight saving time.
Aus Central W. Standard Time	(UTC+08:45)	Eucla	
AUS Eastern Standard Time	(UTC+10:00)	Canberra, Melbourne, Sydney	
Azerbaijan Standard Time	(UTC+04:00)	Baku	
Azores Standard Time	(UTC–01:00)	Azores	
Bahia Standard Time	(UTC–03:00)	Salvador	
Bangladesh Standard Time	(UTC+06:00)	Dhaka	This time zone doesn't observe daylight saving time.

Time zone	Standard time offset	Description	Notes
Belarus Standard Time	(UTC+03:00)	Minsk	This time zone doesn't observe daylight saving time.
Bougainville Standard Time	(UTC+11:00)	Bougainville Island	
Canada Central Standard Time	(UTC–06:00)	Saskatchewan	This time zone doesn't observe daylight saving time.
Cape Verde Standard Time	(UTC–01:00)	Cabo Verde Is.	This time zone doesn't observe daylight saving time.
Caucasus Standard Time	(UTC+04:00)	Yerevan	
Cen. Australia Standard Time	(UTC+09:30)	Adelaide	
Central America Standard Time	(UTC–06:00)	Central America	This time zone doesn't observe daylight saving time.
Central Asia Standard Time	(UTC+06:00)	Astana	This time zone doesn't observe daylight saving time.
Central Brazilian Standard Time	(UTC–04:00)	Cuiaba	

Time zone	Standard time offset	Description	Notes
Central Europe Standard Time	(UTC+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Prague	
Central European Standard Time	(UTC+01:00)	Sarajevo, Skopje, Warsaw, Zagreb	
Central Pacific Standard Time	(UTC+11:00)	Solomon Islands, New Caledonia	This time zone doesn't observe daylight saving time.
Central Standard Time	(UTC–06:00)	Central Time (US and Canada)	
Central Standard Time (Mexico)	(UTC–06:00)	Guadalajara, Mexico City, Monterrey	
Chatham Islands Standard Time	(UTC+12:45)	Chatham Islands	
China Standard Time	(UTC+08:00)	Beijing, Chongqing, Hong Kong, Urumqi	This time zone doesn't observe daylight saving time.
Cuba Standard Time	(UTC–05:00)	Havana	
Dateline Standard Time	(UTC–12:00)	International Date Line West	This time zone doesn't observe daylight saving time.

Time zone	Standard time offset	Description	Notes
E. Africa Standard Time	(UTC+03:00)	Nairobi	This time zone doesn't observe daylight saving time.
E. Australia Standard Time	(UTC+10:00)	Brisbane	This time zone doesn't observe daylight saving time.
E. Europe Standard Time	(UTC+02:00)	Chisinau	
E. South America Standard Time	(UTC–03:00)	Brasilia	
Easter Island Standard Time	(UTC–06:00)	Easter Island	
Eastern Standard Time	(UTC–05:00)	Eastern Time (US and Canada)	
Eastern Standard Time (Mexico)	(UTC–05:00)	Chetumal	
Egypt Standard Time	(UTC+02:00)	Cairo	
Ekaterinburg Standard Time	(UTC+05:00)	Ekaterinburg	
Fiji Standard Time	(UTC+12:00)	Fiji	
FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius	

Time zone	Standard time offset	Description	Notes
Georgian Standard Time	(UTC+04:00)	Tbilisi	This time zone doesn't observe daylight saving time.
GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London	This time zone isn't the same as Greenwich Mean Time. This time zone does observe daylight saving time.
Greenland Standard Time	(UTC–03:00)	Greenland	
Greenwich Standard Time	(UTC)	Monrovia, Reykjavik	This time zone doesn't observe daylight saving time.
GTB Standard Time	(UTC+02:00)	Athens, Bucharest	
Haiti Standard Time	(UTC–05:00)	Haiti	
Hawaiian Standard Time	(UTC–10:00)	Hawaii	
India Standard Time	(UTC+05:30)	Chennai, Kolkata, Mumbai, New Delhi	This time zone doesn't observe daylight saving time.
Iran Standard Time	(UTC+03:30)	Tehran	
Israel Standard Time	(UTC+02:00)	Jerusalem	
Jordan Standard Time	(UTC+02:00)	Amman	

Time zone	Standard time offset	Description	Notes
Kaliningrad Standard Time	(UTC+02:00)	Kaliningrad	
Kamchatka Standard Time	(UTC+12:00)	Petropavlovsk-Kamchatsky – Old	
Korea Standard Time	(UTC+09:00)	Seoul	This time zone doesn't observe daylight saving time.
Libya Standard Time	(UTC+02:00)	Tripoli	
Line Islands Standard Time	(UTC+14:00)	Kiritimati Island	
Lord Howe Standard Time	(UTC+10:30)	Lord Howe Island	
Magadan Standard Time	(UTC+11:00)	Magadan	This time zone doesn't observe daylight saving time.
Magallanes Standard Time	(UTC–03:00)	Punta Arenas	
Marquesas Standard Time	(UTC–09:30)	Marquesas Islands	
Mauritius Standard Time	(UTC+04:00)	Port Louis	This time zone doesn't observe daylight saving time.
Middle East Standard Time	(UTC+02:00)	Beirut	
Montevideo Standard Time	(UTC–03:00)	Montevideo	
Morocco Standard Time	(UTC+01:00)	Casablanca	

Time zone	Standard time offset	Description	Notes
Mountain Standard Time	(UTC–07:00)	Mountain Time (US and Canada)	
Mountain Standard Time (Mexico)	(UTC–07:00)	Chihuahua, La Paz, Mazatlan	
Myanmar Standard Time	(UTC+06:30)	Yangon (Rangoon)	This time zone doesn't observe daylight saving time.
N. Central Asia Standard Time	(UTC+07:00)	Novosibirsk	
Namibia Standard Time	(UTC+02:00)	Windhoek	
Nepal Standard Time	(UTC+05:45)	Kathmandu	This time zone doesn't observe daylight saving time.
New Zealand Standard Time	(UTC+12:00)	Auckland, Wellington	
Newfoundland Standard Time	(UTC–03:30)	Newfoundland	
Norfolk Standard Time	(UTC+11:00)	Norfolk Island	
North Asia East Standard Time	(UTC+08:00)	Irkutsk	
North Asia Standard Time	(UTC+07:00)	Krasnoyarsk	
North Korea Standard Time	(UTC+09:00)	Pyongyang	
Omsk Standard Time	(UTC+06:00)	Omsk	

Time zone	Standard time offset	Description	Notes
Pacific SA Standard Time	(UTC–03:00)	Santiago	
Pacific Standard Time	(UTC–08:00)	Pacific Time (US and Canada)	
Pacific Standard Time (Mexico)	(UTC–08:00)	Baja California	
Pakistan Standard Time	(UTC+05:00)	Islamabad, Karachi	This time zone doesn't observe daylight saving time.
Paraguay Standard Time	(UTC–04:00)	Asuncion	
Romance Standard Time	(UTC+01:00)	Brussels, Copenhagen, Madrid, Paris	
Russia Time Zone 10	(UTC+11:00)	Chokurdakh	
Russia Time Zone 11	(UTC+12:00)	Anadyr, Petropavlovsk-Kamchatsky	
Russia Time Zone 3	(UTC+04:00)	Izhevsk, Samara	
Russian Standard Time	(UTC+03:00)	Moscow, St. Petersburg, Volgograd	This time zone doesn't observe daylight saving time.
SA Eastern Standard Time	(UTC–03:00)	Cayenne, Fortaleza	This time zone doesn't observe daylight saving time.

Time zone	Standard time offset	Description	Notes
SA Pacific Standard Time	(UTC–05:00)	Bogota, Lima, Quito, Rio Branco	This time zone doesn't observe daylight saving time.
SA Western Standard Time	(UTC–04:00)	Georgetown, La Paz, Manaus, San Juan	This time zone doesn't observe daylight saving time.
Saint Pierre Standard Time	(UTC–03:00)	Saint Pierre and Miquelon	
Sakhalin Standard Time	(UTC+11:00)	Sakhalin	
Samoa Standard Time	(UTC+13:00)	Samoa	
Sao Tome Standard Time	(UTC+01:00)	Sao Tome	
Saratov Standard Time	(UTC+04:00)	Saratov	
SE Asia Standard Time	(UTC+07:00)	Bangkok, Hanoi, Jakarta	This time zone doesn't observe daylight saving time.
Singapore Standard Time	(UTC+08:00)	Kuala Lumpur, Singapore	This time zone doesn't observe daylight saving time.
South Africa Standard Time	(UTC+02:00)	Harare, Pretoria	This time zone doesn't observe daylight saving time.

Time zone	Standard time offset	Description	Notes
Sri Lanka Standard Time	(UTC+05:30)	Sri Jayawarde nepura	This time zone doesn't observe daylight saving time.
Sudan Standard Time	(UTC+02:00)	Khartoum	
Syria Standard Time	(UTC+02:00)	Damascus	
Taipei Standard Time	(UTC+08:00)	Taipei	This time zone doesn't observe daylight saving time.
Tasmania Standard Time	(UTC+10:00)	Hobart	
Tocantins Standard Time	(UTC–03:00)	Araguaina	
Tokyo Standard Time	(UTC+09:00)	Osaka, Sapporo, Tokyo	This time zone doesn't observe daylight saving time.
Tomsk Standard Time	(UTC+07:00)	Tomsk	
Tonga Standard Time	(UTC+13:00)	Nuku'alofa	This time zone doesn't observe daylight saving time.
Transbaikal Standard Time	(UTC+09:00)	Chita	
Turkey Standard Time	(UTC+03:00)	Istanbul	
Turks And Caicos Standard Time	(UTC–05:00)	Turks and Caicos	

Time zone	Standard time offset	Description	Notes
Ulaanbaatar Standard Time	(UTC+08:00)	Ulaanbaatar	This time zone doesn't observe daylight saving time.
US Eastern Standard Time	(UTC–05:00)	Indiana (East)	
US Mountain Standard Time	(UTC–07:00)	Arizona	This time zone doesn't observe daylight saving time.
UTC	UTC	Coordinated Universal Time	This time zone doesn't observe daylight saving time.
UTC–02	(UTC–02:00)	Coordinated Universal Time–02	This time zone doesn't observe daylight saving time.
UTC–08	(UTC–08:00)	Coordinated Universal Time–08	
UTC–09	(UTC–09:00)	Coordinated Universal Time–09	
UTC–11	(UTC–11:00)	Coordinated Universal Time–11	This time zone doesn't observe daylight saving time.

Time zone	Standard time offset	Description	Notes
UTC+12	(UTC+12:00)	Coordinated Universal Time+12	This time zone doesn't observe daylight saving time.
UTC+13	(UTC+13:00)	Coordinated Universal Time+13	
Venezuela Standard Time	(UTC–04:00)	Caracas	This time zone doesn't observe daylight saving time.
Vladivostok Standard Time	(UTC+10:00)	Vladivostok	
Volgograd Standard Time	(UTC+04:00)	Volgograd	
W. Australia Standard Time	(UTC+08:00)	Perth	This time zone doesn't observe daylight saving time.
W. Central Africa Standard Time	(UTC+01:00)	West Central Africa	This time zone doesn't observe daylight saving time.
W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna	
W. Mongolia Standard Time	(UTC+07:00)	Hovd	

Time zone	Standard time offset	Description	Notes
West Asia Standard Time	(UTC+05:00)	Ashgabat, Tashkent	This time zone doesn't observe daylight saving time.
West Bank Standard Time	(UTC+02:00)	Gaza, Hebron	
West Pacific Standard Time	(UTC+10:00)	Guam, Port Moresby	This time zone doesn't observe daylight saving time.
Yakutsk Standard Time	(UTC+09:00)	Yakutsk	

Using a Service Master Key with RDS Custom for SQL Server

RDS Custom for SQL Server supports using a Service Master Key (SMK). RDS Custom retains the same SMK throughout the lifespan of your RDS Custom for SQL Server DB instance. By retaining the same SMK, your DB instance can use objects that are encrypted with the SMK, such as linked server passwords and credentials. If you use a Multi-AZ deployment, RDS Custom also synchronizes and maintains the SMK between primary and secondary DB instances.

Topics

- Region and version availability
- Supported features
- Using TDE
- Configuring features
- Requirements and limitations

Region and version availability

Using an SMK is supported in all Regions where RDS Custom for SQL Server is available, for all SQL Server versions available on RDS Custom. For more information on version and Region availability

of Amazon RDS with RDS Custom for SQL Server, see Supported Regions and DB engines for RDS Custom for SQL Server.

Supported features

When using a SMK with RDS Custom for SQL Server, the following features are supported:

- Transparent Data Encryption (TDE)
- Column-level encryption
- Database Mail
- Linked Servers
- SQL Server Integration Services (SSIS)

Using TDE

An SMK enables the ability to configure Transparent Data Encryption (TDE), which encrypts data before it is written to storage, and automatically decrypts data when the data is read from storage. Unlike RDS for SQL Server, configuring TDE on an RDS Custom for SQL Server DB instance doesn't require using option groups. Instead, once you've created a certificate and database encryption key, you can run the following command to turn on TDE at the database level:

```
ALTER DATABASE [myDatabase] SET ENCRYPTION ON;
```

For more information on using TDE with RDS for SQL Server, see Support for Transparent Data Encryption in SQL Server.

For detailed information on TDE in Microsoft SQL Server, see Transparent data encryption in the Microsoft documentation.

Configuring features

For detailed steps on configuring features that use a SMK with RDS Custom for SQL Server, you can use the following posts in the Amazon RDS database blog:

- Linked servers: Configuring linked servers on RDS Custom for SQL Server.
- SSIS: Migrate SSIS packages to RDS Custom for SQL Server.
- TDE: Secure your data using TDE on RDS Custom for SQL Server.

Requirements and limitations

When using an SMK with an RDS Custom for SQL Server DB instance, keep in mind the following requirements and limitations:

- If you re-generate the SMK on your DB instance, you should immediately perform a manual DB snapshot. We recommend avoiding re-generating the SMK if possible.
- You must maintain backups of the server certificates and database master key passwords. If you don't maintain backups of these, it may result in data loss.
- If you configure SSIS, you should use an SSM document to join the RDS Custom for SQL Server DB instance to the domain in case of a scale compute or host replacement.
- When TDE or column-encryption is enabled, database backups are automatically encrypted. When you perform a snapshot restore or point in time recovery, the SMK from the source DB instance will be restored to decrypt data for the restore, and a new SMK will be generated to re-encrypt data on the restored instance.

For more information on Service Master Keys in Microsoft SQL Server, see SQL Server and Database Encryption Keys in the Microsoft documentation.

Change data capture (CDC) support with RDS Custom for SQL Server

Overview

RDS Custom for SQL Server provides native support for Change data capture (CDC), enabling you to track and capture data modifications in your SQL Server tables. CDC stores detailed metadata about these changes for subsequent retrieval and analysis. For detailed information about CDC functionality, see Change data capture in the Microsoft documentation.

CDC operation in SQL Server requires matching values between the *local server* (that has `server_id` = 0) in `sys.servers` and `SERVERPROPERTY('ServerName')` identifiers. RDS Custom for SQL Server automatically maintains this synchronization throughout the instance's lifecycle to ensuring continuous CDC functioning even if hosts are replaced during maintenance or recovery operations.

> **Important**
>
> Following a Multi-AZ instance failover, the `SERVERPROPERTY('Servername')` function automatically reflects changes in the network/computer name. However, the

@@SERVERNAME function retains the old server name until the MSSQLSERVER service is restarted. Querying @@SERVERNAME returns the previous server name after a failover. To obtain the accurate server name after a failover, use the following SQL query:

```
SELECT name FROM sys.servers WHERE server_id=0
```

This query provides the most up-to-date server name information without requiring a service restart.

Region and version availability

CDC functionality is supported in all AWS Regions where RDS Custom for SQL Server is available, for all SQL Server versions supported by RDS Custom. For more information about supported versions and Region availability of RDS Custom for SQL Server, see Supported Regions and DB engines for RDS Custom for SQL Server.

Requirements and limitations

When implementing CDC on RDS Custom for SQL Server, be aware the following key considerations:

- If you manually set `@@SERVERNAME` and/or *local server* in `sys.servers` to use features like MS Replication, if the value of the local server (that has `server_id = 0`) in `sys.servers` is set to a format that matches `*.rds.amazonaws.com` or `*.awsrds.*.com`, RDS Custom for SQL Server does not attempt to modify it to match `SERVERPROPERTY('ServerName')`.
- RDS cannot modify the local server (that has `server_id = 0`) in `sys.servers` to a new hostname while remote logins or linked servers are actively using the old hostname. This limitation applies in two scenarios:
  - When a linked server establishes a connection to the local server using a remote login associated with the old hostname
  - When an RDS Custom for SQL Server instance acts as a publisher or distributor and has linked logins associated with the old hostname to its subscriber instances.

Troubleshooting

To identify remote logins or linked logins associated with the old server name, use the following queries. Validate the results and remove these logins to ensure proper CDC functionality.

```
SELECT * FROM sys.remote_logins WHERE server_id=0
```

or

```
select sss.srvname,ssp.name,srl.remote_name  from sys.server_principals ssp
inner join sys.remote_logins srl on srl.local_principal_id=ssp.principal_id
inner join sys.sysservers sss  on srl.server_id = sss.srvid
where sss.srvname = @@SERVERNAME
```

Setting up your environment for Amazon RDS Custom for SQL Server

Before you create and manage a DB instance for Amazon RDS Custom for SQL Server DB instance, make sure to perform the following tasks.

Contents

- Prerequisites for setting up RDS Custom for SQL Server
  - Automated instance profile creation using the AWS Management Console
- Step 1: Grant required permissions to your IAM principal
- Step 2: Configure networking, instance profile, and encryption
  - Configuring with CloudFormation
    - Parameters required by CloudFormation
    - Download CloudFormation template file
    - Configuring resources using CloudFormation
  - Configuring manually
    - Make sure that you have a symmetric encryption AWS KMS key
    - Creating your IAM role and instance profile manually
      - Create the AWSRDSCustomSQLServerInstanceRole IAM role
      - Add an access policy to AWSRDSCustomSQLServerInstanceRole
      - Create your RDS Custom for SQL Server instance profile
      - Add AWSRDSCustomSQLServerInstanceRole to your RDS Custom for SQL Server instance profile
    - Configuring your VPC manually
      - Configure your VPC security group
      - Configure endpoints for dependent AWS services
      - Configure the instance metadata service
- Cross-instance restriction

> **Note**
>
> For a step-by-step tutorial on how to set up prerequisites and launch Amazon RDS Custom for SQL Server, see Get started with Amazon RDS Custom for SQL Server using an

CloudFormation template (Network setup) and Explore the prerequisites required to create an Amazon RDS Custom for SQL Server instance.

Prerequisites for setting up RDS Custom for SQL Server

Before creating an RDS Custom for SQL Server DB instance, make sure that your environment meets the requirements described in this topic. You can also use the CloudFormation template to set up the prerequisites within your AWS account. For more information, see Configuring with CloudFormation

RDS Custom for SQL Server requires that you configure the following prerequisites:

- Configure the AWS Identity and Access Management (IAM) permissions required for instance creation. This is the AWS Identity and Access Management (IAM) user or role needed to make a `create-db-instance` request to RDS.
- Configure prerequisite resources required by RDS Custom for SQL Server DB instance:
  - Configure the AWS KMS key required for encryption of RDS Custom instance. RDS Custom requires a customer managed key at the time of instance creation for encryption. The KMS key ARN, ID, alias ARN, or alias name is passed as `kms-key-id` parameter in the request to create the RDS Custom DB instance.
  - Configure the permissions required inside RDS Custom for SQL Server DB instance. RDS Custom attaches an instance profile to DB instance at creation and uses it for automation within the DB instance. The instance profile name is set to `custom-iam-instance-profile` in the RDS Custom create request. You can create an instance profile from the AWS Management Console or manually create your instance profile. For more information, see Automated instance profile creation using the AWS Management Console and Creating your IAM role and instance profile manually.
  - Configure the networking setup as per the requirements of RDS Custom for SQL Server. RDS Custom instances reside in the subnets (configured with DB subnet group) that you provide at instance creation. These subnets must allow RDS Custom instances to communicate with services required for RDS automation.

Note

For the requirements mentioned above, make sure there aren't any service control policies (SCPs) restricting account level permissions.

If the account that you're using is part of an AWS Organization, it might have service control policies (SCPs) restricting account level permissions. Make sure that the SCPs don't restrict the permissions on users and roles that you create using the following procedures. For more information about SCPs, see Service control policies (SCPs) in the *AWS Organizations User Guide*. Use the describe-organization AWS CLI command to check whether your account is part of an AWS Organization.
For more information about AWS Organizations, see What is AWS Organizations in the *AWS Organizations User Guide*.

For general requirements that apply to RDS Custom for SQL Server, see General requirements for RDS Custom for SQL Server.

Automated instance profile creation using the AWS Management Console

RDS Custom requires you to create and configure an instance profile to launch any RDS Custom for SQL Server DB instance. Use the AWS Management Console to create and attach a new instance profile in a single step. This option is available under RDS Custom security section in the **Create database**, **Restore snapshot**, and **Restore to point in time** console pages. Choose **Create a new instance profile** to provide an instance profile name suffix. The AWS Management Console creates a new instance profile that has the permissions required for RDS Custom automation tasks. To automatically create new instance profiles, your logged-in AWS Management Console user must have `iam:CreateInstanceProfile`, `iam:AddRoleToInstanceProfile`, `iam:CreateRole`, and `iam:AttachRolePolicy` permissions.

Note

This option is only available in the AWS Management Console. If you are using the CLI or SDK, use the RDS Custom provided CloudFormation template or manually create an instance profile. For more information, see Creating your IAM role and instance profile manually.

Step 1: Grant required permissions to your IAM principal

Make sure that you have sufficient access to create an RDS Custom instance. The IAM role or IAM user (referred to as the *IAM principal*) for creating an RDS Custom for SQL Server DB instance using the console or CLI must have either of the following policies for successful DB instance creation:

- The `AdministratorAccess` policy
- The `AmazonRDSFullAccess` policy with the following additional permissions:

```
iam:SimulatePrincipalPolicy
cloudtrail:CreateTrail
cloudtrail:StartLogging
s3:CreateBucket
s3:PutBucketPolicy
s3:PutBucketObjectLockConfiguration
s3:PutBucketVersioning
kms:CreateGrant
kms:DescribeKey
kms:Decrypt
kms:ReEncryptFrom
kms:ReEncryptTo
kms:GenerateDataKeyWithoutPlaintext
kms:GenerateDataKey
ec2:DescribeImages
ec2:RunInstances
ec2:CreateTags
```

RDS Custom uses these permissions during instance creation. These permissions configure resources in your account that are required for RDS Custom operations.

For more information about the `kms:CreateGrant` permission, see AWS KMS key management.

The following sample JSON policy grants the required permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ValidateIamRole",
            "Effect": "Allow",
            "Action": "iam:SimulatePrincipalPolicy",
            "Resource": "*"
        },
        {
            "Sid": "CreateCloudTrail",
```

```
            "Effect": "Allow",
            "Action": [
                "cloudtrail:CreateTrail",
                "cloudtrail:StartLogging"
            ],
            "Resource": "arn:aws:cloudtrail:*:*:trail/do-not-delete-rds-custom-*"
        },
        {
            "Sid": "CreateS3Bucket",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketPolicy",
                "s3:PutBucketObjectLockConfiguration",
                "s3:PutBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::do-not-delete-rds-custom-*"
        },
        {
            "Sid": "CreateKmsGrant",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        }
    ]
}
```

The IAM principal requires the following additional permissions to work with custom engine versions (CEVs):

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ConfigureKmsKeyEncryptionPermission",
            "Effect": "Allow",
```

```
            "Action": [
                "kms:CreateGrant",
                "kms:DescribeKey",
                "kms:Decrypt",
                "kms:ReEncryptFrom",
                "kms:ReEncryptTo",
                "kms:GenerateDataKeyWithoutPlaintext",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/key_id"
        },
        {
            "Sid": "CreateEc2Instance",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages",
                "ec2:RunInstances",
                "ec2:CreateTags"
            ],
            "Resource": "*"
        }
    ]
}
```

Replace *111122223333* with the ID of the account that you are using to create your instance. Replace *us-east-1* with the AWS Region you are using to create your instance. Replace *key_id* with your customer managed key ID. You can add multiple keys as required.

For more information about the resource-level permissions that are required to launch an EC2 instance, see Launch instances (RunInstances).

Also, the IAM principal requires the `iam:PassRole` permission on the IAM role. That must be attached to the instance profile passed in the `custom-iam-instance-profile` parameter in the request to create the RDS Custom DB instance. The instance profile and its attached role are created later in Step 2: Configure networking, instance profile, and encryption.

> **Note**
>
> Make sure that the previously listed permissions aren't restricted by service control policies (SCPs), permission boundaries, or session policies associated with the IAM principal.

Step 2: Configure networking, instance profile, and encryption

You can configure your IAM instance profile role, virtual private cloud (VPC), and AWS KMS symmetric encryption key by using either of the following processes:

- Configuring with CloudFormation (recommended)
- Configuring manually

> **Note**
>
> If your account is part of any AWS Organizations, make sure that the permissions required by the instance profile role aren't restricted by service control policies (SCPs).
> The networking configurations in this topic work best with DB instances that aren't publicly accessible. You can't connect directly to such DB instances from outside the VPC.

Configuring with CloudFormation

To simplify setup, you can use an CloudFormation template file to create a CloudFormation stack. A CloudFormation template creates all the networking, instance profiles, and encryption resources according the requirements of RDS Custom.

To learn how to create stacks, see Creating a stack on the CloudFormation console in the *CloudFormation User Guide*.

For a tutorial on how to launch Amazon RDS Custom for SQL Server using an CloudFormation template, see Get started with Amazon RDS Custom for SQL Server using an CloudFormation template in the *AWS Database Blog* .

Topics

- Parameters required by CloudFormation
- Download CloudFormation template file
- Configuring resources using CloudFormation

Parameters required by CloudFormation

The following parameters are required to configure RDS Custom prerequisite resources with CloudFormation:

Parameter group	Parameter name	Default Value	Description
Availability Configuration	Select an availability configuration for prerequisites setup	Multi-AZ	Specify whether to setup prerequisites in Single-AZ or Multi-AZ configuration for RDS Custom instances . You should use Multi-AZ configuration if you require at least one Multi-AZ DB instance in this configuration
Network Configuration	IPv4 CIDR block for VPC	10.0.0.0/16	Specify an IPv4 CIDR block (or IP address range) for your VPC. This VPC is configured to create and work with RDS Custom DB instance.
	IPv4 CIDR block for 1 of 2 private subnets	10.0.128.0/20	Specify an IPv4 CIDR block (or IP address range) for your first private subnet. This is one of the two subnets in which the RDS Custom DB instance can be created. This is a private subnet with no access to internet.
	IPv4 CIDR block for 2 of 2 private subnets	10.0.144.0/20	Specify an IPv4 CIDR block (or IP address range) for

Parameter group	Parameter name	Default Value	Description
			your second private subnet. This is one of the two subnets in which the RDS Custom DB instance can be created. This is a private subnet with no access to internet.
	IPv4 CIDR block for public subnet	10.0.0.0/20	Specify an IPv4 CIDR block (or IP address range) for your public subnet. This is one of the subnet in which the EC2 instance can connect with RDS Custom DB instance can be created. This is a public subnet with access to internet.
RDP Access Configuration	IPv4 CIDR block of your source	-	Specify an IPv4 CIDR block (or IP address range) of your source. This is the IP range from where you make RDP connection to EC2 instance in the public subnet. If not set, RDP connection to EC2 instance is not configured.

Parameter group	Parameter name	Default Value	Description
	Setup RDP access to RDS Custom for SQL Server instance	No	Specify whether to enable the RDP connection from the EC2 instance to the RDS Custom for SQL Server instance. By default, RDP connection from the EC2 instance to the DB instance is not configured.

Resources created by CloudFormation

Successfully creating the CloudFormation stack using default settings creates the following resources in your AWS account:

- Symmetric encryption KMS key for encryption of data managed by RDS Custom.
- The instance profile is is associated to an IAM role with `AmazonRDSCustomInstanceProfileRolePolicy` to provide permissions required by RDS Custom. For more information, see AmazonRDSCustomServiceRolePolicy in the *AWS Managed Policy Reference Guide*.
- VPC with the CIDR range specified as the CloudFormation parameter. The default value is `10.0.0.0/16`.
- Two private subnets with the CIDR range specified in the parameters, and two different Availability Zones in the AWS Region. The default values for the subnet CIDRs are `10.0.128.0/20` and `10.0.144.0/20`.
- One public subnet with the CIDR range specified in the parameters. The default value for the subnet CIDR is 10.0.0.0/20. The EC2 instance resides in this subnet and can be used to connect to the RDS Custom instance.
- DHCP option set for the VPC with domain name resolution to an Amazon Domain Name System (DNS) server.
- Route table to associate with two private subnets and no access to the internet.

- Route table to associate with public subnet and has access to the internet.
- Internet gateway associated with the VPC to allow internet access to public subnet.
- Network access control list (ACL) to associate with two private subnets and access restricted to HTTPS and DB port within VPC.
- VPC security group to be associated with the RDS Custom instance. Access is restricted for outbound HTTPS to AWS service endpoints that are required by RDS Custom and inbound DB port from EC2 instance security group.
- VPC security group to be associated with the EC2 instance in public subnet. Access is restricted for outbound DB port to RDS Custom instance security group.
- VPC security group to be associated with VPC endpoints that are created for AWS service endpoints that are required by RDS Custom.
- DB subnet group in which RDS Custom instances are created. Two private subnets created by this template are added to the DB subnet group.
- VPC endpoints for each of the AWS service endpoints that are required by RDS Custom.

Setting availability configuration to multi-az will create following resources in addition to above list:

- Network ACL rules allowing communication between private subnets.
- Inbound and outbound access to Multi-AZ port within VPC security group associated with the RDS Custom instance.
- VPC endpoints to AWS service endpoint(s) that are required for Multi-AZ communication.

In addition, setting RDP access configuration creates the following resources:

- Configuring RDP access to public subnet from your source IP address:
  - Network ACL rules that allow RDP connection from your source IP to public subnet.
  - Ingress access to RDP port from your source IP to VPC security group associated with the EC2 instance.
- Configuring RDP access from EC2 instance in public subnet to RDS Custom Instance in private subnets:
  - Network ACL rules allowing RDP connection from public subnet to private subnets.
  - Inbound access to RDP port from VPC security group associated with the EC2 instance to VPC security group associated with the RDS Custom Instance.

Use the following procedures to create the CloudFormation stack for RDS Custom for SQL Server.

Download CloudFormation template file

To download the template file

1. Open the context (right-click) menu for the link custom-sqlserver-onboard.zip and choose **Save Link As**.
2. Save and extract the file to your computer.

Configuring resources using CloudFormation

To configure resources using CloudFormation

1. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. To start the Create Stack wizard, choose **Create Stack**.

   The **Create stack** page appears.
3. For **Prerequisite - Prepare template**, choose **Template is ready**.
4. For **Specify template**, do the following:

   a. For **Template source**, choose **Upload a template file**.

   b. For **Choose file**, navigate to and then choose the correct file.
5. Choose **Next**.

   The **Specify stack details** page appears.
6. For **Stack name**, enter **`rds-custom-sqlserver`**.
7. For **Parameters**, do the following:

   a. To keep the default options, choose **Next**.

   b. To change options, choose the appropriate availability configuration, networking configuration, and RDP access configuration, and then choose **Next**.

      Read the description of each parameter carefully before changing parameters.

> **Note**
>
> If you choose to create at least one Multi-AZ instance in this CloudFormation stack, make sure that the CloudFormation stack parameter **Select an availability configuration for prerequisites setup** is set to `Multi-AZ`. If you create the CloudFormation stack as Single-AZ, update the CloudFormation stack to Multi-AZ configuration before creating the first Multi-AZ instance.

8. On the **Configure stack options page**, choose **Next.**
9. On the **Review rds-custom-sqlserver** page, do the following:
   a. For **Capabilities**, select the **I acknowledge that CloudFormation might create IAM resources with custom names** check box.
   b. Choose **Create stack**.

> **Note**
>
> Do not update the resources created from this CloudFormation stack directly from the resource pages. This prevents you from applying future updates to these resources by using a CloudFormation template.

CloudFormation creates the resources that RDS Custom for SQL Server requires. If the stack creation fails, read through the **Events** tab to see which resource creation failed and its status reason.

The **Outputs** tab for this CloudFormation stack in the console should have information about all resources to be passed as parameters for creating an RDS Custom for SQL Server DB instance. Make sure to use the VPC security group and DB subnet group created by CloudFormation for RDS Custom DB instances. By default, RDS tries to attach the default VPC security group, which might not have the access that you need.

If you used CloudFormation to create resources, you can skip Configuring manually.

Updating the CloudFormation stack

You can also update some of the configuration on the CloudFormation stack after creation. The configurations that can be updated are:

- Availability Configuration for RDS Custom for SQL Server
  - **Select an availability configuration for prerequisites setup**: Update this parameter to switch between Single-AZ and Multi-AZ configuration. If you are using this CloudFormation stack for at least one Multi-AZ instance, you must update the stack to choose Multi-AZ configuration.
- RDP Access Configuration for RDS Custom for SQL Server
  - IPv4 CIDR block of your source: You can update the IPv4 CIDR block (or IP address range) of your source by updating this parameter. Setting this parameter to blank removes RDP access configuration from your source CIDR block to public subnet.
  - Setup RDP access to RDS Custom for SQL Server: Enable or disable the RDP connection from the EC2 instance to the RDS Custom for SQL Server instance.

Deleting the CloudFormation stack

You can delete the CloudFormation stack after deleting all the RDS Custom instances that uses resources from the stack. RDS Custom doesn't keep track of the CloudFormation stack, hence it doesn't block deletion of the stack when there are DB instances that uses stack resources. Make sure that there are no RDS Custom DB instances that uses the stack resources when deleting the stack.

> **Note**
>
> When you delete a CloudFormation stack, all of the resources created by the stack are deleted except the KMS key. The KMS key goes into a pending-deletion state and is deleted after 30 days. To keep the KMS key, perform a CancelKeyDeletion operation during the 30-day grace period.

Configuring manually

If you choose to configure resources manually, perform the following tasks.

> **Note**
>
> To simplify setup, you can use the CloudFormation template file to create a CloudFormation stack rather than a manual configuration. For more information, see Configuring with CloudFormation.
> You can also use the AWS CLI to complete this section. If so, download and install the latest CLI.

Topics

- Make sure that you have a symmetric encryption AWS KMS key
- Creating your IAM role and instance profile manually
- Configuring your VPC manually

Make sure that you have a symmetric encryption AWS KMS key

A symmetric encryption AWS KMS key is required for RDS Custom. When you create an RDS Custom for SQL Server DB instance, make sure to supply the KMS key identifier as parameter `kms-key-id`. For more information, see Creating and connecting to a DB instance for Amazon RDS Custom for SQL Server.

You have the following options:

- If you have an existing customer managed KMS key in your AWS account, you can use it with RDS Custom. No further action is necessary.
- If you already created a customer managed symmetric encryption KMS key for a different RDS Custom engine, you can reuse the same KMS key. No further action is necessary.
- If you don't have an existing customer managed symmetric encryption KMS key in your account, create a KMS key by following the instructions in Creating keys in the *AWS Key Management Service Developer Guide*.
- If you're creating a CEV or RDS Custom DB instance, and your KMS key is in a different AWS account, make sure to use the AWS CLI. You can't use the AWS console with cross-account KMS keys.

> **Important**
>
> RDS Custom doesn't support AWS managed KMS keys.

Make sure that your symmetric encryption key grants access to the `kms:Decrypt` and `kms:GenerateDataKey` operations to the AWS Identity and Access Management (IAM) role in your IAM instance profile. If you have a new symmetric encryption key in your account, no changes are required. Otherwise, make sure that your symmetric encryption key's policy grants access to these operations.

For more information, see Step 4: Configure IAM for RDS Custom for Oracle.

Creating your IAM role and instance profile manually

You can manually create an instance profile and use it to launch RDS Custom instances. If you plan to create the instance in the AWS Management Console, skip this section. The AWS Management Console allows you to create and attach an instance profile to your RDS Custom DB instances. For more information, see Automated instance profile creation using the AWS Management Console.

When you manually create an instance profile, pass the instance profile name as the `custom-iam-instance-profile` parameter to your `create-db-instance` CLI command. RDS Custom uses the role associated with this instance profile to run automation to manage the instance.

To create the IAM instance profile and IAM roles for RDS Custom for SQL Server

1. Create the IAM role named `AWSRDSCustomSQLServerInstanceRole` with a trust policy that lets Amazon EC2 assume this role.
2. Add the AWS Managed Policy `AmazonRDSCustomInstanceProfileRolePolicy` to `AWSRDSCustomSQLServerInstanceRole`.
3. Create an IAM instance profile for RDS Custom for SQL Server that is named `AWSRDSCustomSQLServerInstanceProfile`.
4. Add `AWSRDSCustomSQLServerInstanceRole` to the instance profile.

Create the AWSRDSCustomSQLServerInstanceRole IAM role

The following example creates the `AWSRDSCustomSQLServerInstanceRole` role. The trust policy lets Amazon EC2 assume the role.

```
aws iam create-role \
    --role-name AWSRDSCustomSQLServerInstanceRole \
    --assume-role-policy-document '{
        "Version": "2012-10-17",
          "Statement": [
            {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                  "Service": "ec2.amazonaws.com"
              }
            }
          ]
        }'
```

Add an access policy to AWSRDSCustomSQLServerInstanceRole

To provide the required permissions, attach the AWS managed policy `AmazonRDSCustomInstanceProfileRolePolicy` to `AWSRDSCustomSQLServerInstanceRole`. `AmazonRDSCustomInstanceProfileRolePolicy` allows RDS Custom instances to send and receive messages, and perform various automation actions.

> **Note**
>
> Make sure that the permissions in the access policy aren't restricted by SCPs or permission boundaries associated with the instance profile role.

The following example attaches AWS managed policy `AmazonRDSCustomInstanceProfileRolePolicy` to the `AWSRDSCustomSQLServerInstanceRole` role.

```
aws iam attach-role-policy \
    --role-name AWSRDSCustomSQLServerInstanceRole \
    --policy-arn arn:aws:iam::aws:policy/AmazonRDSCustomInstanceProfileRolePolicy
```

Create your RDS Custom for SQL Server instance profile

An instance profile is a container that includes a single IAM role. RDS Custom uses the instance profile to pass the role to the instance.

If you use the AWS Management Console to create a role for Amazon EC2, the console automatically creates an instance profile and gives it the same name as the role when the role is created. Create your instance profile as follows, naming it `AWSRDSCustomSQLServerInstanceProfile`.

```
aws iam create-instance-profile \
    --instance-profile-name AWSRDSCustomSQLServerInstanceProfile
```

Add AWSRDSCustomSQLServerInstanceRole to your RDS Custom for SQL Server instance profile

Add the `AWSRDSCustomInstanceRoleForRdsCustomInstance` role to the previously created `AWSRDSCustomSQLServerInstanceProfile` profile.

```
aws iam add-role-to-instance-profile \
    --instance-profile-name AWSRDSCustomSQLServerInstanceProfile \
    --role-name AWSRDSCustomSQLServerInstanceRole
```

Configuring your VPC manually

Your RDS Custom DB instance is in a virtual private cloud (VPC) based on the Amazon VPC service, just like an Amazon EC2 instance or Amazon RDS instance. You provide and configure your own VPC. Thus, you have full control over your instance networking setup.

RDS Custom sends communication from your DB instance to other AWS services. Make sure the following services are accessible from the subnet in which you create your RDS Custom DB instances:

- Amazon CloudWatch (`com.amazonaws.`*`region`*`.monitoring`)
- Amazon CloudWatch Logs (`com.amazonaws.`*`region`*`.logs`)
- Amazon CloudWatch Events (`com.amazonaws.`*`region`*`.events`)
- Amazon EC2 (`com.amazonaws.`*`region`*`.ec2` and `com.amazonaws.`*`region`*`.ec2messages`)
- Amazon S3 (`com.amazonaws.`*`region`*`.s3`)
- AWS Secrets Manager (`com.amazonaws.`*`region`*`.secretsmanager`)

- AWS Systems Manager (`com.amazonaws.region.ssm` and `com.amazonaws.region.ssmmessages`)

If creating Multi-AZ deployments

- Amazon Simple Queue Service (`com.amazonaws.region.sqs`)

If RDS Custom can't communicate with the necessary services, it publishes the following events:

```
Database instance in incompatible-network. SSM Agent connection not available. Amazon
 RDS can't connect to the dependent AWS services.
```

```
Database instance in incompatible-network. Amazon RDS can't connect to dependent AWS
 services. Make sure port 443 (HTTPS) allows outbound connections, and try again.
 "Failed to connect to the following services: s3 events"
```

To avoid `incompatible-network` errors, make sure that VPC components involved in communication between your RDS Custom DB instance and AWS services satisfy the following requirements:

- The DB instance can make outbound connections on port 443 to other AWS services.
- The VPC allows incoming responses to requests originating from your RDS Custom DB instance.
- RDS Custom can correctly resolve the domain names of endpoints for each AWS service.

If you already configured a VPC for a different RDS Custom DB engine, you can reuse that VPC and skip this process.

Topics

- Configure your VPC security group
- Configure endpoints for dependent AWS services
- Configure the instance metadata service

Configure your VPC security group

A *security group* acts as a virtual firewall for a VPC instance, controlling both inbound and outbound traffic. An RDS Custom DB instance has a security group attached to its network

interface that protects the instance. Make sure that your security group permits traffic between RDS Custom and other AWS services through HTTPS. You pass this security group as the `vpc-security-group-ids` parameter in the instance creation request.

To configure your security group for RDS Custom

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc.
2. Allow RDS Custom to use the default security group, or create your own security group.

   For detailed instructions, see Provide access to your DB instance in your VPC by creating a security group.
3. Make sure that your security group permits outbound connections on port 443. RDS Custom needs this port to communicate with dependent AWS services.
4. If you have a private VPC and use VPC endpoints, make sure that the security group associated with the DB instance allows outbound connections on port 443 to VPC endpoints. Also make sure that the security group associated with the VPC endpoint allows inbound connections on port 443 from the DB instance.

   If incoming connections aren't allowed, the RDS Custom instance can't connect to the AWS Systems Manager and Amazon EC2 endpoints. For more information, see Create a Virtual Private Cloud endpoint in the *AWS Systems Manager User Guide*.
5. For RDS Custom for SQL Server Multi-AZ instances, make sure that the security group associated with the DB instance allows inbound and outbound connections on port 1120 to this security group itself. This is required for peer host connection on a Multi-AZ RDS Custom for SQL Server DB instance.

For more information about security groups, see Security groups for your VPC in the *Amazon VPC Developer Guide*.

Configure endpoints for dependent AWS services

We recommend that you add endpoints for every service to your VPC using the following instructions. However, you can use any solution that lets your VPC communicate with AWS service endpoints. For example, you can use Network Address Translation (NAT) or AWS Direct Connect.

To configure endpoints for AWS services with which RDS Custom works

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. On the navigation bar, use the Region selector to choose the AWS Region.
3. In the navigation pane, choose **Endpoints**. In the main pane, choose **Create Endpoint**.
4. For **Service category**, choose **AWS services**.
5. For **Service Name**, choose the endpoint shown in the table.
6. For **VPC**, choose your VPC.
7. For **Subnets**, choose a subnet from each Availability Zone to include.

   The VPC endpoint can span multiple Availability Zones. AWS creates an elastic network interface for the VPC endpoint in each subnet that you choose. Each network interface has a Domain Name System (DNS) host name and a private IP address.
8. For **Security group**, choose or create a security group.

   You can use security groups to control access to your endpoint, much as you use a firewall. Make sure that the security group allows inbound connections on port 443 from the DB instances. For more information about security groups, see Security groups for your VPC in the *Amazon VPC User Guide*.
9. Optionally, you can attach a policy to the VPC endpoint. Endpoint policies can control access to the AWS service to which you are connecting. The default policy allows all requests to pass through the endpoint. If you're using a custom policy, make sure that requests from the DB instance are allowed in the policy.
10. Choose **Create endpoint**.

The following table explains how to find the list of endpoints that your VPC needs for outbound communications.

Service	Endpoint format	Notes and links
AWS Systems Manager	Use the following endpoint formats: • `ssm.region.amazonaws.com` • `ssmmessages. region.amazonaws.com`	For the list of endpoints in each Region, see AWS Systems Manager endpoints and quotas in the *Amazon Web Services General Reference*.

Service	Endpoint format	Notes and links
AWS Secrets Manager	Use the endpoint format `secretsmanager.` *region* `.amazonaws.com .`	For the list of endpoints in each Region, see AWS Secrets Manager endpoints and quotas in the *Amazon Web Services General Reference*.
Amazon CloudWatch	Use the following endpoint formats: • For CloudWatch metrics, use `monitoring.` *region* `.amazonaws.com` • For CloudWatch Events, use `events.`*region*`.amazonaws.com` • For CloudWatch Logs, use `logs.`*region*`.amazonaws.com`	For the list of endpoints in every Region, see: • Amazon CloudWatch endpoints and quotas in the *Amazon Web Services General Reference* • Amazon CloudWatch Logs endpoints and quotas in the *Amazon Web Services General Reference* • Amazon CloudWatch Events endpoints and quotas in the *Amazon Web Services General Reference*
Amazon EC2	Use the following endpoint formats: • `ec2.`*region*`.amazonaws.com` • `ec2messages.` *region* `.amazonaws.com`	For the list of endpoints in each Region, see Amazon Elastic Compute Cloud endpoints and quotas in the *Amazon Web Services General Reference*.

Service	Endpoint format	Notes and links
Amazon S3	Use the endpoint format `s3.region.amazonaws.com` .	For the list of endpoints in each Region, see Amazon Simple Storage Service endpoints and quotas in the *Amazon Web Services General Reference*. To learn more about gateway endpoints for Amazon S3, see Endpoints for Amazon S3 in the *Amazon VPC Developer Guide*. To learn how to create an access point, see Creating access points in the *Amazon VPC Developer Guide*. To learn how to create a gateway endpoints for Amazon S3, see Gateway VPC endpoints.
Amazon Simple Queue Service	Use the endpoint format `sqs.region.amazonaws.com`	For the list of endpoints in each Region, see Amazon Simple Queue Service endpoints and quotas.

Configure the instance metadata service

Make sure that your instance can do the following:

- Access the instance metadata service using Instance Metadata Service Version 2 (IMDSv2).
- Allow outbound communications through port 80 (HTTP) to the IMDS link IP address.
- Request instance metadata from `http://169.254.169.254`, the IMDSv2 link.

For more information, see Use IMDSv2 in the *Amazon EC2 User Guide*.

Cross-instance restriction

When you create an instance profile by following the steps above, it uses the AWS managed policy `AmazonRDSCustomInstanceProfileRolePolicy` to provide the required permissions to RDS Custom which allows instance management and monitoring automation. The managed policy ensures that the permissions allow access to only those resources which RDS Custom requires to run automation. We recommend using the managed policy to support new features and address security requirements which are automatically applied to existing instance profiles without manual intervention. For more information, see AWS managed policy: AmazonRDSCustomInstanceProfileRolePolicy.

The `AmazonRDSCustomInstanceProfileRolePolicy` managed policy restricts the instance profile to have cross-account access but it might allow access to some RDS Custom managed resources across RDS Custom instances within the same account. Based on your requirement, you can use permission boundaries to further restrict cross-instance access. Permission boundaries define the maximum permissions that the identity-based policies can grant to an entity, but doesn't grant permissions by themselves. For more information, see Evaluating effective permissions with boundaries.

For example, the following boundary policy restricts instance profile role to access a specific AWS KMS key and limits access to RDS Custom managed resources across instances which are using different AWS KMS keys.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DenyOtherKmsKeyAccess",
            "Effect": "Deny",
            "Action": "kms:*",
            "NotResource": "arn:aws:kms:us-east-1:111122223333:key/KMS_key_ID"
        }
    ]
}
```

Note

Make sure the permissions boundary does not block any permissions that `AmazonRDSCustomInstanceProfileRolePolicy` grants to RDS Custom.

Bring Your Own Media with RDS Custom for SQL Server

RDS Custom for SQL Server supports two licensing models: License Included (LI) and Bring Your Own Media (BYOM).

With BYOM, you can do the following:

1. Provide and install your own Microsoft SQL Server binaries with supported cumulative updates (CU) on an AWS EC2 Windows AMI.
2. Save the AMI as a golden image, which is a template that you can use to create a custom engine version (CEV).
3. Create a CEV from your golden image.
4. Create new RDS Custom for SQL Server DB instances by using your CEV.

Amazon RDS then manages your DB instances for you.

> **Note**
>
> If you also have a License Included (LI) RDS Custom for SQL Server DB instance, you can't use the SQL Server software from this DB instance with BYOM. You must bring your own SQL Server binaries to BYOM.

Requirements for BYOM for RDS Custom for SQL Server

The same general requirements for custom engine versions with RDS Custom for SQL Server also apply to BYOM. For more information, see Requirements for RDS Custom for SQL Server CEVs.

When using BYOM, make you sure that you meet the following additional requirements:

- Use one of the following supported editions: SQL Server 2022 or 2019 Enterprise, Standard, or Developer edition.
- Grant the SQL Server sysadmin (SA) server role privilege to `NT AUTHORITY\SYSTEM`.
- Open TCP port 1433 and UDP port 1434 to allow SSM connections.

Limitations of BYOM for RDS Custom for SQL Server

The same general limitations for RDS Custom for SQL Server also apply to BYOM. For more information, see Requirements and limitations for Amazon RDS Custom for SQL Server.

With BYOM, the following additional limitations apply:

- Only the default SQL Server instance (MSSQLSERVER) is supported. Named SQL Server instances aren't supported. RDS Custom for SQL Server detects and monitors only the default SQL Server instance.
- Only a single installation of SQL Server is supported on each AMI. Multiple installations of different SQL Server versions aren't supported.
- SQL Server Web edition isn't supported with BYOM.
- Evaluation versions of SQL Server editions aren't supported with BYOM. When you install SQL Server, don't select the checkbox for using an evaluation version.
- Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information, see Region availability for RDS Custom for SQL Server CEVs and Version support for RDS Custom for SQL Server CEVs.

Creating an RDS Custom for SQL Server DB instance with BYOM

To prepare and create an RDS Custom for SQL Server DB instance with BYOM, see Preparing a CEV using Bring Your Own Media (BYOM).

Working with custom engine versions for RDS Custom for SQL Server

A *custom engine version (CEV)* for RDS Custom for SQL Server is an Amazon Machine Image (AMI) that includes Microsoft SQL Server.

The basic steps of the CEV workflow are as follows:

1. Choose an AWS EC2 Windows AMI to use as a base image for a CEV. You have the option to use pre-installed Microsoft SQL Server, or bring your own media to install SQL Server yourself.
2. Install other software on the operating system (OS) and customize the configuration of the OS and SQL Server to meet your enterprise needs.
3. Save the AMI as a golden image
4. Create a custom engine version (CEV) from your golden image.
5. Create new RDS Custom for SQL Server DB instances by using your CEV.

Amazon RDS then manages these DB instances for you.

A CEV allows you to maintain your preferred baseline configuration of the OS and database. Using a CEV ensures that the host configuration, such as any third-party agent installation or other OS customizations, are persisted on RDS Custom for SQL Server DB instances. With a CEV, you can quickly deploy fleets of RDS Custom for SQL Server DB instances with the same configuration.

Topics

- Preparing to create a CEV for RDS Custom for SQL Server
- Creating a CEV for RDS Custom for SQL Server
- Modifying a CEV for RDS Custom for SQL Server
- Viewing CEV details for Amazon RDS Custom for SQL Server
- Deleting a CEV for RDS Custom for SQL Server

Preparing to create a CEV for RDS Custom for SQL Server

You can create a CEV using an Amazon Machine Image (AMI) that contains pre-installed, License Included (LI) Microsoft SQL Server, or with an AMI on which you install your own SQL Server installation media (BYOM).

Preparing a CEV

Use the following procedures to create a CEV using Bring Your Own Media (BYOM) or pre-installed Microsoft SQL Server (LI).

Preparing a CEV using Bring Your Own Media (BYOM)

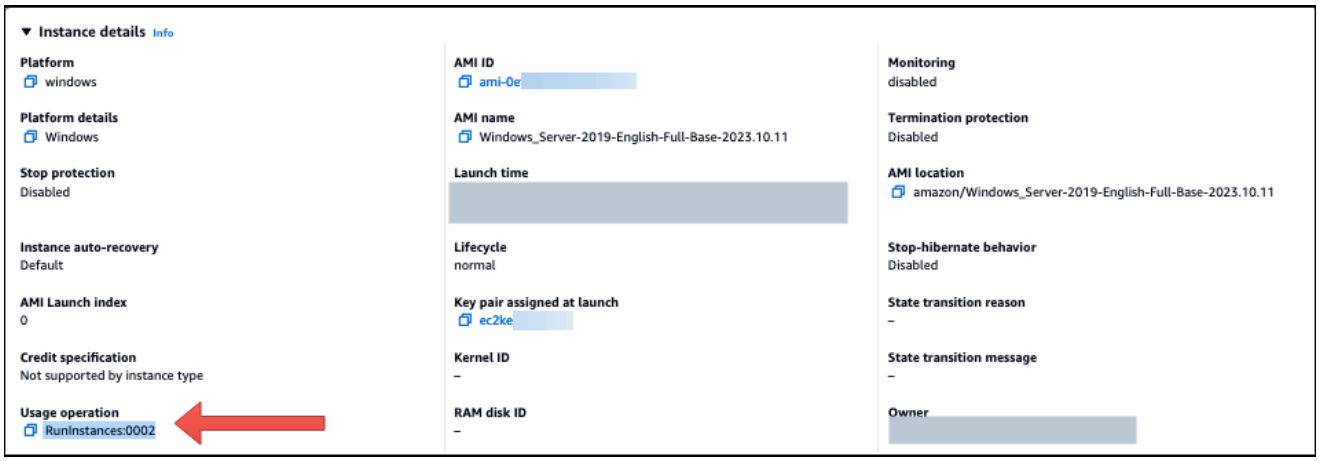
The following steps use an AMI with **Windows Server 2019 Base** as an example.

To create a CEV using BYOM

1. On the Amazon EC2 console, choose **Launch Instance**.
2. For **Name**, enter the name of the instance.
3. Under Quick Start, choose **Windows**.
4. Choose **Microsoft Windows Server 2019 Base**.
5. Choose an appropriate instance type, key pair, network and storage settings, and launch the instance.
6. After launching or creating the EC2 instance, ensure the correct Windows AMI was selected from Step 4:

    a. Select the EC2 instance in the Amazon EC2 console.

    b. In the **Details** section, check the **Usage operation** and ensure that it is set to **RunInstances:0002**.



7. Log in to the EC2 instance and copy your SQL Server installation media to it.

> **Note**
>
> If you're building a CEV using SQL Server Developer edition, you may need to obtain the installation media using your Microsoft Visual Studio subscription.

8. Install SQL Server. Make sure that you do the following:

   a. Review Requirements for BYOM for RDS Custom for SQL Server and Version support for RDS Custom for SQL Server CEVs.

   b. Set the instance root directory to the default `C:\Program Files\Microsoft SQL Server\`. Don't change this directory.

   c. Set the SQL Server Database Engine Account Name to either `NT Service\MSSQLSERVER` or `NT AUTHORITY\NETWORK SERVICE`.

   d. Set the SQL Server Startup mode to **Manual**.

   e. Choose SQL Server Authentication mode as **Mixed**.

   f. Leave the current settings for the default Data directories and TempDB locations.

9. Grant the SQL Server sysadmin (SA) server role privilege to `NT AUTHORITY\SYSTEM`:

```
USE [master]
GO
EXEC master..sp_addsrvrolemember @loginame = N'NT AUTHORITY\SYSTEM' , @rolename =
 N'sysadmin'
GO
```

10. Install additional software or customize the OS and database configuration to meet your requirements.

11. Run Sysprep on the EC2 instance. For more information, see Create an Amazon EC2 AMI using Windows Sysprep.

12. Save the AMI that contains your installed SQL Server version, other software, and customizations. This will be your golden image.

13. Create a new CEV by providing the AMI ID of the image that you created. For detailed steps, see Creating a CEV for RDS Custom for SQL Server.

14. Create a new RDS Custom for SQL Server DB instance using the CEV. For detailed steps, see Create an RDS Custom for SQL Server DB instance from a CEV.

Preparing a CEV using pre-installed SQL Server (LI)

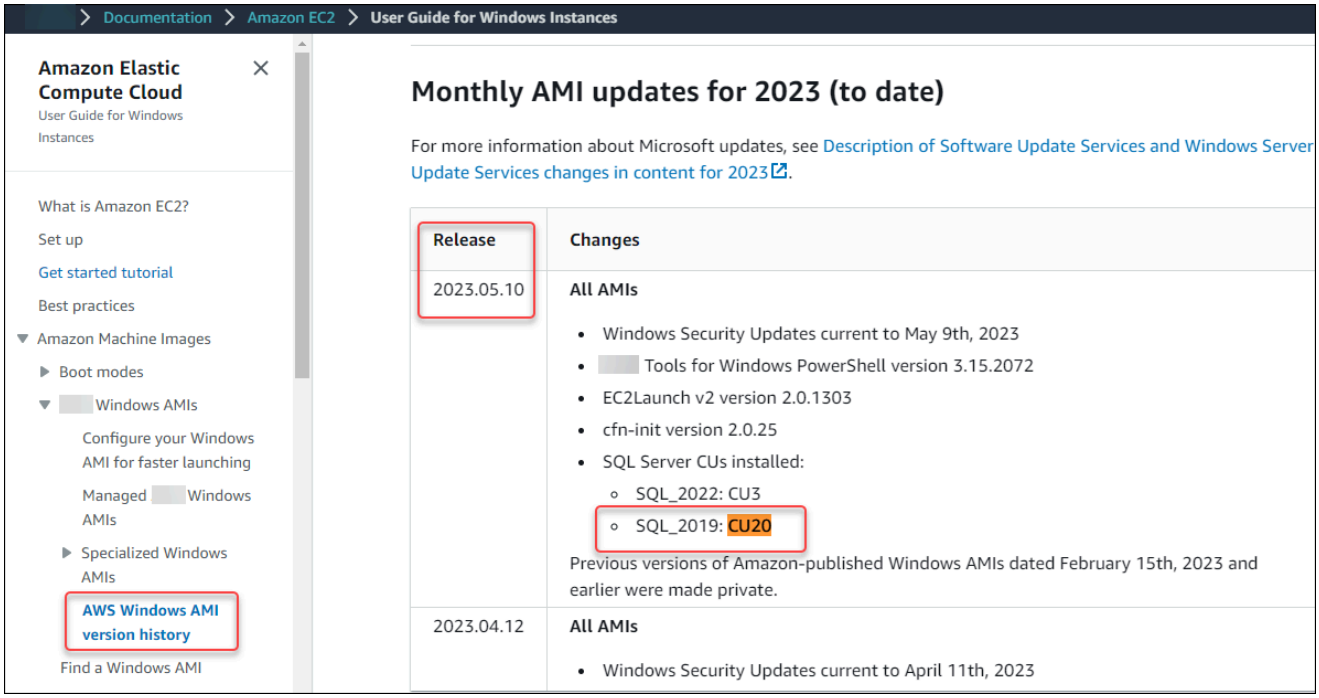
The following steps to create a CEV using pre-installed Microsoft SQL Server (LI) use an AMI with **SQL Server CU20** Release number `2023.05.10` as an example. When you create a CEV, choose an AMI with the most recent release number. This ensures that you are using a supported version of Windows Server and SQL Server with the latest Cumulative Update (CU).

To create a CEV using pre-installed Microsoft SQL Server (LI)

1. Choose the latest available AWS EC2 Windows Amazon Machine Image (AMI) with License Included (LI) Microsoft Windows Server and SQL Server.

   a. Search for **CU20** within the Windows AMI version history.

   b. Note the Release number. For SQL Server 2019 CU20, the release number is `2023.05.10`.



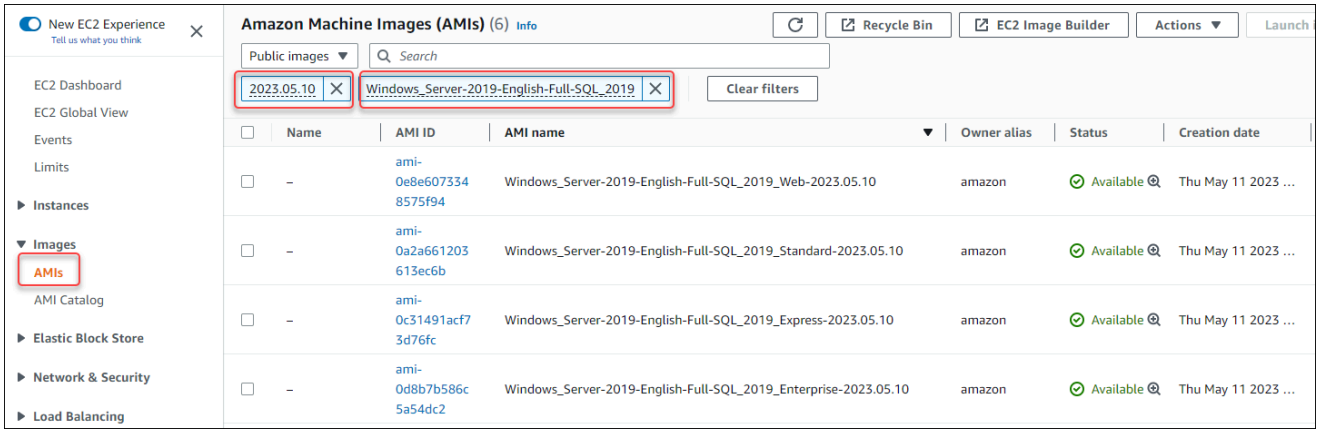
   c. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

   d. In the left navigation panel of the Amazon EC2 console choose **Images**, then **AMIs**.

   e. Choose **Public images**.

   f. Enter `2023.05.10` into the search box. A list of AMIs appears.

   g. Enter `Windows_Server-2019-English-Full-SQL_2019` into the search box to filter the results. The following results should appear.



h. Choose the AMI with the SQL Server edition that you want to use.

2. Create or launch an EC2 instance from your chosen AMI.
3. Log in to the EC2 instance and install additional software or customize the OS and database configuration to meet your requirements.
4. Run Sysprep on the EC2 instance. For more information prepping an AMI using Sysprep, see Create a standardized Amazon Machine Image (AMI) using Sysprep.
5. Save the AMI that contains your installed SQL Server version, other software, and customizations. This will be your golden image.
6. Create a new CEV by providing the AMI ID of the image that you created. For detailed steps on creating a CEV, see Creating a CEV for RDS Custom for SQL Server.
7. Create a new RDS Custom for SQL Server DB instance using the CEV. For detailed steps, see Create an RDS Custom for SQL Server DB instance from a CEV.

Region availability for RDS Custom for SQL Server CEVs

Custom engine version (CEV) support for RDS Custom for SQL Server is available in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (Oregon)
- US West (N. California)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)

- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- South America (São Paulo)

Version support for RDS Custom for SQL Server CEVs

CEV creation for RDS Custom for SQL Server is supported for the following AWS EC2 Windows AMIs:

- For CEVs using pre-installed media, AWS EC2 Windows AMIs with License Included (LI) Microsoft Windows Server 2019 (OS) and SQL Server 2022 or 2019
- For CEVs using bring your own media (BYOM), AWS EC2 Windows AMIs with Microsoft Windows Server 2019 (OS)

CEV creation for RDS Custom for SQL Server is supported for the following operating system (OS) and database editions:

- For CEVs using pre-installed media:
  - SQL Server 2022 Enterprise, Standard, or Web, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR.
  - SQL Server 2019 Enterprise, Standard, or Web, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.
- For CEVs using bring your own media (BYOM):
  - SQL Server 2022 Enterprise, Standard, or Developer, with CU9, CU13, CU14-GDR, CU15-GDR, CU16, CU17, CU18, CU19, CU19-GDR, CU20-GDR, CU21-GDR, CU22 and CU22-GDR.
  - SQL Server 2019 Enterprise, Standard, or Developer, with CU8, CU17, CU18, CU20, CU24, CU26, CU28-GDR, CU29-GDR, CU30, CU32, and CU32-GDR.

- For CEVs using pre-installed media or bring your own media (BYOM), Windows Server 2019 is the only supported OS.

For more information, see AWS Windows AMI version history.

Requirements for RDS Custom for SQL Server CEVs

The following requirements apply to creating a CEV for RDS Custom for SQL Server:

- The AMI used to create a CEV must be based on an OS and database configuration supported by RDS Custom for SQL Server. For more information on supported configurations, see Requirements and limitations for Amazon RDS Custom for SQL Server.
- The CEV must have a unique name. You can't create a CEV with the same name as an existing CEV.
- You must name the CEV using a naming pattern of SQL Server *major version + minor version + customized string*. The *major version + minor version* must match the SQL Server version provided with the AMI. For example, you can name an AMI with SQL Server 2019 CU17 as **15.00.4249.2.my_cevtest**.
- You must prepare an AMI using Sysprep. For more information about prepping an AMI using Sysprep, see Create a standardized Amazon Machine Image (AMI) using Sysprep.
- You are responsible for maintaining the life cycle of the AMI. An RDS Custom for SQL Server DB instance created from a CEV doesn't store a copy of the AMI. It maintains a pointer to the AMI that you used to create the CEV. The AMI must exist for an RDS Custom for SQL Server DB instance to remain operable.

Limitations for RDS Custom for SQL Server CEVs

The following limitations apply to custom engine versions with RDS Custom for SQL Server:

- You can't delete a CEV if there are resources, such as DB instances or DB snapshots, associated with it.
- To create an RDS Custom for SQL Server DB instance, a CEV must have a status of `pending-validation`, `available`, `failed`, or `validating`. You can't create an RDS Custom for SQL Server DB instance using a CEV if the CEV status is `incompatible-image-configuration`.
- To modify a RDS Custom for SQL Server DB instance to use a new CEV, the CEV must have a status of `available`.

- You can't create an AMI or CEV from an existing RDS Custom for SQL Server DB instance.
- You can't modify an existing CEV to use a different AMI. However, you can modify an RDS Custom for SQL Server DB instance to use a different CEV. For more information, see Modifying an RDS Custom for SQL Server DB instance.
- Encrypting an AMI or CEV with a customer-managed KMS key different than the KMS key provided during DB instance creation is not supported.
- Cross-Region copy of CEVs isn't supported.
- Cross-account copy of CEVs isn't supported.
- You can't restore or recover a CEV after you delete it. However, you can create a new CEV from the same AMI.
- A RDS Custom for SQL Server DB instance stores your SQL Server database files in the *D:*\drive. The AMI associated with a CEV should store the Microsoft SQL Server system database files in the *C:*\ drive.
- An RDS Custom for SQL Server DB instance retains your configuration changes made to SQL Server. Any configuration changes to the OS on a running RDS Custom for SQL Server DB instance created from a CEV aren't retained. If you need to make a permanent configuration change to the OS and have it retained as your new baseline configuration, create a new CEV and modify the DB instance to use the new CEV.

  Important

  Modifying an RDS Custom for SQL Server DB instance to use a new CEV is an offline operation. You can perform the modification immediately or schedule it to occur during a weekly maintenance window.

- When you modify a CEV, Amazon RDS doesn't push those modifications to any associated RDS Custom for SQL Server DB instances. You must modify each RDS Custom for SQL Server DB instance to use the new or updated CEV. For more information, see Modifying an RDS Custom for SQL Server DB instance.
- **Important**

  If an AMI used by a CEV is deleted, any modifications that may require host replacement, for example, scale compute, will fail. The RDS Custom for SQL Server DB instance will then be placed outside of the RDS support perimeter. We recommend that you avoid deleting any AMI that's associated to a CEV.

Creating a CEV for RDS Custom for SQL Server

You can create a custom engine version (CEV) using the AWS Management Console or the AWS CLI. You can then use the CEV to create an RDS Custom for SQL Server DB instance.

Make sure that the Amazon Machine Image (AMI) is in the same AWS account and Region as your CEV. Otherwise, the process to create a CEV fails.

For more information, see Creating and connecting to a DB instance for Amazon RDS Custom for SQL Server.

> **Important**
>
> The steps to create a CEV are the same for AMIs created with pre-installed SQL Server and those created using bring your own media (BYOM).

Console

To create a CEV

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Custom engine versions**.

   The **Custom engine versions** page shows all CEVs that currently exist. If you haven't created any CEVs, the table is empty.
3. Choose **Create custom engine version**.
4. For **Engine type**, choose **Microsoft SQL Server**.
5. For **Edition**, choose the DB engine edition that you want to use.
6. For **Major version**, choose the major engine version that's installed on your AMI.
7. In **Version details**, enter a valid name in **Custom engine version name**.

   The name format is *major-engine-version*.*minor-engine-version*.*customized_string*. You can use 1–50 alphanumeric characters, underscores, dashes, and periods. For example, you might enter the name **15.00.4249.2.my_cevtest**.

   Optionally, enter a description for your CEV.

8. For **Installation Media**, browse to or enter the AMI ID that you'd like to create the CEV from.
9. In the **Tags** section, add any tags to identify the CEV.
10. Choose **Create custom engine version**.

The **Custom engine versions** page appears. Your CEV is shown with the status **pending-validation**

AWS CLI

To create a CEV by using the AWS CLI, run the create-custom-db-engine-version command.

The following options are required:

- `--engine`
- `--engine-version`
- `--image-id`

You can also specify the following options:

- `--description`
- `--region`
- `--tags`

The following example creates a CEV named `15.00.4249.2.my_cevtest`. Make sure that the name of your CEV begins with the major engine version number.

Example

For Linux, macOS, or Unix:

```
aws rds create-custom-db-engine-version \
    --engine custom-sqlserver-ee \
    --engine-version 15.00.4249.2.my_cevtest \
    --image-id ami-0r93cx31t5r596482 \
    --description "Custom SQL Server EE 15.00.4249.2 cev test"
```

The following partial output shows the engine, parameter groups, and other information.

```
"DBEngineVersions": [
    {
    "Engine": "custom-sqlserver-ee",
    "MajorEngineVersion": "15.00",
    "EngineVersion": "15.00.4249.2.my_cevtest",
    "DBEngineDescription": "Microsoft SQL Server Enterprise Edition for RDS Custom for
 SQL Server",
    "DBEngineVersionArn": "arn:aws:rds:us-east-1:<my-account-id>:cev:custom-sqlserver-
ee/15.00.4249.2.my_cevtest/a1234a1-123c-12rd-bre1-1234567890",
    "DBEngineVersionDescription": "Custom SQL Server EE 15.00.4249.2 cev test",

    "Image": [
        "ImageId": "ami-0r93cx31t5r596482",
        "Status": "pending-validation"
     ],
    "CreateTime": "2022-11-20T19:30:01.831000+00:00",
    "SupportsLogExportsToCloudwatchLogs": false,
    "SupportsReadReplica": false,
    "Status": "pending-validation",
    "SupportsParallelQuery": false,
    "SupportsGlobalDatabases": false,
    "TagList": []
    }
]
```

If the process to create a CEV fails, RDS Custom for SQL Server issues `RDS-EVENT-0198` with the message `Creation failed for custom engine version` *`major-engine-version.cev_name`*. The message includes details about the failure, for example, the event prints missing files. To find troubleshooting ideas for CEV creation issues, see Troubleshooting CEV errors for RDS Custom for SQL Server.

Create an RDS Custom for SQL Server DB instance from a CEV

After you successfully create a CEV, the **CEV status** shows `pending-validation`. You can now create a new RDS Custom for SQL Server DB instance using the CEV. To create a new RDS Custom for SQL Server DB instance from a CEV, see Creating an RDS Custom for SQL Server DB instance.

Lifecycle of a CEV

The CEV lifecycle includes the following statuses.

CEV status	Description	Troubleshooting suggestions	
`pending-validation`	A CEV was created and is pending the validation of the associated AMI. A CEV will remain in `pending-validation` until an RDS Custom for SQL Server DB instance is created from it.	If there are no existing tasks, create a new RDS Custom for SQL Server DB instance from the CEV. When creating the RDS Custom for SQL Server DB instance, the system attempts to validate the associated AMI for a CEV.	
`validating`	A creation task for the RDS Custom for SQL Server DB instance based on a new CEV is in progress. When creating the RDS Custom for SQL Server DB instance, the system attempts to validate the associated AMI of a CEV.	Wait for the creation task of the existing RDS Custom for SQL Server DB instance to complete. You can use the RDS EVENTS console to review detailed event messages for troubleshooting.	
`available`	The CEV was successfully validated. A CEV will enter the	The CEV doesn't require any additional validation. It can be used to create additional RDS Custom for SQL Server DB instances or modify existing ones.	

CEV status	Description	Troubleshooting suggestions
	`available` status once an RDS Custom for SQL Server DB instance has been successfu lly created from it.	
`inactive`	The CEV has been modified to an inactive state.	You can't create or upgrade an RDS Custom DB instance with this CEV. Also, you can't restore a DB snapshot to create a new RDS Custom DB instance with this CEV. For information about how to change the state to `ACTIVE`, see Modifying a CEV for RDS Custom for SQL Server.
`failed`	The create DB instance step failed for this CEV before it could validate the AMI. Alternatively, the underlying AMI used by the CEV isn't in an available state.	Troubleshoot the root cause for why the system couldn't create the DB instance. View the detailed error message and try to create a new DB instance again. Ensure that the underlying AMI used by the CEV is in an available state.

CEV status	Description	Troubleshooting suggestions
`incompatible-image-configuration`	There was an error validating the AMI.	View the technical details of the error. You can't attempt to validate the AMI with this CEV again. Review the following: recommendations: • Ensure your CEV is named using the required naming pattern of SQL Server *major version + minor version + customized string*. • Ensure the SQL Server version in the CEV name matches the version provided with the AMI. • Ensure the OS build version meets the minimum required build version. • Ensure the OS major version meets the minimum required major version. Create a new CEV using the correct information. If needed, create a new EC2 instance using a supported AMI and run the Sysprep process on it.

Modifying a CEV for RDS Custom for SQL Server

You can modify a CEV using the AWS Management Console or the AWS CLI. You can modify the CEV description or its availability status. Your CEV has one of the following status values:

- `available` – You can use this CEV to create a new RDS Custom DB instance or upgrade a DB instance. This is the default status for a newly created CEV.
- `inactive` – You can't create or upgrade an RDS Custom DB instance with this CEV. You can't restore a DB snapshot to create a new RDS Custom DB instance with this CEV.

You can change the CEV status from `available` to `inactive` or from `inactive` to `available`. You might change the status to `INACTIVE` to prevent the accidental use of a CEV or to make a discontinued CEV eligible for use again.

Console

To modify a CEV

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Custom engine versions**.
3. Choose a CEV whose description or status you want to modify.
4. For **Actions**, choose **Modify**.
5. Make any of the following changes:
   - For **CEV status settings**, choose a new availability status.
   - For **Version description**, enter a new description.
6. Choose **Modify CEV**.

   If the CEV is in use, the console displays **You can't modify the CEV status**. Fix the problems, then try again.

The **Custom engine versions** page appears.

AWS CLI

To modify a CEV by using the AWS CLI, run the modify-custom-db-engine-version command. You can find CEVs to modify by running the describe-db-engine-versions command.

The following options are required:

- `--engine`
- `--engine-version` *`cev`*, where *`cev`* is the name of the custom engine version that you want to modify
- `--status` *`status`*, where *`status`* is the availability status that you want to assign to the CEV

The following example changes a CEV named `15.00.4249.2.my_cevtest` from its current status to `inactive`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-custom-db-engine-version \
    --engine custom-sqlserver-ee \
    --engine-version 15.00.4249.2.my_cevtest \
    --status inactive
```

For Windows:

```
aws rds modify-custom-db-engine-version ^
    --engine custom-sqlserver-ee ^
    --engine-version 15.00.4249.2.my_cevtest ^
    --status inactive
```

To modify an RDS Custom for SQL Server DB instance to use a new CEV, see Modifying an RDS Custom for SQL Server DB instance to use a new CEV.

Modifying an RDS Custom for SQL Server DB instance to use a new CEV

You can modify an existing RDS Custom for SQL Server DB instance to use a different CEV. The changes that you can make include:

- Changing the CEV
- Changing the DB instance class
- Changing the backup retention period and backup window
- Changing the maintenance window

Console

To modify an RDS Custom for SQL Server DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify.
4. Choose **Modify**.

5. Make the following changes as needed:

   a. For **DB engine version**, choose a different CEV.

   b. Change the value for **DB instance class**. For supported classes, see DB instance class support for RDS Custom for SQL Server.

   c. Change the value for **Backup retention period**.

   d. For **Backup window**, set values for the **Start time** and **Duration**.

   e. For **DB instance maintenance window**, set values for the **Start day**, **Start time**, and **Duration**.

6. Choose **Continue**.
7. Choose **Apply immediately** or **Apply during the next scheduled maintenance window**.
8. Choose **Modify DB instance**.

> **Note**
>
> When modifying a DB instance from one CEV to an another CEV, for example, when upgrading a minor version, the SQL Server system databases, including their data and configurations, are persisted from the current RDS Custom for SQL Server DB instance.

AWS CLI

To modify a DB instance to use a different CEV by using the AWS CLI, run the modify-db-instance command.

The following options are required:

- `--db-instance-identifier`
- `--engine-version` *`cev`*, where *`cev`* is the name of the custom engine version that you want the DB instance to change to.

The following example modifies a DB instance named `my-cev-db-instance` to use a CEV named `15.00.4249.2.my_cevtest_new` and applies the change immediately.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-cev-db-instance \
    --engine-version 15.00.4249.2.my_cevtest_new \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-cev-db-instance ^
    --engine-version 15.00.4249.2.my_cevtest_new ^
    --apply-immediately
```

Viewing CEV details for Amazon RDS Custom for SQL Server

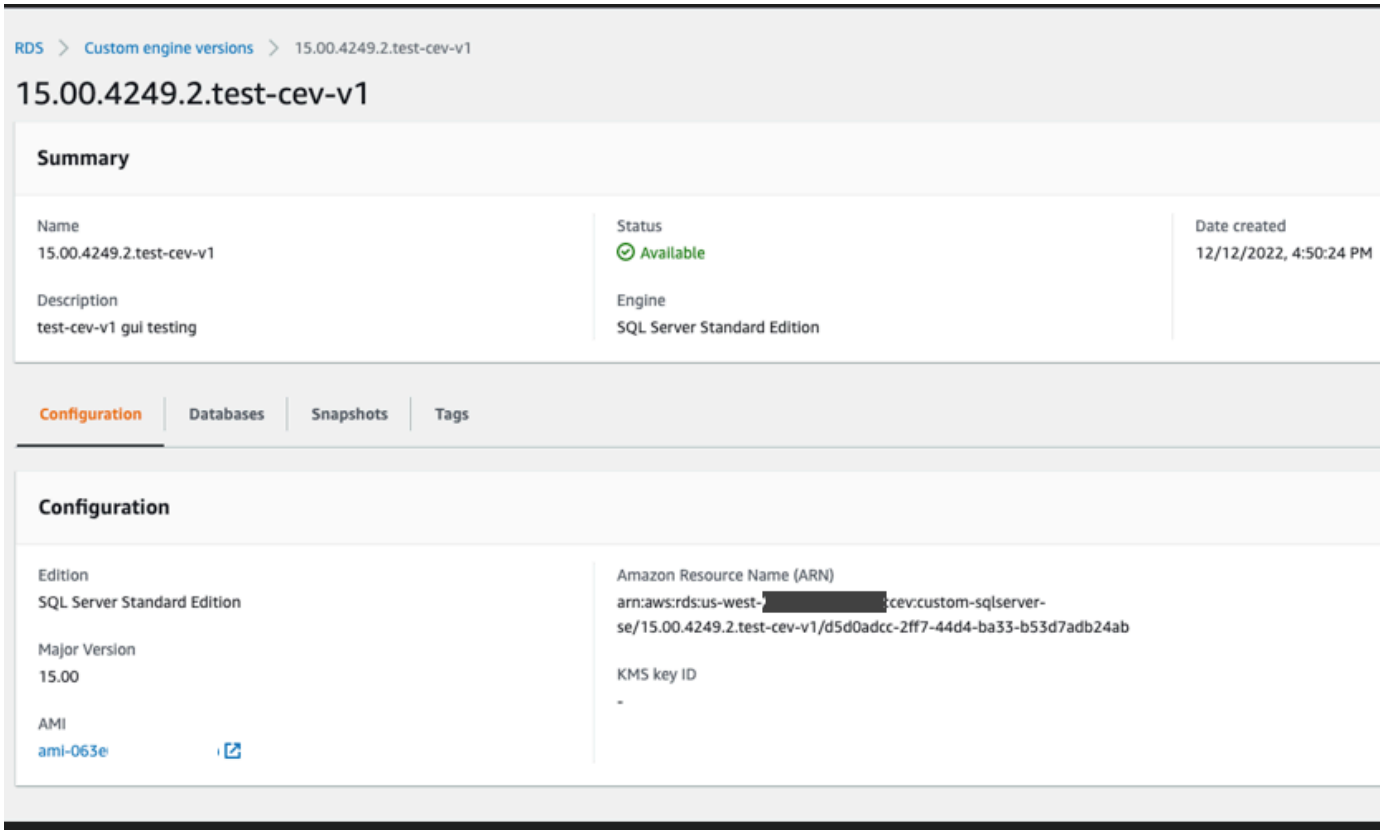
You can view details about your CEV by using the AWS Management Console or the AWS CLI.

Console

To view CEV details

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Custom engine versions**.

   The **Custom engine versions** page shows all CEVs that currently exist. If you haven't created any CEVs, the page is empty.
3. Choose the name of the CEV that you want to view.
4. Choose **Configuration** to view the details.



AWS CLI

To view details about a CEV by using the AWS CLI, run the [describe-db-engine-versions](#) command.

You can also specify the following options:

- `--include-all`, to view all CEVs with any lifecycle state. Without the `--include-all` option, only the CEVs in an `available` lifecycle state will be returned.

```
aws rds describe-db-engine-versions --engine custom-sqlserver-ee --engine-version
 15.00.4249.2.my_cevtest --include-all
{
    "DBEngineVersions": [
        {
            "Engine": "custom-sqlserver-ee",
            "MajorEngineVersion": "15.00",
            "EngineVersion": "15.00.4249.2.my_cevtest",
            "DBParameterGroupFamily": "custom-sqlserver-ee-15.0",
            "DBEngineDescription": "Microsoft SQL Server Enterprise Edition for custom
 RDS",
```

```
            "DBEngineVersionArn": "arn:aws:rds:us-east-1:{my-account-id}:cev:custom-
sqlserver-ee/15.00.4249.2.my_cevtest/a1234a1-123c-12rd-bre1-1234567890",
            "DBEngineVersionDescription": "Custom SQL Server EE 15.00.4249.2 cev test",
            "Image": {
                "ImageId": "ami-0r93cx31t5r596482",
                "Status": "pending-validation"
            },
            "DBEngineMediaType": "AWS Provided",
            "CreateTime": "2022-11-20T19:30:01.831000+00:00",
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": false,
            "SupportedFeatureNames": [],
            "Status": "pending-validation",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "TagList": [],
            "SupportsBabelfish": false
        }
    ]
}
```

You can use filters to view CEVs with a certain lifecycle status. For example, to view CEVs that have a lifecycle status of either `pending-validation`, `available`, or `failed`:

```
aws rds describe-db-engine-versions engine custom-sqlserver-ee
                region us-west-2 include-all query 'DBEngineVersions[?Status ==
 pending-validation ||
                Status == available || Status == failed]'
```

Deleting a CEV for RDS Custom for SQL Server

You can delete a CEV using the AWS Management Console or the AWS CLI. Typically, this task takes a few minutes.

Before deleting a CEV, make sure it isn't being used by any of the following:

- An RDS Custom DB instance
- A snapshot of an RDS Custom DB instance
- An automated backup of your RDS Custom DB instance

Console

To delete a CEV

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Custom engine versions**.
3. Choose a CEV whose description or status you want to delete.
4. For **Actions**, choose **Delete**.

   The **Delete *cev_name*?** dialog box appears.
5. Enter **`delete me`**, and then choose **Delete**.

   In the **Custom engine versions** page, the banner shows that your CEV is being deleted.

AWS CLI

To delete a CEV by using the AWS CLI, run the delete-custom-db-engine-version command.

The following options are required:

- `--engine custom-sqlserver-ee`
- `--engine-version` *`cev`*, where *`cev`* is the name of the custom engine version to be deleted

The following example deletes a CEV named `15.00.4249.2.my_cevtest`.

Example

For Linux, macOS, or Unix:

```
aws rds delete-custom-db-engine-version \
    --engine custom-sqlserver-ee \
    --engine-version 15.00.4249.2.my_cevtest
```

For Windows:

```
aws rds delete-custom-db-engine-version ^
    --engine custom-sqlserver-ee ^
    --engine-version 15.00.4249.2.my_cevtest
```

Creating and connecting to a DB instance for Amazon RDS Custom for SQL Server

You can create an RDS Custom DB instance, and then connect to it using AWS Systems Manager or Remote Desktop Protocol (RDP).

Important

Before you can create or connect to an RDS Custom for SQL Server DB instance, make sure to complete the tasks in Setting up your environment for Amazon RDS Custom for SQL Server.

You can tag RDS Custom DB instances when you create them, but don't create or modify the `AWSRDSCustom` tag that's required for RDS Custom automation. For more information, see Tagging RDS Custom for SQL Server resources.

The first time that you create an RDS Custom for SQL Server DB instance, you might receive the following error: The service-linked role is in the process of being created. Try again later. If you do, wait a few minutes and then try again to create the DB instance.

Topics

- Creating an RDS Custom for SQL Server DB instance
- RDS Custom service-linked role
- Connecting to your RDS Custom DB instance using AWS Systems Manager
- Connecting to your RDS Custom DB instance using RDP

Creating an RDS Custom for SQL Server DB instance

Create an Amazon RDS Custom for SQL Server DB instance using either the AWS Management Console or the AWS CLI. The procedure is similar to the procedure for creating an Amazon RDS DB instance.

For more information, see Creating an Amazon RDS DB instance.

Console

To create an RDS Custom for SQL Server DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose **Create database**.
4. Choose **Standard create** for the database creation method.
5. For **Engine options**, choose **Microsoft SQL Server** for the engine type.
6. For **Database management type**, choose **Amazon RDS Custom**.
7. In the **Edition** section, choose the DB engine edition that you want to use.
8. (Optional) If you intend to create the DB instance from a CEV, check the **Use custom engine version (CEV)** check box. Select your CEV in the drop-down list.
9. For **Database version**, keep the default value version.
10. For **Templates**, choose **Production**.
11. In the **Settings** section, enter a unique name for the **DB instance identifier**.
12. To enter your master password, do the following:

    a. In the **Settings** section, open **Credential Settings**.

    b. Clear the **Auto generate a password** check box.

    c. Change the **Master username** value and enter the same password in **Master password** and **Confirm password**.

    By default, the new RDS Custom DB instance uses an automatically generated password for the master user.
13. In the **DB instance size** section, choose a value for **DB instance class**.

    For supported classes, see DB instance class support for RDS Custom for SQL Server.
14. Choose **Storage** settings.
15. For **RDS Custom security**, do the following:

    a. For **IAM instance profile**, you have two options to choose the instance profile for your RDS Custom for SQL Server DB instance.

1. Choose **Create a new instance profile** and provide an instance profile name suffix. For more information, see Automated instance profile creation using the AWS Management Console.
2. Choose an existing instance profile. From the ddropdown list, choose instance profile that begins with `AWSRDSCustom`.

b. For **Encryption**, choose **Enter a key ARN** to list the available AWS KMS keys. Then choose your key from the list.

An AWS KMS key is required for RDS Custom. For more information, see Make sure that you have a symmetric encryption AWS KMS key.

16. For the remaining sections, specify your preferred RDS Custom DB instance settings. For information about each setting, see Settings for DB instances. The following settings don't appear in the console and aren't supported:

- **Processor features**
- **Storage autoscaling**
- **Availability & durability**
- **Password and Kerberos authentication** option in **Database authentication** (only **Password authentication** is supported)
- **Database options** group in **Additional configuration**
- **Performance Insights**
- **Log exports**
- **Enable auto minor version upgrade**
- **Deletion protection**

Backup retention period is supported, but you can't choose **0 days**.

17. Choose **Create database**.

The **View credential details** button appears on the **Databases** page.

To view the master user name and password for the RDS Custom DB instance, choose **View credential details**.

To connect to the DB instance as the master user, use the user name and password that appear.

> **Important**
>
> You can't view the master user password again. If you don't record it, you might have to change it. To change the master user password after the RDS Custom DB instance is available, modify the DB instance. For more information about modifying a DB instance, see Managing an Amazon RDS Custom for SQL Server DB instance.

18. Choose **Databases** to view the list of RDS Custom DB instances.
19. Choose the RDS Custom DB instance that you just created.

    On the RDS console, the details for the new RDS Custom DB instance appear:

    - The DB instance has a status of **creating** until the RDS Custom DB instance is created and ready for use. When the state changes to **available**, you can connect to the DB instance. Depending on the instance class and storage allocated, it can take several minutes for the new DB instance to be available.
    - **Role** has the value **Instance (RDS Custom)**.
    - **RDS Custom automation mode** has the value **Full automation**. This setting means that the DB instance provides automatic monitoring and instance recovery.

AWS CLI

You create an RDS Custom DB instance by using the create-db-instance AWS CLI command.

The following options are required:

- `--db-instance-identifier`
- `--db-instance-class` (for a list of supported instance classes, see DB instance class support for RDS Custom for SQL Server)
- `--engine` (`custom-sqlserver-ee`, `custom-sqlserver-se`, or `custom-sqlserver-web`)
- `--kms-key-id`
- `--custom-iam-instance-profile`

The following example creates an RDS Custom for SQL Server DB instance named `my-custom-instance`. The backup retention period is 3 days.

> **Note**
>
> To create a DB instance from a custom engine version (CEV), supply an existing CEV name to the `--engine-version` parameter. For example, `--engine-version 15.00.4249.2.my_cevtest`

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --engine custom-sqlserver-ee \
    --engine-version 15.00.4073.23.v1 \
    --db-instance-identifier my-custom-instance \
    --db-instance-class db.m5.xlarge \
    --allocated-storage 20 \
    --db-subnet-group mydbsubnetgroup \
    --master-username myuser \
    --master-user-password mypassword \
    --backup-retention-period 3 \
    --no-multi-az \
    --port 8200 \
    --kms-key-id mykmskey \
    --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance
```

For Windows:

```
aws rds create-db-instance ^
    --engine custom-sqlserver-ee ^
    --engine-version 15.00.4073.23.v1 ^
    --db-instance-identifier my-custom-instance ^
    --db-instance-class db.m5.xlarge ^
    --allocated-storage 20 ^
    --db-subnet-group mydbsubnetgroup ^
    --master-username myuser ^
    --master-user-password mypassword ^
    --backup-retention-period 3 ^
    --no-multi-az ^
    --port 8200 ^
    --kms-key-id mykmskey ^
```

```
    --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Get details about your instance by using the `describe-db-instances` command.

```
aws rds describe-db-instances --db-instance-identifier my-custom-instance
```

The following partial output shows the engine, parameter groups, and other information.

```
{
    "DBInstances": [
        {
            "PendingModifiedValues": {},
            "Engine": "custom-sqlserver-ee",
            "MultiAZ": false,
            "DBSecurityGroups": [],
            "DBParameterGroups": [
                {
                    "DBParameterGroupName": "default.custom-sqlserver-ee-15",
                    "ParameterApplyStatus": "in-sync"
                }
            ],
            "AutomationMode": "full",
            "DBInstanceIdentifier": "my-custom-instance",
            "TagList": []
        }
    ]
}
```

RDS Custom service-linked role

A *service-linked role* gives Amazon RDS Custom access to resources in your AWS account. It makes using RDS Custom easier because you don't have to manually add the necessary permissions. RDS Custom defines the permissions of its service-linked roles, and unless defined otherwise, only RDS Custom can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy can't be attached to any other IAM entity.

When you create an RDS Custom DB instance, both the Amazon RDS and RDS Custom service-linked roles are created (if they don't already exist) and used. For more information, see Using service-linked roles for Amazon RDS.

The first time that you create an RDS Custom for SQL Server DB instance, you might receive the following error: The service-linked role is in the process of being created. Try again later. If you do, wait a few minutes and then try again to create the DB instance.

Connecting to your RDS Custom DB instance using AWS Systems Manager

After you create your RDS Custom DB instance, you can connect to it using AWS Systems Manager Session Manager. Session Manager is a Systems Manager capability that you can use to manage Amazon EC2 instances through a browser-based shell or through the AWS CLI. For more information, see AWS Systems Manager Session Manager.

Console

To connect to your DB instance using Session Manager

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom DB instance to which you want to connect.
3. Choose **Configuration**.
4. Note the **Resource ID** value for your DB instance. For example, the resource ID might be `db-ABCDEFGHIJKLMNOPQRS0123456`.
5. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
6. In the navigation pane, choose **Instances**.
7. Look for the name of your EC2 instance, and then choose the instance ID associated with it. For example, the instance ID might be `i-abcdefghijklm01234`.
8. Choose **Connect**.
9. Choose **Session Manager**.
10. Choose **Connect**.

    A window opens for your session.

AWS CLI

You can connect to your RDS Custom DB instance using the AWS CLI. This technique requires the Session Manager plugin for the AWS CLI. To learn how to install the plugin, see Install the Session Manager plugin for the AWS CLI.

To find the DB resource ID of your RDS Custom DB instance, use `describe-db-instances`.

```
aws rds describe-db-instances \
    --query 'DBInstances[*].[DBInstanceIdentifier,DbiResourceId]' \
    --output text
```

The following sample output shows the resource ID for your RDS Custom instance. The prefix is `db-`.

```
db-ABCDEFGHIJKLMNOPQRS0123456
```

To find the EC2 instance ID of your DB instance, use `aws ec2 describe-instances`. The following example uses `db-ABCDEFGHIJKLMNOPQRS0123456` for the resource ID.

```
aws ec2 describe-instances \
    --filters "Name=tag:Name,Values=db-ABCDEFGHIJKLMNOPQRS0123456" \
    --output text \
    --query 'Reservations[*].Instances[*].InstanceId'
```

The following sample output shows the EC2 instance ID.

```
i-abcdefghijklm01234
```

Use the `aws ssm start-session` command, supplying the EC2 instance ID in the `--target` parameter.

```
aws ssm start-session --target "i-abcdefghijklm01234"
```

A successful connection looks like the following.

```
Starting session with SessionId: yourid-abcdefghijklm1234
[ssm-user@ip-123-45-67-89 bin]$
```

Connecting to your RDS Custom DB instance using RDP

After you create your RDS Custom DB instance, you can connect to this instance using an RDP client. The procedure is the same as for connecting to an Amazon EC2 instance. For more information, see Connect to your Windows instance.

To connect to the DB instance, you need the key pair associated with the instance. RDS Custom creates the key pair for you. The pair name uses the prefix `do-not-delete-rds-custom-`*`DBInstanceIdentifier`*. AWS Secrets Manager stores your private key as a secret.

Complete the task in the following steps:

1. Configure your DB instance to allow RDP connections.
2. Retrieve your secret key.
3. Connect to your EC2 instance using the RDP utility.

Configure your DB instance to allow RDP connections

To allow RDP connections, configure your VPC security group and set a firewall rule on the host.

Configure your VPC security group

Make sure that the VPC security group associated with your DB instance permits inbound connections on port 3389 for Transmission Control Protocol (TCP). To learn how to configure your VPC security group, see Configure your VPC security group.

Set the firewall rule on the host

To permit inbound connections on port 3389 for TCP, set a firewall rule on the host. The following examples show how to do this.

We recommend that you use the specific `-Profile` value: `Public`, `Private`, or `Domain`. Using `Any` refers to all three values. You can also specify a combination of values separated by a comma. For more information about setting firewall rules, see Set-NetFirewallRule in the Microsoft documentation.

To use Systems Manager Session Manager to set a firewall rule

1. Connect to Session Manager as shown in Connecting to your RDS Custom DB instance using AWS Systems Manager.
2. Run the following command.

```
Set-NetFirewallRule -DisplayName "Remote Desktop - User Mode (TCP-In)" -Direction
 Inbound -LocalAddress Any -Profile Any
```

To use Systems Manager CLI commands to set a firewall rule

1. Use the following command to open RDP on the host.

```
OPEN_RDP_COMMAND_ID=$(aws ssm send-command --region $AWS_REGION \
    --instance-ids $RDS_CUSTOM_INSTANCE_EC2_ID \
    --document-name "AWS-RunPowerShellScript" \
    --parameters '{"commands":["Set-NetFirewallRule -DisplayName \"Remote Desktop -
 User Mode (TCP-In)\" -Direction Inbound -LocalAddress Any -Profile Any"]}' \
    --comment "Open RDP port" | jq -r ".Command.CommandId")
```

2. Use the command ID returned in the output to get the status of the previous command. To use the following query to return the command ID, make sure that you have the jq plug-in installed.

```
aws ssm list-commands \
    --region $AWS_REGION \
    --command-id $OPEN_RDP_COMMAND_ID
```

Retrieve your secret key

Retrieve your secret key using either AWS Management Console or the AWS CLI.

Console

To retrieve the secret key

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom DB instance to which you want to connect.
3. Choose the **Configuration** tab.
4. Note the **DB instance ID** for your DB instance, for example, *my-custom-instance*.
5. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

6. In the navigation pane, choose **Instances**.
7. Look for the name of your EC2 instance, and then choose the instance ID associated with it.

   In this example, the instance ID is `i-abcdefghijklm01234`.
8. In **Details**, find **Key pair name**. The pair name includes the DB identifier. In this example, the pair name is `do-not-delete-rds-custom-`*`my-custom-instance`*`-`*`0d726c`*.
9. In the instance summary, find **Public IPv4 DNS**. For the example, the public DNS might be `ec2-12-345-678-901.us-east-2.compute.amazonaws.com`.
10. Open the AWS Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
11. Choose the secret that has the same name as your key pair.
12. Choose **Retrieve secret value**.

AWS CLI

To retrieve the private key

1. Get the list of your RDS Custom DB instances by calling the `aws rds describe-db-instances` command.

   ```
   aws rds describe-db-instances \
       --query 'DBInstances[*].[DBInstanceIdentifier,DbiResourceId]' \
       --output text
   ```
2. Choose the DB instance identifier from the sample output, for example `do-not-delete-rds-custom-`*`my-custom-instance`*.
3. Find the EC2 instance ID of your DB instance by calling the `aws ec2 describe-instances` command. The following example uses the EC2 instance name to describe the DB instance.

   ```
   aws ec2 describe-instances \
       --filters "Name=tag:Name,Values=do-not-delete-rds-custom-my-custom-instance" \
       --output text \
       --query 'Reservations[*].Instances[*].InstanceId'
   ```

   The following sample output shows the EC2 instance ID.

   ```
   i-abcdefghijklm01234
   ```

4. Find the key name by specifying the EC2 instance ID, as shown in the following example.

```
aws ec2 describe-instances \
    --instance-ids i-abcdefghijklm01234 \
    --output text \
    --query 'Reservations[*].Instances[*].KeyName'
```

The following sample output shows the key name, which uses the prefix `do-not-delete-rds-custom-`*`DBInstanceIdentifier`*.

```
do-not-delete-rds-custom-my-custom-instance-0d726c
```

Connect to your EC2 instance using the RDP utility

Follow the procedure in Connect to your Windows instance using RDP in the *Amazon EC2 User Guide*. This procedure assumes that you created a .pem file that contains your private key.

Managing an Amazon RDS Custom for SQL Server DB instance

Amazon RDS Custom for SQL Server supports a subset of the usual management tasks for Amazon RDS DB instances. Following, you can find instructions for the supported RDS Custom for SQL Server management tasks using the AWS Management Console and the AWS CLI.

Topics

- Pausing and resuming RDS Custom automation
- Modifying an RDS Custom for SQL Server DB instance
- Modifying the storage for an RDS Custom for SQL Server DB instance
- Tagging RDS Custom for SQL Server resources
- Deleting an RDS Custom for SQL Server DB instance
- Starting and stopping an RDS Custom for SQL Server DB instance

Pausing and resuming RDS Custom automation

RDS Custom automatically provides monitoring and instance recovery for an RDS Custom for SQL Server DB instance. If you need to customize the instance, do the following:

1. Pause RDS Custom automation for a specified period. The pause ensures that your customizations don't interfere with RDS Custom automation.
2. Customize the RDS Custom for SQL Server DB instance as needed.
3. Do either of the following:
   - Resume automation manually.
   - Wait for the pause period to end. In this case, RDS Custom resumes monitoring and instance recovery automatically.

> **Important**
>
> Pausing and resuming automation are the only supported automation tasks when modifying an RDS Custom for SQL Server DB instance.

Console

To pause or resume RDS Custom automation

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom DB instance that you want to modify.
3. Choose **Modify**. The **Modify DB instance** page appears.
4. For **RDS Custom automation mode**, choose one of the following options:
   - **Paused** pauses the monitoring and instance recovery for the RDS Custom DB instance. Enter the pause duration that you want (in minutes) for **Automation mode duration**. The minimum value is 60 minutes (default). The maximum value is 1,440 minutes.
   - **Full automation** resumes automation.
5. Choose **Continue** to check the summary of modifications.

   A message indicates that RDS Custom will apply the changes immediately.
6. If your changes are correct, choose **Modify DB instance**. Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

   On the RDS console, the details for the modification appear. If you paused automation, the **Status** of your RDS Custom DB instance indicates **Automation paused**.
7. (Optional) In the navigation pane, choose **Databases**, and then your RDS Custom DB instance.

   In the **Summary** pane, **RDS Custom automation mode** indicates the automation status. If automation is paused, the value is **Paused. Automation resumes in *num* minutes**.

AWS CLI

To pause or resume RDS Custom automation, use the `modify-db-instance` AWS CLI command. Identify the DB instance using the required parameter `--db-instance-identifier`. Control the automation mode with the following parameters:

- `--automation-mode` specifies the pause state of the DB instance. Valid values are `all-paused`, which pauses automation, and `full`, which resumes it.

- `--resume-full-automation-mode-minutes` specifies the duration of the pause. The default value is 60 minutes.

> **Note**
>
> Regardless of whether you specify `--no-apply-immediately` or `--apply-immediately`, RDS Custom applies modifications asynchronously as soon as possible.

In the command response, `ResumeFullAutomationModeTime` indicates the resume time as a UTC timestamp. When the automation mode is `all-paused`, you can use `modify-db-instance` to resume automation mode or extend the pause period. No other `modify-db-instance` options are supported.

The following example pauses automation for `my-custom-instance` for 90 minutes.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --automation-mode all-paused \
    --resume-full-automation-mode-minutes 90
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --automation-mode all-paused ^
    --resume-full-automation-mode-minutes 90
```

The following example extends the pause duration for an extra 30 minutes. The 30 minutes is added to the original time shown in `ResumeFullAutomationModeTime`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
```

```
    --db-instance-identifier my-custom-instance \
    --automation-mode all-paused \
    --resume-full-automation-mode-minutes 30
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --automation-mode all-paused ^
    --resume-full-automation-mode-minutes 30
```

The following example resumes full automation for `my-custom-instance`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --automation-mode full \
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --automation-mode full
```

In the following partial sample output, the pending `AutomationMode` value is `full`.

```
{
    "DBInstance": {
        "PubliclyAccessible": true,
        "MasterUsername": "admin",
        "MonitoringInterval": 0,
        "LicenseModel": "bring-your-own-license",
        "VpcSecurityGroups": [
            {
                "Status": "active",
                "VpcSecurityGroupId": "0123456789abcdefg"
            }
        ],
        "InstanceCreateTime": "2020-11-07T19:50:06.193Z",
```

```
        "CopyTagsToSnapshot": false,
        "OptionGroupMemberships": [
            {
                "Status": "in-sync",
                "OptionGroupName": "default:custom-oracle-ee-19"
            }
        ],
        "PendingModifiedValues": {
            "AutomationMode": "full"
        },
        "Engine": "custom-oracle-ee",
        "MultiAZ": false,
        "DBSecurityGroups": [],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.custom-oracle-ee-19",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        ...
        "ReadReplicaDBInstanceIdentifiers": [],
        "AllocatedStorage": 250,
        "DBInstanceArn": "arn:aws:rds:us-west-2:012345678912:db:my-custom-instance",
        "BackupRetentionPeriod": 3,
        "DBName": "ORCL",
        "PreferredMaintenanceWindow": "fri:10:56-fri:11:26",
        "Endpoint": {
            "HostedZoneId": "ABCDEFGHIJKLMNO",
            "Port": 8200,
            "Address": "my-custom-instance.abcdefghijk.us-west-2.rds.amazonaws.com"
        },
        "DBInstanceStatus": "automation-paused",
        "IAMDatabaseAuthenticationEnabled": false,
        "AutomationMode": "all-paused",
        "EngineVersion": "19.my_cev1",
        "DeletionProtection": false,
        "AvailabilityZone": "us-west-2a",
        "DomainMemberships": [],
        "StorageType": "gp2",
        "DbiResourceId": "db-ABCDEFGHIJKLMNOPQRSTUVW",
        "ResumeFullAutomationModeTime": "2020-11-07T20:56:50.565Z",
        "KmsKeyId": "arn:aws:kms:us-west-2:012345678912:key/
aa111a11-111a-11a1-1a11-1111a11a1a1a",
        "StorageEncrypted": false,
```

```
        "AssociatedRoles": [],
        "DBInstanceClass": "db.m5.xlarge",
        "DbInstancePort": 0,
        "DBInstanceIdentifier": "my-custom-instance",
        "TagList": []
    }
```

Modifying an RDS Custom for SQL Server DB instance

Modifying an RDS Custom for SQL Server DB instance is similar to doing this for Amazon RDS, but the changes that you can make are limited to the following:

- Changing the DB instance class
- Changing the backup retention period and backup window
- Changing the maintenance window
- Upgrading the DB engine version when a new version becomes available
- Changing the allocated storage, provisioned IOPS, and storage type
- Allowing and removing Multi-AZ deployments

The following limitations apply to modifying an RDS Custom for SQL Server DB instance:

- Custom DB option and parameter groups aren't supported.
- Any storage volumes that you attach manually to your RDS Custom DB instance are outside the support perimeter.

  For more information, see RDS Custom support perimeter.

Console

To modify an RDS Custom for SQL Server DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify.
4. Choose **Modify**.
5. Make the following changes as needed:

a. For **DB engine version**, choose the new version.

b. Change the value for **DB instance class**. For supported classes, see DB instance class support for RDS Custom for SQL Server

c. Change the value for **Backup retention period**.

d. For **Backup window**, set values for the **Start time** and **Duration**.

e. For **DB instance maintenance window**, set values for the **Start day**, **Start time**, and **Duration**.

6. Choose **Continue**.
7. Choose **Apply immediately** or **Apply during the next scheduled maintenance window**.
8. Choose **Modify DB instance**.

AWS CLI

To modify an RDS Custom for SQL Server DB instance, use the modify-db-instance AWS CLI command. Set the following parameters as needed:

- `--db-instance-class` – For supported classes, see DB instance class support for RDS Custom for SQL Server
- `--engine-version` – The version number of the database engine to which you're upgrading.
- `--backup-retention-period` – How long to retain automated backups, from 0–35 days.
- `--preferred-backup-window` – The daily time range during which automated backups are created.
- `--preferred-maintenance-window` – The weekly time range (in UTC) during which system maintenance can occur.
- `--apply-immediately` – Use `--apply-immediately` to apply the storage changes immediately.

  Or use `--no-apply-immediately` (the default) to apply the changes during the next maintenance window.

Modifying the storage for an RDS Custom for SQL Server DB instance

Modifying storage for an RDS Custom for SQL Server DB instance is similar to modifying storage for an Amazon RDS DB instance, but you can only do the following:

- Increase the allocated storage size.
- Change the storage type. You can use available storage types such as General Purpose or Provisioned IOPS. Provisioned IOPS is supported for the gp3, io1, and io2 Block Express storage types.
- Change the provisioned IOPS, if you're using the volume types that support Provisioned IOPS.

The following limitations apply to modifying the storage for an RDS Custom for SQL Server DB instance:

- The minimum allocated storage size for RDS Custom for SQL Server is 20 GiB. The maximum storage limit for io1, gp2, and gp3 is 16 TiB while io2 supports 64 TiB.
- As with Amazon RDS, you can't decrease the allocated storage. This is a limitation of Amazon Elastic Block Store (Amazon EBS) volumes. For more information, see Working with storage for Amazon RDS DB instances
- Storage autoscaling isn't supported for RDS Custom for SQL Server DB instances.
- Any storage volumes that you manually attach to your RDS Custom DB instance are not considered for storage scaling. Only the RDS-provided default data volumes, i.e., the D drive, are considered for storage scaling.

  For more information, see RDS Custom support perimeter.
- Scaling storage usually doesn't cause any outage or performance degradation of the DB instance. After you modify the storage size for a DB instance, the status of the DB instance is **storage-optimization**.
- Storage optimization can take several hours. You can't make further storage modifications for either six (6) hours or until storage optimization has completed on the instance, whichever is longer. For more information, see Working with storage for Amazon RDS DB instances

For more information about storage, see Amazon RDS DB instance storage.

For general information about storage modification, see Working with storage for Amazon RDS DB instances.

Important

Do not modify storage for your RDS Custom for SQL Server DB instance using Amazon EC2 or Amazon EBS consoles or APIs. Direct storage modifications outside of Amazon RDS console or APIs result in an `unsupported-configuration` state for your database. When you make direct storage changes using Amazon EC2 or Amazon EBS, Amazon RDS cannot track or manage your database instance state. This might cause:

- High availability failover mechanisms from functioning correctly
- Database replication setups to break
- Redundancy features to fail

Modify storage only through Amazon RDS console or APIs to keep your database in a supported state. See Fixing unsupported configurations in RDS Custom for SQL Server for recovery steps.

Console

To modify the storage for an RDS Custom for SQL Server DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify.
4. Choose **Modify**.
5. Make the following changes as needed:
   a. Enter a new value for **Allocated storage**. It must be greater than the current value, and from 20 GiB–16 TiB.
   b. Change the value for **Storage type**. You can choose from the available General Purpose or Provisioned IOPS storage types. Provisioned IOPS is supported for the gp3, io1, and io2 Block Express storage types.
   c. If you're specifying a storage type that supports Provisioned IOPS, you can define the **Provisioned IOPS** value.
6. Choose **Continue**.

7. Choose **Apply immediately** or **Apply during the next scheduled maintenance window**.
8. Choose **Modify DB instance**.

AWS CLI

To modify the storage for an RDS Custom for SQL Server DB instance, use the modify-db-instance AWS CLI command. Set the following parameters as needed:

- `--allocated-storage` – Amount of storage to be allocated for the DB instance, in gibibytes. It must be greater than the current value, and from 20–16,384 GiB.
- `--storage-type` – The storage type, for example, gp2, gp3, io1, or io2.
- `--iops` – Provisioned IOPS for the DB instance. You can specify this only for storage types that support Provisioned IOPS (gp3, io1, and io2).
- `--apply-immediately` – Use `--apply-immediately` to apply the storage changes immediately.

  Or use `--no-apply-immediately` (the default) to apply the changes during the next maintenance window.

The following example changes the storage size of my-custom-instance to 200 GiB, storage type to io1, and Provisioned IOPS to 3000.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --storage-type io1 \
    --iops 3000 \
    --allocated-storage 200 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --storage-type io1 ^
```

```
    --iops 3000 ^
    --allocated-storage 200 ^
    --apply-immediately
```

Tagging RDS Custom for SQL Server resources

You can tag RDS Custom resources as with Amazon RDS resources, but with some important differences:

- Don't create or modify the `AWSRDSCustom` tag that's required for RDS Custom automation. If you do, you might break the automation.
- The `Name` tag is added to RDS Custom resources with the prefix `do-not-delete-rds-custom`. Any customer-passed value for the key is overwritten.
- Tags added to RDS Custom DB instances during creation are propagated to all other related RDS Custom resources.
- Tags aren't propagated when you add them to RDS Custom resources after DB instance creation.

For general information about resource tagging, see Tagging Amazon RDS resources.

Deleting an RDS Custom for SQL Server DB instance

To delete an RDS Custom for SQL Server DB instance, do the following:

- Provide the name of the DB instance.
- Choose or clear the option to take a final DB snapshot of the DB instance.
- Choose or clear the option to retain automated backups.

You can delete an RDS Custom for SQL Server DB instance using the console or the CLI. The time required to delete the DB instance can vary depending on the backup retention period (that is, how many backups to delete), how much data is deleted, and whether a final snapshot is taken.

> **Warning**
>
> Deleting a RDS Custom for SQL Server DB instance will permanently delete the EC2 instance and the associated Amazon EBS volumes. You shouldn’t terminate or delete these resources at any time, otherwise, the deletion and the final snapshot creation may fail.

Note

You can't create a final DB snapshot of your DB instance if it has a status of `creating`, `failed`, `incompatible-create`, `incompatible-restore`, or `incompatible-network`. For more information, see Viewing Amazon RDSDB instance status.

Important

When you choose to take a final snapshot, we recommend that you avoid writing data to your DB instance while the DB instance deletion is in progress. Once the DB instance deletion is initiated, data changes are not guaranteed to be captured by the final snapshot.

Console

To delete an RDS Custom DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the RDS Custom for SQL Server DB instance that you want to delete. RDS Custom for SQL Server DB instances show the role **Instance (RDS Custom for SQL Server)**.
3. For **Actions**, choose **Delete**.
4. To take a final snapshot, choose **Create final snapshot**, and provide a name for the **Final snapshot name**.
5. To retain automated backups, choose **Retain automated backups**.
6. Enter **`delete me`** in the box.
7. Choose **Delete**.

AWS CLI

You delete an RDS Custom for SQL Server DB instance by using the delete-db-instance AWS CLI command. Identify the DB instance using the required parameter `--db-instance-identifier`. The remaining parameters are the same as for an Amazon RDS DB instance.

The following example deletes the RDS Custom for SQL Server DB instance named `my-custom-instance`, takes a final snapshot, and retains automated backups.

Example

For Linux, macOS, or Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier my-custom-instance \
    --no-skip-final-snapshot \
    --final-db-snapshot-identifier my-custom-instance-final-snapshot \
    --no-delete-automated-backups
```

For Windows:

```
aws rds delete-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --no-skip-final-snapshot ^
    --final-db-snapshot-identifier my-custom-instance-final-snapshot ^
    --no-delete-automated-backups
```

To take a final snapshot, the `--final-db-snapshot-identifier` option is required and must be specified.

To skip the final snapshot, specify the `--skip-final-snapshot` option instead of the `--no-skip-final-snapshot` and `--final-db-snapshot-identifier` options in the command.

To delete automated backups, specify the `--delete-automated-backups` option instead of the `--no-delete-automated-backups` option in the command.

Starting and stopping an RDS Custom for SQL Server DB instance

You can start and stop your RDS Custom for SQL Server DB instance. The same general requirements and limitations for RDS for SQL Server DB instances apply to stopping and starting your RDS Custom for SQL Server DB instances. For more information, see Stopping an Amazon RDS DB instance temporarily.

The following considerations also apply to starting and stopping your RDS Custom for SQL Server DB instance:

- Modifying an EC2 instance attribute of an RDS Custom for SQL Server DB instance while the DB instance is STOPPED isn't supported.

- You can stop and start an RDS Custom for SQL Server DB instance only if it's configured for a single Availability Zone. You can't stop an RDS Custom for SQL Server DB instance in a Multi-AZ configuration.
- A `SYSTEM` snapshot will be created when you stop an RDS Custom for SQL Server DB instance. The snapshot will be automatically deleted when you start the RDS Custom for SQL Server DB instance again.
- If you delete your EC2 instance while your RDS Custom for SQL Server DB instance is stopped, the `C:` drive will be replaced when you start the RDS Custom for SQL Server DB instance again.
- The `C:\` drive, hostname, and your custom configurations are persisted when you stop an RDS Custom for SQL Server DB instance, as long as you don't modify the instance type.
- The following actions will result in RDS Custom placing the DB instance outside the support perimeter, and you're still charged for DB instance hours:
  - Starting the underlying EC2 instance while Amazon RDS is stopped. To resolve, you can call the `start-db-instance` Amazon RDS API, or stop the EC2 so the RDS Custom instance returns to `STOPPED`.
  - Stopping underlying EC2 instance when the RDS Custom for SQL Server DB instance is `ACTIVE`.

For more details about stopping and starting DB instances, see Stopping an Amazon RDS DB instance temporarily, and Starting an Amazon RDS DB instance that was previously stopped.

Working with Microsoft Active Directory with RDS Custom for SQL Server

RDS Custom for SQL Server allows to join your instances to a Self-Managed Active Directory (AD) or AWS Managed Microsoft AD. This is regardless of where your AD is hosted, like an On-premises data center, Amazon EC2 or with any other cloud service providers.

For authentication of users and services, you can use NTLM or Kerberos authentication on your RDS Custom for SQL Server DB instance without using intermediary domains and forest trusts. When a user tries to authenticate on your RDS Custom for SQL Server DB instance with a self joined Active Directory, requests for authentication are forwarded to a self-managed AD or AWS Managed Microsoft AD that you specify.

In the following sections, you can find information about working with Self Managed Active Directory and AWS Managed Active Directory for RDS Custom for SQL Server.

Topics

- Region and version availability
- Configure Self-Managed or On-premise AD
- Configure Microsoft Active Directory using Directory Service
- Network configuration port rules
- Network Validation
- Setting up Windows Authentication for RDS Custom for SQL Server instances
- Managing a DB instance in a Domain
- Understanding Domain membership
- Troubleshooting Active Directory

Region and version availability

RDS Custom for SQL Server supports both Self Managed AD and AWS Managed Microsoft AD using NTLM or Kerberos in all Regions where RDS Custom for SQL Server is supported. For more information, see Supported Regions and DB engines for RDS Custom.

Configure Self-Managed or On-premise AD

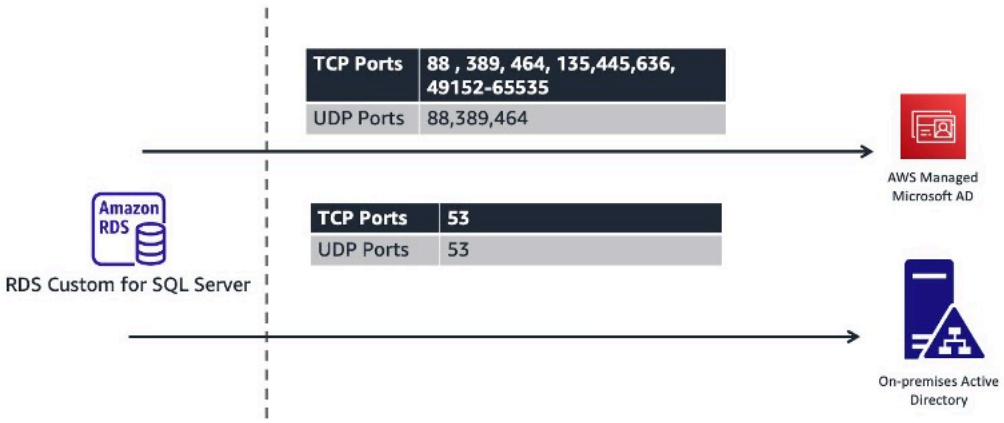
To join your on-premise or self-managed Microsoft AD to your RDS Custom for SQL Server DB instance, your Active Domain must be configured as follows:

- Define the subnets in the VPC associated with your RDS Custom for SQL Server DB instance in your self-managed or on-premises AD. Confirm there are no conflicts between the subnets in your VPC and the subnets in your AD sites.
- Your AD domain controller has a domain functional level of Windows Server 2008 R2 or higher.
- Your AD domain name can't be in Single Lable Domain (SLD) format. RDS Custom for SQL Server does not support SLD domains.
- The fully qualified domain name (FQDN) for your AD can't exceed 47 characters.

Configure your network connectivity

Configure your self-managed or on-premise AD network connectivity in the following manner:

- Set up connectivity between Amazon VPC where your RDS Custom for SQL Server instance is running, and your AD. Use Direct Connect, Site-to-Site VPN, AWS Transit Gateway, and VPC Peering.
- Allow traffic on the ports your RDS Custom for SQL Server security groups and network ACLs to your self-managed or on-premise AD. For more information, see Network configuration port rules.



Configure DNS resolution

Set up the following requirements to configure DNS resolution with self-managed or on-premises AD's:

- Configure DNS resolution within your VPC to resolve your self-hosted Active Directory's fully qualified domain name (FQDN). An example of an FQDN is `corp.example.local`. To configure DNS resolution, configure the VPC DNS resolver to forward queries for certain domains with an Amazon Route 53 outbound endpoint and resolver rule. For more information, see Configure a Route 53 Resolver outbound endpoint to resolve DNS records.
- For workloads that leverage both VPCs and on-premises resources, you must resolve DNS records hosted on-premises. On-premise resources might need to resolve names hosted on AWS.

  To create a hybrid cloud setup, use resolver endpoints and conditional forwarding riles to resolve DNS queries between your on-premise resources and custom VPC. For more information, see Resolving DNS queries between VPCs and your network in the *Amazon Route 53 Developer Guide*.

> **Important**
>
> Modifying the DNS resolver settings of the network interface on the RDS Custom for SQL Server causes DNS-enabled VPC endpoints to no longer work correctly. DNS-enabled VPC endpoints are required for instances within private subnets without internet access.

Configure Microsoft Active Directory using Directory Service

AWS Managed Microsoft AD creates a fully managed Microsoft Active Directory in AWS that is powered by Windows Server 2019 and operates at the 2012 R2 Forest and Domain functional levels. Directory Service creates the domain controllers in different subnets in an Amazon VPC, making your directory highly available even in the event of failure.

To create a directory with AWS Managed Microsoft AD, see Getting started with AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.

Configure your network connectivity

Enable cross-VPC traffic between the directory and the DB instance

To locate the directory and the DB instance in the same VPC, skip this step and move on to next step in Network configuration port rules.

To locate the directory and the DB instance in different VPCs, configure cross-VPC traffic using VPC peering or AWS Transit Gateway. For more information about using VPC peering, see What is VPC peering? in the *Amazon VPC Peering Guide* and What is AWS Transit Gateway? in the *Amazon VPC Transit Gateways*.

Enable cross-VPC traffic using VPC peering

1. Set up appropriate VPC routing rules to ensure that network traffic can flow both ways.
2. Allow the DB instance's security group to recieve inbound traffic from the directory's security group. For more information, see Network configuration port rules.
3. Network access control list (ACL) must not block traffic.

If a different AWS account owns the directory, you must share the directory. To share the directory with AWS account within which the RDS Custom for SQL Server instance is by following the Tutorial: Sharing your AWS Managed Microsoft AD for seamless EC2 domain-join in the *AWS Directory Service Administration Guide*.

Sharing a directory betweens AWS accounts

1. Sign in to the Directory Service console using the account for the DB instance and check if the domain has the SHARED status before proceeding.
2. After signing in to the Directory Service console using the account for the DB instance, note the **Directory ID** value. You use this ID to join the DB instance to the domain.

Configure DNS resolution

When you create a directory with AWS Managed Microsoft AD, Directory Service creates two domain controllers and adds the DNS service on your behalf.

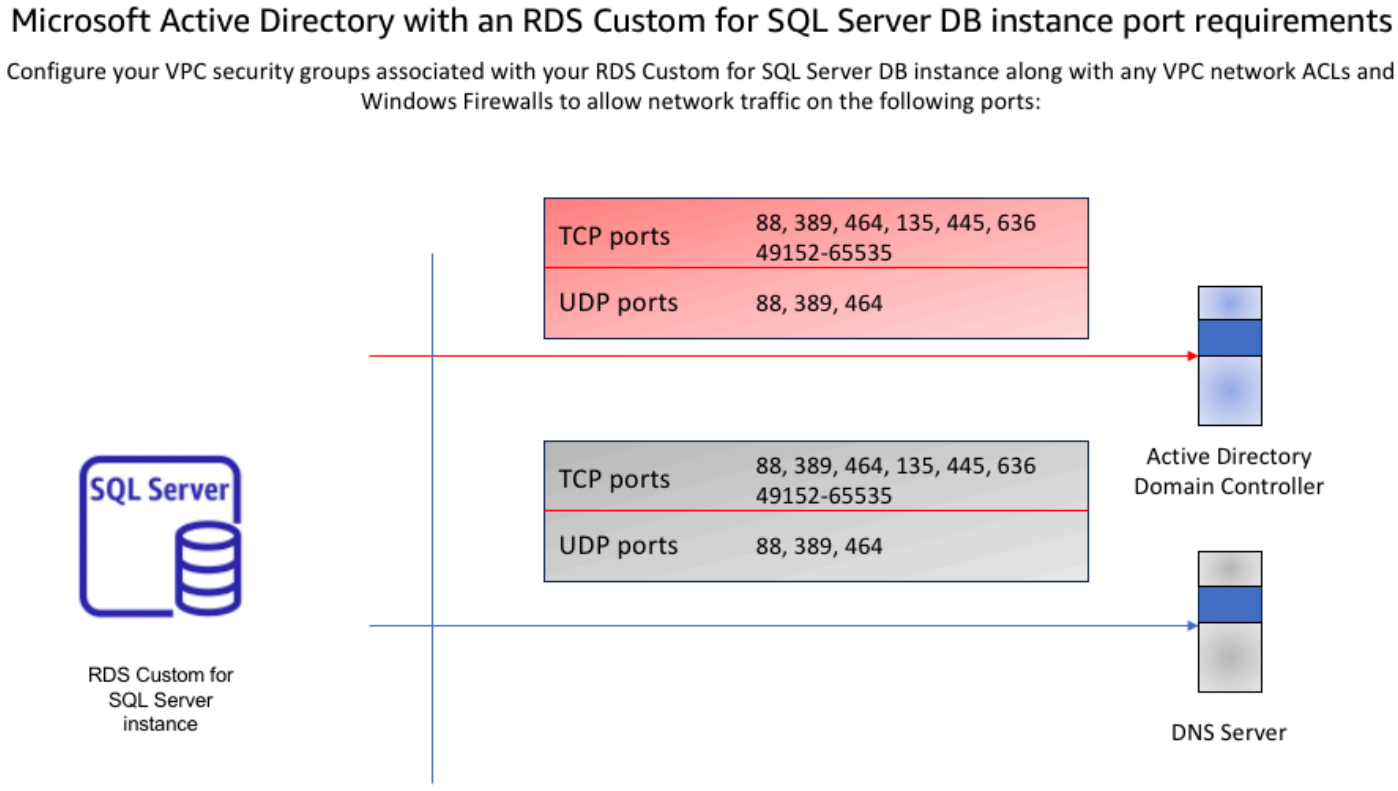
If you have an existing AWS Managed Microsoft AD or plan on launching one in a VPC other than your RDS Custom for SQL Server DB instance, configure the VPC DNS resolver to forward

queries for certain domains with a Route 53 outbound and resolver rule, see Configure a Route 53 Resolver outbound endpoint to resolve DNS records.

Network configuration port rules

Make sure that you have met the following network configurations:

- Connectivity configured between the Amazon VPC where you want to create the RDS Custom for SQL Server DB instance to either your self-managed Active Directory or AWS Managed Microsoft AD. For self-managed Active Directory, set up connectivity using AWS Direct Connect, AWS VPN, VPC peering, or AWS Transit Gateway. For AWS Managed Microsoft AD, set up connectivity using VPC peering.
- Make sure that the security group and the VPC network ACLs for the subnet(s) where you're creating your RDS Custom for SQL Server DB instance allow traffic on the ports and in the directions shown in the following diagram.



The following table identifies the role of each port.

Protocol	Ports	Role
TCP/UDP	53	Domain Name System (DNS)
TCP/UDP	88	Kerberos authentication
TCP/UDP	464	Change/Set password
TCP/UDP	389	Lightweight Directory Access Protocol (LDAP)
TCP	135	Distributed Computing Environment / End Point Mapper (DCE / EPMAP)
TCP	445	Directory Services SMB file sharing
TCP	636	Lightweight Directory Access Protocol over TLS/SSL (LDAPS)
TCP	49152 - 65535	Ephemeral ports for RPC

- Generally, the domain DNS servers are located in the AD domain controllers. You do not need to configure the VPC DHCP option set to use this feature. For more information, see DHCP option sets in the *Amazon VPC User Guide*.

Important

If you're using VPC network ACLs, you must also allow outbound traffic on dynamic ports (49152-65535) from your RDS Custom for SQL Server DB instance. Ensure that these traffic rules are also mirrored on the firewalls that apply to each of the AD domain controllers, DNS servers, and RDS Custom for SQL Server DB instances.
While VPC security groups require ports to be opened only in the direction that network traffic is initiated, most Windows firewalls and VPC network ACLs require ports to be open in both directions.

Network Validation

Before joining your RDS Custom instance to either self-managed or AWS Managed Microsoft AD, check the following from a EC2 instance in the same VPC as where you plan to launch the RDS Custom for SQL Server instance.

- Check if you are able to resolve the fully qualified domain name (FQDN) to domain controller IPs.

```
nslookup corp.example.com
```

  The command must return a similar output:

```
Server:  ip-10-0-0-2.us-west-2.compute.internal
Address:  25.0.0.2

Non-authoritative answer:
Name:    corp.example.com
Addresses:  40.0.9.25 (DC1 IP)
            40.0.50.123 (DC2 IP)
```

- Resolve AWS services from an EC2 instance in the VPC where you are launching your RDS Custom instance:

```
$region='input-your-aws-region'
$domainFQDN='input-your-domainFQDN'

function Test-DomainPorts {
    param (
        [string]$Domain,
        [array]$Ports
    )

    foreach ($portInfo in $Ports) {
        try {
            $conn = New-Object System.Net.Sockets.TcpClient
            $connectionResult = $conn.BeginConnect($Domain, $portInfo.Port, $null,
 $null)
            $success = $connectionResult.AsyncWaitHandle.WaitOne(1000) # 1 second
 timeout
            if ($success) {
                $conn.EndConnect($connectionResult)
                $result = $true
```

```
            } else {
                $result = $false
            }
        }
        catch {
            $result = $false
        }
        finally {
            if ($null -ne $conn) {
                $conn.Close()
            }
        }
        Write-Host "$($portInfo.Description) port open: $result"
    }
}

# Check if ports can be reached
$ports = @(
    @{Port = 53;   Description = "DNS"},
    @{Port = 88;   Description = "Kerberos"},
    @{Port = 389;  Description = "LDAP"},
    @{Port = 445;  Description = "SMB"},
    @{Port = 5985; Description = "WinRM"},
    @{Port = 636;  Description = "LDAPS"},
    @{Port = 3268; Description = "Global Catalog"},
    @{Port = 3269; Description = "Global Catalog over SSL"},
    @{Port = 9389; Description = "AD DS"}
)

function Test-DomainReachability {
    param (
        [string]$DomainName
    )

    try {
        $dnsResults = Resolve-DnsName -Name $DomainName -ErrorAction Stop
        Write-Host "Domain $DomainName is successfully resolving to following IP
 addresses: $($dnsResults.IpAddress)"
        Write-Host ""
        return $true
    }
    catch {
        Write-Host ""
        Write-Host "Error Message: $($_.Exception.Message)"
```

```
        Write-Host "Domain $DomainName reachability check failed, please Configure
 DNS resolution"
        return $false
    }
}

$domain = (Get-WmiObject Win32_ComputerSystem).Domain
if ($domain -eq 'WORKGROUP') {
    Write-Host ""
    Write-Host "Host $env:computername is still part of WORKGROUP and not part of any
 domain"
    }
else {
    Write-Host ""
    Write-Host "Host $env:computername is joined to $domain domain"
    Write-Host ""
    }


$isReachable = Test-DomainReachability -DomainName $domainFQDN
if ($isReachable) {
    write-Host "Checking if domain $domainFQDN is reachable on required ports  "
    Test-DomainPorts -Domain $domainFQDN -Ports $ports
}
else {
    Write-Host "Port check skipped. Domain not reachable"
}



# Get network adapter configuration
$networkConfig = Get-WmiObject Win32_NetworkAdapterConfiguration |
                 Where-Object { $_.IPEnabled -eq $true } |
                 Select-Object -First 1

# Check DNS server settings
$dnsServers = $networkConfig.DNSServerSearchOrder

if ($dnsServers) {
    Write-Host "`nDNS Server settings:"
    foreach ($server in $dnsServers) {
        Write-Host "  - $server"
    }
} else {
```

```
    Write-Host "`nNo DNS servers configured or unable to retrieve DNS server
 information."
}

write-host ""

# Checks reachability to dependent services
$services = "s3", "ec2", "secretsmanager", "logs", "events", "monitoring", "ssm",
 "ec2messages", "ssmmessages"

function Get-TcpConnectionAsync {
    param (
        $ServicePrefix,
        $region
    )
    $endpoint = "${ServicePrefix}.${region}.amazonaws.com"
    $tcp = New-Object Net.Sockets.TcpClient
    $result = $false

    try {
        $connectTask = $tcp.ConnectAsync($endpoint, 443)
        $timedOut = $connectTask.Wait(3000)
        $result = $tcp.Connected
    }
    catch {
        $result = $false
    }
    return $result
}

foreach ($service in $services) {
    $validationResult = Get-TcpConnectionAsync -ServicePrefix $service -Region
 $region
    Write-Host "Reachability to $service is $validationResult"
}
```

The `TcpTestSucceeded` value must return `True` for `s3`, `ec2`, `secretsmanager`, `logs`, `events`, `monitoring`, `ssm`, `ec2messages`, and `ssmmessages`.

Setting up Windows Authentication for RDS Custom for SQL Server instances

We recommend creating a dedicated OU and service credentials scoped to that OU for any AWS account that owns an RDS Custom for SQL Server DB instance joined to your AD domain. By dedicating an OU and service credentials, you avoid conflicting permissions and follow the principle of least privilege.

Active directory level group policies might conflict with AWS automations and permissions. We recommend selecting GPO's that apply only to the OU that you create for RDS Custom for SQL Server.

- To create OU and AD domain user in your self-managed or on-premise AD, you can connect the domain controller as a domain administrator.
- To create users and groups in an Directory Service directory, you must be connected to a management instance and you must also be logged in as a user with privileges to create users and groups. For more information, see User and group management in AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.
- To manage your Active Directory from Amazon EC2 Windows Server instance, you need to install the Active Directory domain services and Active Directory Lightweight Directory services tools on the EC2 instance. For more information, see Installing Active Directory Administration Tools for AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.
- We recommend that you install these tools on a separate EC2 instance for administration, and not on your RDS Custom for SQL Server DB instance for ease of administration.

The following are the requirements for an AD domain service account:

- You must have a service account in your AD domain with delegated permissions to join computers to the domain. A domain service account is a user account in your AD that has delegated permission to perform certain tasks.
- Delegate the following permissions to your domain service account in the Organizational Unit that you're joining your RDS Custom for SQL Server instance to:
  - Validated ability to write to the DNS host name
  - Validated ability to write to the service principal name
  - Create and delete computer objects
- For self-managed and on-premises AD, the domain service account must be a member of the "AWS Delegated Domain Name System Administrators" group.

- For AWS Managed Microsoft AD, the domain service account should be member of "DnsAdmins" group.

These are the minimum set of permissions required to join computer objects to your self-managed AD and AWS Managed Microsoft AD. For more information, see Error: Access is denied when non-administrator users who have been delegated control try to join computers to a domain controller in the Microsoft Windows Server documentation.

> **Important**
>
> Do not move computer objects that RDS Custom for SQL Server creates in the Organizational Unit (OU) after your DB instance is created. Moving associated objects might cause your RDS Custom for SQL Server DB instance to become misconfigured. If you need to move the computer objects created by Amazon RDS, use the ModifyDBInstance action to modify the domain parameters with the desired location of the computer objects.

Topics

- Step 1: Create an organizational unit (OU) in your AD
- Step 2: Create an AD domain user
- Step 3: Delegate control to the AD user in self-managed or AWS Managed Microsoft AD
- Step 4: Create a secret
- Step 5: Create or modify a RDS Custom for SQL Server DB instance
- Step 6: Create Windows Authentication SQL Server Login
- Step 7: Using Kerberos or NTLM Authentication

Step 1: Create an organizational unit (OU) in your AD

Use the following steps to create an organization unit in your AD:

Create an OU in your AD

1. Connect to your domain AD as a domain administrator.
2. Open **Active Directory Users and Computers** and select the domain where you want to create your OU.
3. Right-click the domain and choose **New**, then **Organization Unit**.

4. Enter a name for the OU.

   Enable **Protect container from accidental deletion**.
5. Choose **OK**. Your new OU appears under your domain.

For AWS Managed Microsoft AD, the name of this OU is based off the NetBIOS name you typed when you created your directory. This OU is owned by AWS and contains all of your AWS-related directory objects, which you are granted full control over. By default, two child OUs exist under this OU, namely **Computers and Users**. New OUs that RDS Custom creates are a child of the OU that is based off of the NetBIOS.

Step 2: Create an AD domain user

The domain user credentials are used for the secret in Secrets Manager.

Create an AD domain user in your AD

1. Open **Active Directory Users and Computers** and select the domain and OU where you want to create the user.
2. Right-click the **Users** object and choose **New**, then **User**.
3. Enter a first name, last name, and login name for the user. Click **Next**.
4. Enter a password for the user. Don't select **User must change password at next login** or **Account is disabled.**. Click **Next**.
5. Click **OK**. You new user appears under your domain.

Step 3: Delegate control to the AD user in self-managed or AWS Managed Microsoft AD

To delegate control to the AD domain user in your domain

1. Open **Active Directory Users and Computers** MMC snap-in and select your domain.
2. Right-click on the OU you created earlier and choose **Delegate Control**.
3. In the **Delegation Control Wizard**, click **Next**.
4. In **Users or Groups** section, click **Add**.
5. In **Select Users, Computers, or Groups**, enter the AD user you created and click **Check Names**. If your AD user check is successful, click **OK**.
6. In the **Users or Groups** section, confirm your AD user was added and click **Next**.

7. In the **Tasks to Delegate** section, choose **Create a custom task to delegate** and click **Next**.
8. In the **Active Directory Object Type** section:

   Choose **ONly the following objects in the folder**.

   Select **Computer Objects**

   Select **Create selected objects in this folder**

   Select **Delete selected objects in this folder** and click **Next**.
9. In the **Permissions** section:

   Keep **General** selected.

   Select **Validated write to DNS host name**.

   Select **Validated write to service principal name** and click **Next**.
10. In **Completing the Delegation of Control Wizard**, confirm your settings and click **Finish**.

Step 4: Create a secret

Create the secret in the same AWS account and Region that contains the RDS Custom for SQL Server DB instance that you want to include in your active directory. Store credentials of the AD domain user created in Step 2: Create an AD domain user.

Console

- In AWS Secrets Manager, choose **Store a new secret**.
- For **Secret type**, choose **Other type of secret**.
- For **Key/value pairs**, add two keys:
  - The first key, `SELF_MANAGED_ACTIVE_DIRECTORY_USERNAME` and enter the name of your AD user (without the domain prefix) for the value.
  - For the second key, enter `SELF_MANAGED_ACTIVE_DIRECTORY_PASSWORD` and enter the password for your AD user on your domain.
- For **Encryption key**, enter the same AWS KMS key you used to create RDS Custom for SQL Server instance.

- For **Secret name**, choose the secret name starting with `do-not-delete-rds-custom-` to allow your instance profile to access this secret. IF you want to choose a different name for the secret, update `RDSCustomInstanceProfile` to access **Secret name**.
- (Optional) For **Description**, enter a description for the secret name.
- Add the tags `Key="AWSRDSCustom",Value="custom-sqlserver"`
- Click **Save**, then **Next**.
- For **Configure rotation settings**, keep the default values and choose **Next**.
- Review the settings for the secret and click **Store**.
- Choose the new secret and copy the value for **Secret ARN**. We use this in the next step to set up you Active Directory.

CLI

Run the following command in your CLI to create a secret:

```
# Linux based
aws secretsmanager create-secret \
--name do-not-delete-rds-custom-DomainUserCredentails \
--description "Active directory user credentials for managing RDS Custom" \
--secret-string "{\"SELF_MANAGED_ACTIVE_DIRECTORY_USERNAME\":\"tester\",
\"SELF_MANAGED_ACTIVE_DIRECTORY_PASSWORD\":\"xxxxxxxx\"}" \
--kms-key-id <RDSCustomKMSKey> \
--tags Key="AWSRDSCustom",Value="custom-sqlserver"

# Windows based
aws secretsmanager create-secret ^
--name do-not-delete-rds-custom-DomainUserCredentails ^
--description "Active directory user credentials for managing RDS Custom" ^
--secret-string "{\"SELF_MANAGED_ACTIVE_DIRECTORY_USERNAME\":\"tester\",
\"SELF_MANAGED_ACTIVE_DIRECTORY_PASSWORD\":\"xxxxxxxx\"}" ^
--kms-key-id <RDSCustomKMSKey> ^
--tags Key="AWSRDSCustom",Value="custom-sqlserver"
```

Step 5: Create or modify a RDS Custom for SQL Server DB instance

Create or modify a RDS Custom for SQL Server DB instance for use with your directory. You can use the console, CLI, or RDS API to associate a DB instance with a directory. You can do this in one of the following ways:

- Create a new SQL Server DB instance using the console, the create-db-instance CLI command, or the CreateDBInstance RDS API operation.

  For instructions, see Creating an Amazon RDS DB instance.

- Modify an existing SQL Server DB instance using the console, the modify-db-instance CLI command, or the ModifyDBInstance RDS API operation.

  For instructions, see Modifying an Amazon RDS DB instance.

- Restore a SQL Server DB instance from a DB snapshot using the console, the restore-db-instance-from-db-snapshot CLI command, or the RestoreDBInstanceFromDBSnapshot RDS API operation.

  For instructions, see Restoring to a DB instance.

- Restore a SQL Server DB instance to a point-in-time using the console, the restore-db-instance-to-point-in-time CLI command, or the RestoreDBInstanceToPointInTime RDS API operation.

  For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

> **Note**
>
> If your RDS Custom for SQL Server instance is already joined to an AD manually, check the settings for Network configuration port rules, Network Validation, and complete steps 1 though Step 4. Update the `--domain-fqdn`, `--domain-ou`, and `--domain-auth-secret-arn` to your AD, so that domain join credentials and configurations are registered with RDS Custom to monitor, register CNAME, and take recovery actions.

When you use the AWS CLI, the following parameters are required for the DB instance to be able to use the directory that you created:

- For the `--domain-fqdn` parameter, use the fully qualified domain name of your self-managed AD.
- For the `--domain-ou` parameter, use the OU that you created in your self-managed AD.
- For the `--domain-auth-secret-arn` parameter, use the value of the **Secret ARN** that you created.

Important

If you modify a DB instance to join or remove from a self-managed AD domain or AWS Managed Microsoft AD, a reboot of the DB instance is required for the modification to take effect. You can choose to apply the changes immediately or wait until the next maintenance window. Choosing the **Apply Immediately** option causes downtime for a single-AZ DB instance. A Multi-AZ DB cluster performs a failover before completing a reboot. For more information, see Modifying an Amazon RDS DB instance.

The following CLI command creates a new RDS Custom for SQL Server DB instance and joins it to self-managed or AWS Managed Microsoft AD domain.

For Linux, macOS, or Unix:

```
aws rds create-db-instance  \
--engine custom-sqlserver-se \
--engine-version 15.00.4312.2.v1 \
--db-instance-identifier my-custom-instance \
--db-instance-class db.m5.large \
--allocated-storage 100 --storage-type io1 --iops 1000 \
--master-username my-master-username \
--master-user-password my-master-password \
--kms-key-id  my-RDSCustom-key-id \
--custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance  \
--domain-fqdn "corp.example.com" \
--domain-ou "OU=RDSCustomOU,DC=corp,DC=example,DC=com" \
--domain-auth-secret-arn "arn:aws:secretsmanager:region:account-number:secret:do-not-delete-rds-custom-my-AD-test-secret-123456" \
--db-subnet-group-name my-DB-subnet-grp \
--vpc-security-group-ids  my-securitygroup-id \
--no-publicly-accessible \
--backup-retention-period 3 \
--port 8200 \
--region us-west-2 \
--no-multi-az
```

For Windows:

```
aws rds create-db-instance  ^
```

```
--engine custom-sqlserver-se ^
--engine-version 15.00.4312.2.v1 ^
--db-instance-identifier my-custom-instance ^
--db-instance-class db.m5.large ^
--allocated-storage 100 --storage-type io1 --iops 1000 ^
--master-usernamemy-master-username ^
--master-user-password my-master-password ^
--kms-key-id  my-RDSCustom-key-id ^
--custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance  ^
--domain-fqdn "corp.example.com" ^
--domain-ou "OU=RDSCustomOU,DC=corp,DC=example,DC=com" ^
--domain-auth-secret-arn "arn:aws:secretsmanager:region:account-number:secret:do-not-
delete-rds-custom-my-AD-test-secret-123456" ^
--db-subnet-group-name my-DB-subnet-grp ^
--vpc-security-group-ids  my-securitygroup-id ^
--no-publicly-accessible ^
--backup-retention-period 3 ^
--port 8200 ^
--region us-west-2 ^
--no-multi-az
```

Important

If your NetBIOS for AWS Managed Microsoft AD is **corpexample**, then it appears as an OU itself. Any new OU created earlier will appear as a nested OU. For AWS Managed Microsoft AD, set `--domain-ou` to `"OU=RDSCustomOU,OU=corpexample,DC=corp,DC=example,DC=com"`.

The following command modifies an existing RDS Custom for SQL Server DB instance to use an Active Directory domain.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --domain-fqdn "corp.example.com" \
    --domain-ou "OU=RDSCustomOU,DC=corp,DC=example,DC=com" \
    --domain-auth-secret-arn "arn:aws:secretsmanager:region:account-number:secret:do-
not-delete-rds-custom-my-AD-test-secret-123456" \
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --domain-fqdn "corp.example.com" ^
    --domain-ou "OU=RDSCustomOU,DC=corp,DC=example,DC=com" ^
    --domain-auth-secret-arn "arn:aws:secretsmanager:region:account-number:secret:do-
not-delete-rds-custom-my-AD-test-secret-123456" ^
```

The following CLI command removes and RDS Custom for SQL Server DB instance from a Active Directory domain.

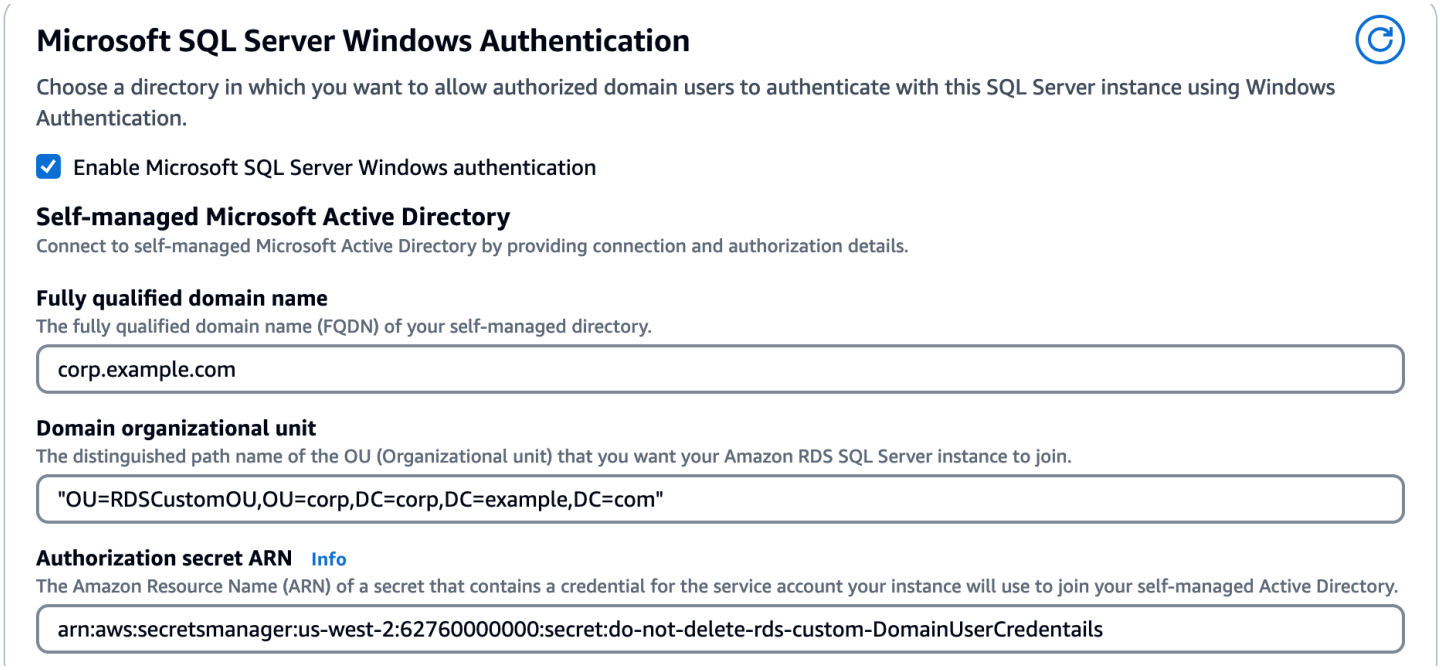
For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-custom-instance \
    --disable-domain
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-custom-instance ^
    --disable-domain
```

When using the console to create or modify your instance, click on **Enable Microsoft SQL Server Windows Authentication** to see the following options.



You are responsible to make sure your domain FQDN is resolving to the domain controller IP addresses. If domain controller IPs are not resolving, domain join operations fail but RDS Custom for SQL Server instance creation succeeds. For troubleshooting information, see Troubleshooting Active Directory.

Step 6: Create Windows Authentication SQL Server Login

Use the Amazon RDS master user credentials to connect to the SQL Server DB instance as you do for any other DB instance. Because the DB instance is joined to the AD domain, you can provision SQL Server logins and users. You do this from the AD users and groups utility in your AD domain. Database permissions are managed through standard SQL Server permissions granted and revoked to these Windows logins.

For an AD user to authenticate with SQL Server, a SQL Server Windows login must exist for the AD user or an Active Directory group that the user is a member of. Fine-grained access control is handled through granting and revoking permissions on these SQL Server logins. An AD user that doesn't have a SQL Server login or belong to an AD group with such a login can't access the SQL Server DB instance.

The `ALTER ANY LOGIN` permission is required to create an AD SQL Server login. If you haven't created any logins with this permission, connect as the DB instance's master user using SQL Server Authentication and create your AD SQL Server logins under the context of the master user.

You can run a data definition language (DDL) command such as the following to create a SQL Server login for an AD user or group.

```
USE [master]
GO
CREATE LOGIN [mydomain\myuser] FROM WINDOWS WITH DEFAULT_DATABASE = [master],
 DEFAULT_LANGUAGE = [us_english];
GO
```

Users (both humans and applications) from your domain can now connect to the RDS Custom for SQL Server instance from a domain-joined client machine using Windows authentication.

Step 7: Using Kerberos or NTLM Authentication

NTLM authentication using RDS endpoint

Each Amazon RDS DB instance has an endpoint and each endpoint has a DNS name and port number for the DB instance. To connect to your DB instance using a SQL client application, you need the DNS name and port number for your DB instance. To authenticate using NTLM authentication, you must connect to the RDS endpoint.

During planned database maintenance or unplanned service disruption, Amazon RDS automatically fails over to the up-to-date secondary database so operations can resume quickly without manual intervention. The primary and secondary instances use the same endpoint, whose physical network address transitions to the secondary as part of the failover process. You don't have to reconfigure your application when a failover occurs.

Kerberos authentication

Kerberos-based authentication for RDS Custom for SQL Server requires connections be made to a specific Service Principal Name (SPN). However, after a failover event, the application might not be aware of the new SPN. To address this, RDS Custom for SQL Server offers a Kerberos-based endpoint.

The Kerberos-based endpoint follows a specific format. If your RDS endpoint is *rds-instance-name*.*account-region-hash*.*aws-region*.rds.amazonaws.com, the corresponding Kerberos-based endpoint would be *rds-instance-name*.*account-region-hash*.*aws-region*.awsrds.*fully qualified domain name (FQDN)*.

For example, if the RDS endpoint is `ad-test.cocv6zwtircu.us-east-1.rds.amazonaws.com` and the domain name is `corp-ad.company.com`, the Kerberos-based endpoint would be `ad-test.cocv6zwtircu.us-east-1.awsrds.corp-ad.company.com`.

This Kerberos-based endpoint can be used to authenticate with the SQL Server instance using Kerberos, even after a failover event, as the endpoint is automatically updated to point to the new SPN of the primary SQL Server instance.

Finding your CNAME

To find your CNAME, connect to your domain controller and open **DNS Manager**. Navigate to **Forward Lookup Zones** and your FQDN.

Navigate through **awsrds**, **aws-region**, and **account and region specific hash**.

If you are connecting the RDS Custom EC2 instance and trying to connect to the database locally using CNAME, your connection will use NTLM authentication instead of Kerberos.

If after connecting CNAME from remote client, an NTLM connection is returned, check if required ports are allowlisted.

To check if your connection is using Kerberos, run the following query:

```
SELECT net_transport, auth_scheme
    FROM sys.dm_exec_connections
    WHERE session_id = @@SSPID;
```

Managing a DB instance in a Domain

You can use the console, AWS CLI, or the Amazon RDS API to manage your DB instance and its relationship with your domain. For example, you can move the DB instance into, out of, or between domains.

For example, using the Amazon RDS API, you can do the following:

- To reattempt a domain join for a failed membership, use the ModifyDBInstance API operation and specify the current membership's directory ID.
- To update the IAM role name for membership, use the `ModifyDBInstance` API operation and specify the current membership's directory ID and the new IAM role.
- To remove a DB instance from a domain, use the `ModifyDBInstance` API operation and specify none as the domain parameter.

- To move a DB instance from one domain to another, use the `ModifyDBInstance` API operation and specify the domain identifier of the new domain as the domain parameter.
- To list membership for each DB instance, use the DescribeDBInstances API operation.

Restoring a RDS Custom for SQL Server DB instance and adding it to an Active Directory domain

You can restore a DB snapshot or do point-in-time recovery (PITR) for a SQL Server DB instance and then add it to an Active Directory domain. Once the DB instance is restored, modify the instance using the process explained in Step 5: Create or modify a RDS Custom for SQL Server DB instance to add the DB instance to an AD domain.

Understanding Domain membership

After you create or modify your DB instance, the instance becomes a member of the domain. The AWS console indicates the status of the domain membership for the DB instance. The status of the DB instance can be one of the following:

- **joined** – The instance is a member of the domain.
- **joining** – The instance is in the process of becoming a member of the domain.
- **pending-join** – The instance membership is pending.
- **pending-maintenance-join** – AWS will attempt to make the instance a member of the domain during the next scheduled maintenance window.
- **pending-removal** – The removal of the instance from the domain is pending.
- **pending-maintenance-removal** – AWS will attempt to remove the instance from the domain during the next scheduled maintenance window.
- **failed** – A configuration problem has prevented the instance from joining the domain. Check and fix your configuration before reissuing the instance modify command.
- **removing** – The instance is being removed from the domain.

A request to become a member of a domain can fail because of a network connectivity issue or an incorrect IAM role. For example, you might create a DB instance or modify an existing instance and have the attempt fail for the DB instance to become a member of a domain. In this case, either reissue the command to create or modify the DB instance or modify the newly created instance to join the domain.

Troubleshooting Active Directory

The following are issues you might encounter when you set up or modify an AD.

Error Code	Description	Common causes	Troubleshooting suggestions
Error 2 / 0x2	The system cannot find the file specified.	The format or location for the Organizational Unit (OU) specified with the `–domain-ou` parameter is invalid. The domain service account specified via AWS Secrets Manager lack the permissions required to join the OU.	Review the `–domain-ou` parameter. Ensure the domain service account has the correct permissions to the OU.
Error 5 / 0x5	Access is denied.	Misconfigured permissions for the domain service account, or the computer account already exists in the domain.	Review the domain service account permissions in the domain, and verify that the RDS computer account is not duplicated in the domain. You can verify the name of the RDS computer account by running `SELECT @@SERVERNAME` on your RDS Custom for SQL Server DB instance. If you are using Multi-AZ, try rebooting with failover and then verify that the RDS computer account again. For more information, see Rebooting a DB instance.

Error Code	Description	Common causes	Troubleshooting suggestions
Error 87 / 0x57	The parameter is incorrect.	The domain service account specified via AWS Secrets Manager doesn't have the correct permissions. The user profile may also be corrupted.	Review the requirements for the domain service account.
Error 234 / 0xEA	Specified Organizational Unit (OU) does not exist.	The OU specified with the `–domain-ou` parameter doesn't exist in your AD.	Review the `–domain-ou` parameter and ensure the specified OU exists in your AD.
Error 1326 / 0x52E	The user name or password is incorrect.	The domain service account credentials provided in AWS Secrets Manager contains an unknown username or bad password. The domain account may also be disabled in your AD.	Ensure the credentials provided in AWS Secrets Manager are correct and the domain account is enabled in your Active Directory.
Error 1355 / 0x54B	The specified domain either does not exist or could not be contacted.	The domain is down, the specified set of DNS IPs are unreachable, or the specified FQDN is unreachable.	Review the `–domain-dns-ips` and `–domain-fqdn` parameters to ensure they're correct. Review the networking configuration of your RDS Custom for SQL Server DB instance and ensure your AD is reachable.

Error Code	Description	Common causes	Troubleshooting suggestions
Error 1722 / 0x6BA	The RPC server is unavailable.	There was an issue reaching the RPC service of your AD domain. This might be a service or network issue.	Validate that the RPC service is running on your domain controllers and that the TCP ports 135 and 49152-65535 are reachable on your domain from your RDS Custom for SQL Server DB instance.
Error 2224 / 0x8B0	The user account already exists.	The computer account that's attempting to be added to your AD already exists.	Identify the computer account by running `SELECT @@SERVERNAME` on your RDS Custom for SQL Server DB instance and then carefully remove it from your AD.
Error 2242 / 0x8c2	The password of this user has expired.	The password for the domain service account specified via AWS Secrets Manager has expired.	Update the password for the domain service account used to join your RDS Custom for SQL Server DB instance to your AD.

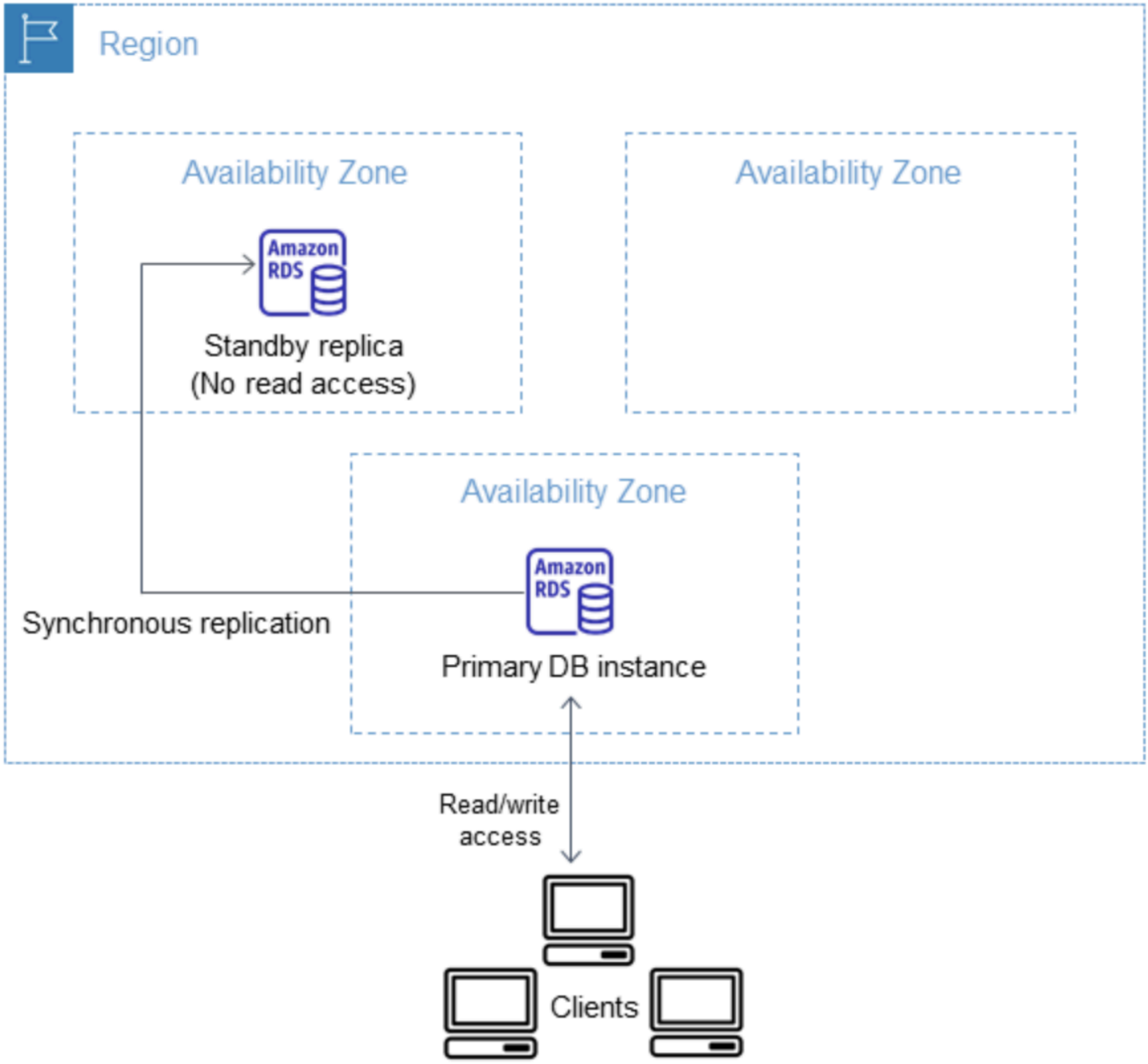
Managing a Multi-AZ deployment for RDS Custom for SQL Server

In a Multi-AZ DB instance deployment for RDS Custom for SQL Server, Amazon RDS automatically provisions and maintains a synchronous standby replica in a different Availability Zone (AZ). The primary DB instance is synchronously replicated across Availability Zones to a standby replica to provide data redundancy.

Important

A Multi-AZ deployment for RDS Custom for SQL Server is different than Multi-AZ for RDS for SQL Server. Unlike Multi-AZ for RDS for SQL Server, you must set up prerequisites for RDS Custom for SQL Server before creating your Multi-AZ DB instance because RDS Custom runs inside your own account, which requires permissions.
If you don't complete the prerequisites, your Multi-AZ DB instance might fail to run, or automatically revert to a Single-AZ DB instance. For more information about prerequisites, see Prerequisites for a Multi-AZ deployment with RDS Custom for SQL Server.

Running a DB instance with high availability can enhance availability during planned system maintenance. In the event of planned database maintenance or unplanned service disruption, Amazon RDS automatically fails over to the up-to-date secondary DB instance. This functionality lets database operations resume quickly without manual intervention. The primary and standby instances use the same endpoint, whose physical network address transitions to the secondary replica as part of the failover process. You don't have to reconfigure your application when a failover occurs.



You can create an RDS Custom for SQL Server Multi-AZ deployment by specifying Multi-AZ when creating an RDS Custom DB instance. You can use the console to convert existing RDS Custom for SQL Server DB instances to Multi-AZ deployments by modifying the DB instance and specifying the Multi-AZ option. You can also specify a Multi-AZ DB instance deployment with the AWS CLI or Amazon RDS API.

The RDS console shows the Availability Zone of the standby replica (the secondary AZ). You can also use the `describe-db-instances` CLI command or the `DescribeDBInstances` API operation to find the secondary AZ.

RDS Custom for SQL Server DB instances with Multi-AZ deployment can have increased write and commit latency compared to a Single-AZ deployment. This increase can happen because of the synchronous data replication between DB instances. You might have a change in latency if your deployment fails over to the standby replica, although AWS is engineered with low-latency network connectivity between Availability Zones.

> **Note**
>
> For production workloads, we recommend that you use a DB instance class with Provisioned IOPS (input/output operations per second) for fast, consistent performance. For more information about DB instance classes, see Requirements and limitations for Amazon RDS Custom for SQL Server.

Topics

- Region and version availability
- Limitations for a Multi-AZ deployment with RDS Custom for SQL Server
- Prerequisites for a Multi-AZ deployment with RDS Custom for SQL Server
- Creating an RDS Custom for SQL Server Multi-AZ deployment
- Modifying an RDS Custom for SQL Server Single-AZ deployment to a Multi-AZ deployment
- Modifying an RDS Custom for SQL Server Multi-AZ deployment to a Single-AZ deployment
- Failover process for an RDS Custom for SQL Server Multi-AZ deployment

Region and version availability

Multi-AZ deployments for RDS Custom for SQL Server are supported for the following SQL Server editions:

- SQL Server 2022 and 2019: Enterprise, Standard, Web, and Developer Edition

Note

Multi-AZ deployments for RDS Custom for SQL Server aren't supported on SQL Server 2019 CU8 (15.00.4073.23) or lower versions.

Multi-AZ deployments for RDS Custom for SQL Server are available in all Regions where RDS Custom for SQL Server is available. For more information on Region availability of Multi-AZ deployments for RDS Custom for SQL Server, see Supported Regions and DB engines for RDS Custom for SQL Server.

Limitations for a Multi-AZ deployment with RDS Custom for SQL Server

Multi-AZ deployments with RDS Custom for SQL Server have the following limitations:

- Cross-Region Multi-AZ deployments aren't supported.
- You can’t configure the secondary DB instance to accept database read activity.
- When you use a Custom Engine Version (CEV) with a Multi-AZ deployment, your secondary DB instance will also use the same CEV. The secondary DB instance can't use a different CEV.

Prerequisites for a Multi-AZ deployment with RDS Custom for SQL Server

If you have an existing RDS Custom for SQL Server Single-AZ deployment, the following additional prerequisites are required before modifying it to a Multi-AZ deployment. You can choose to complete the prerequisites manually or with the provided CloudFormation template. The latest CloudFormation template contains the prerequisites for both Single-AZ and Multi-AZ deployments.

Important

To simplify setup, we recommend that you use the latest CloudFormation template file provided in the network setup instructions to create the prerequisites. For more information, see Configuring with CloudFormation.

Note

When you modify an existing RDS Custom for SQL Server Single-AZ deployment to a Multi-AZ deployment, you must complete these prerequisites. If you don't complete the

prerequisites, the Multi-AZ setup will fail. To complete the prerequisites, follow the steps in Modifying an RDS Custom for SQL Server Single-AZ deployment to a Multi-AZ deployment.

- Update the RDS security group inbound and outbound rules to allow port 1120.
- Add a rule in your private network Access Control List (ACL) that allows TCP ports `0-65535` for the DB instance VPC.
- Create new Amazon SQS VPC endpoints that allow the RDS Custom for SQL Server DB instance to communicate with SQS.
- Update the SQS permissions in the instance profile role.

Creating an RDS Custom for SQL Server Multi-AZ deployment

To create an RDS Custom for SQL Server Multi-AZ deployment, follow the steps in Creating and connecting to a DB instance for Amazon RDS Custom for SQL Server.

Important

To simplify setup, we recommend that you use the latest CloudFormation template file provided in the network setup instructions. For more information, see Configuring with CloudFormation.

Creating a Multi-AZ deployment takes a few minutes to complete.

Modifying an RDS Custom for SQL Server Single-AZ deployment to a Multi-AZ deployment

You can modify an existing RDS Custom for SQL Server DB instance from a Single-AZ deployment to a Multi-AZ deployment. When you modify the DB instance,Amazon RDS performs several actions:

- Takes a snapshot of the primary DB instance.
- Creates new volumes for the standby replica from the snapshot. These volumes initialize in the background, and maximum volume performance is achieved after the data is fully initialized.
- Turns on synchronous block-level replication between the primary and secondary DB instances.

Important

We recommend that you avoid modifying your RDS Custom for SQL Server DB instance from a Single-AZ to a Multi-AZ deployment on a production DB instance during periods of peak activity.

AWS uses a snapshot to create the standby instance to avoid downtime when you convert from Single-AZ to Multi-AZ, but performance might be impacted during and after converting to Multi-AZ. This impact can be significant for workloads that are sensitive to write latency. While this capability allows large volumes to quickly be restored from snapshots, it can cause increase in the latency of I/O operations because of the synchronous replication. This latency can impact your database performance.

Note

If you created your RDS Custom for SQL Server DB instance before 29 August, 2024, patch to the latest minor version before modifying.

- For SQL Server 2019 instances, upgrade the DB engine version to `15.00.4410.1.v1` or higher.
- For SQL Server 2022 instances, upgrade the DB engine version to `16.00.4150.1.v1` or higher.

Topics

- Configuring prerequisites to modify a Single-AZ to a Multi-AZ deployment using CloudFormation
- Configuring prerequisites to modify a Single-AZ to a Multi-AZ deployment manually
- Modify using the RDS console, AWS CLI, or RDS API.

Configuring prerequisites to modify a Single-AZ to a Multi-AZ deployment using CloudFormation

To use a Multi-AZ deployment, you must ensure you've applied the latest CloudFormation template with prerequisites, or manually configure the latest prerequisites. If you've already applied the latest CloudFormation prerequisite template, you can skip these steps.

To configure the RDS Custom for SQL Server Multi-AZ deployment prerequisites using CloudFormation

1. Open the CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. To start the Create Stack wizard, select the existing stack you used to create a Single-AZ deployment and choose **Update**.

   The **Update stack** page appears.
3. For **Prerequisite - Prepare template**, choose **Replace current template**.
4. For **Specify template**, do the following:

   a. Download the latest CloudFormation template file. Open the context (right-click) menu for the link custom-sqlserver-onboard.zip and choose **Save Link As**.

   b. Save and extract the `custom-sqlserver-onboard.json` file to your computer.

   c. For **Template source**, choose **Upload a template file**.

   d. For **Choose file**, navigate to and then choose `custom-sqlserver-onboard.json`.
5. Choose **Next**.

   The **Specify stack details** page appears.
6. To keep the default options, choose **Next**.

   The **Advanced Options** page appears.
7. To keep the default options, choose **Next**.
8. To keep the default options, choose **Next**.
9. On the **Review Changes** page, do the following:

   a. For **Capabilities**, select the **I acknowledge that CloudFormation might create IAM resources with custom names** check box.

   b. Choose **Submit**.
10. Verify the update is successful. The status of a successful operation shows `UPDATE_COMPLETE`.

If the update fails, any new configuration specified in the update process will be rolled back. The existing resource will still be usable. For example, if you add network ACL rules numbered 18 and 19, but there were existing rules with same numbers, the update would return the following error: `Resource handler returned message: "The network acl entry identified by 18`

`already exists.` In this scenario you can modify the existing ACL rules to use a number lower than 18, then retry the update.

Configuring prerequisites to modify a Single-AZ to a Multi-AZ deployment manually

> **Important**
>
> To simplify setup, we recommend that you use the latest CloudFormation template file provided in the network setup instructions. For more information, see Configuring prerequisites to modify a Single-AZ to a Multi-AZ deployment using CloudFormation.

If you choose to configure the prerequisites manually, perform the following tasks.

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose **Endpoint**. The **Create Endpoint** page appears.
3. For **Service Category**, choose **AWS services**.
4. In **Services**, search for *SQS*
5. In **VPC**, choose the VPC where your RDS Custom for SQL Server DB instance is deployed.
6. In **Subnets**, choose the subnets where your RDS Custom for SQL Server DB instance is deployed.
7. In **Security Groups**, choose the *-vpc-endpoint-sg* group.
8. For **Policy**, choose **Custom**
9. In your custom policy, replace the *AWS partition*, *Region*, *accountId*,and *IAM-Instance-role* with your own values.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Condition": {
                "StringLike": {
                    "aws:ResourceTag/AWSRDSCustom": "custom-sqlserver"
                }
            },
            "Action": [
```

```
                "SQS:SendMessage",
                "SQS:ReceiveMessage",
                "SQS:DeleteMessage",
                "SQS:GetQueueUrl"
            ],
            "Resource": "arn:aws:sqs:us-east-1:111122223333:do-not-delete-
rds-custom-*",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/{IAM-Instance-role}"
            }
        }
    ]
}
```

10. Update the **Instance profile** with permission to access Amazon SQS. Replace the *AWS partition*, *Region*, and *accountId* with your own values.

```
                    {
    "Sid": "SendMessageToSQSQueue",
    "Effect": "Allow",
    "Action": [
      "SQS:SendMessage",
      "SQS:ReceiveMessage",
      "SQS:DeleteMessage",
      "SQS:GetQueueUrl"

    ],
    "Resource": [
      {
        "Fn::Sub": "arn:${AWS::Partition}:sqs:${AWS::Region}:${AWS::AccountId}:do-
not-delete-rds-custom-*"
      }
    ],
    "Condition": {
      "StringLike": {
        "aws:ResourceTag/AWSRDSCustom": "custom-sqlserver"
      }
    }
  }
                    >
```

11. Update the Amazon RDS security group inbound and outbound rules to allow port 1120.

    a. In **Security Groups**, choose the *-rds-custom-instance-sg* group.

    b. For **Inbound Rules**, create a **Custom TCP** rule to allow port *1120* from the source *-rds-custom-instance-sg* group.

    c. For **Outbound Rules**, create a **Custom TCP** rule to allow port *1120* to the destination *-rds-custom-instance-sg* group.

12. Add a rule in your private network Access Control List (ACL) that allows TCP ports `0-65535` for the source subnet of the DB instance.

    > **Note**
    >
    > When creating an **Inbound Rule** and **Outbound Rule**, take note of the highest existing **Rule number**. The new rules you create must have a **Rule number** lower than 100 and not match any existing **Rule number**.

    a. In **Network ACLs**, choose the *-private-network-acl* group.

    b. For **Inbound Rules**, create an **All TCP** rule to allow TCP ports `0-65535` with a source from *privatesubnet1* and *privatesubnet2*.

    c. For **Outbound Rules**, create an **All TCP** rule to allow TCP ports `0-65535` to destination *privatesubnet1* and *privatesubnet2*.

Modify using the RDS console, AWS CLI, or RDS API.

After you've completed the prerequisites, you can modify an RDS Custom for SQL Server DB instance from a Single-AZ to Multi-AZ deployment using the RDS console, AWS CLI, or RDS API.

Console

To modify an existing RDS Custom for SQL Server Single-AZ to Multi-AZ deployment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the Amazon RDS console, choose **Databases**.

   The **Databases** pane appears.

3. Choose the RDS Custom for SQL Server DB instance that you want to modify.
4. For **Actions**, choose **Convert to Multi-AZ deployment**.
5. On the **Confirmation** page, choose **Apply immediately** to apply the changes immediately. Choosing this option doesn't cause downtime, but there is a possible performance impact. Alternatively, you can choose to apply the update during the next maintenance window. For more information, see Using the schedule modifications setting.
6. On the **Confirmation** page, choose **Convert to Multi-AZ**.

AWS CLI

To convert to a Multi-AZ DB instance deployment by using the AWS CLI, call the modify-db-instance command and set the `--multi-az` option. Specify the DB instance identifier and the values for other options that you want to modify. For information about each option, see Settings for DB instances.

Example

The following code modifies `mycustomdbinstance` by including the `--multi-az` option. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately. For more information, see Using the schedule modifications setting.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mycustomdbinstance \
    --multi-az \
    --no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mycustomdbinstance ^
    --multi-az  \ ^
    --no-apply-immediately
```

RDS API

To convert to a Multi-AZ DB instance deployment with the RDS API, call the ModifyDBInstance operation and set the `MultiAZ` parameter to true.

Modifying an RDS Custom for SQL Server Multi-AZ deployment to a Single-AZ deployment

You can modify an existing RDS Custom for SQL Server DB instance from a Multi-AZ to a Single-AZ deployment.

Console

To modify an RDS Custom for SQL Server DB instance from a Multi-AZ to Single-AZ deployment.

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the Amazon RDS console, choose **Databases**.

   The **Databases** pane appears.
3. Choose the RDS Custom for SQL Server DB instance that you want to modify.
4. For **Multi-AZ deployment**, choose **No**.
5. On the **Confirmation** page, choose **Apply immediately** to apply the changes immediately. Choosing this option doesn't cause downtime, but there is a possible performance impact. Alternatively, you can choose to apply the update during the next maintenance window. For more information, see Using the schedule modifications setting.
6. On the **Confirmation** page, choose **Modify DB Instance**.

AWS CLI

To modify a Multi-AZ deployment to a Single-AZ deployment by using the AWS CLI, call the modify-db-instance command and include the `--no-multi-az` option. Specify the DB instance identifier and the values for other options that you want to modify. For information about each option, see Settings for DB instances.

Example

The following code modifies `mycustomdbinstance` by including the `--no-multi-az` option. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately. For more information, see Using the schedule modifications setting.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mycustomdbinstance \
    --no-multi-az  \
    --no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mycustomdbinstance ^
    --no-multi-az \ ^
    --no-apply-immediately
```

RDS API

To modify a Multi-AZ deployment to a Single-AZ deployment by using the RDS API, call the ModifyDBInstance operation and set the `MultiAZ` parameter to `false`.

Failover process for an RDS Custom for SQL Server Multi-AZ deployment

If a planned or unplanned outage of your DB instance results from an infrastructure defect, Amazon RDS automatically switches to a standby replica in another Availability Zone if you have turned on Multi-AZ. The time that it takes for the failover to complete depends on the database activity and other conditions at the time that the primary DB instance became unavailable. Failover times are typically 60 – 120 seconds. However, large transactions or a lengthy recovery process can increase failover time. When the failover is complete, it can take additional time for the RDS console to show the new Availability Zone.

> **Note**
>
> You can force a failover manually when you reboot a DB instance with failover. For more information on rebooting a DB instance, see Rebooting a DB instance

Amazon RDS handles failovers automatically so you can resume database operations as quickly as possible without administrative intervention. The primary DB instance switches over automatically to the standby replica if any of the conditions described in the following table occurs. You can view these failover reasons in the RDS event log.

Failover reason	Description
`The operating system for the RDS Custom for SQL Server Multi-AZ DB instance is being patched in an offline operation`	A failover was triggered during the maintenance window for an OS patch or a security update. For more information, see Maintaining a DB instance.
`The primary host of the RDS Custom for SQL Server Multi-AZ DB instance is unhealthy.`	The Multi-AZ DB instance deployment detected an impaired primary DB instance and failed over.
`The primary host of the RDS Custom for SQL Server Multi-AZ DB instance is unreachable due to loss of network connectivity.`	RDS monitoring detected a network reachability failure to the primary DB instance and triggered a failover.
`The RDS Custom for SQL Server Multi-AZ DB instance was modified by the customer.`	A DB instance modification triggered a failover. For more information, see Modifying an RDS Custom for SQL Server DB instance.
`The storage volume of the primary host of the RDS Custom`	The Multi-AZ DB instance deployment detected a storage issue on the primary DB instance and failed over.

Failover reason	Description
`for SQL Server Multi-AZ DB instance experienced a failure.`	
`The user requested a failover of the RDS Custom for SQL Server Multi-AZ DB instance.`	The RDS Custom for SQL Server Multi-AZ DB instance was rebooted with failover. For more information, see Rebooting a DB instance.
`The RDS Custom for SQL Server Multi-AZ primary DB instance is busy or unresponsive.`	The primary DB instance is unresponsive. We recommend that you try the following steps: • Examine the event logs and CloudWatch logs for excessive CPU, memory, or swap space usage. For more information, see Working with Amazon RDS event notification. • Create a rule that triggers on an Amazon RDS event. For more information, see Creating a rule that triggers on an Amazon RDS event. • Evaluate your workload to determine whether you're using the appropriate DB instance class. For more information, see DB instance classes.

To determine if your Multi-AZ DB instance has failed over, you can do the following:

- Set up DB event subscriptions to notify you by email or SMS that a failover has been initiated. For more information about events, see Working with Amazon RDS event notification.
- View your DB events by using the RDS console or API operations.
- View the current state of your RDS Custom for SQL Server Multi-AZ DB instance deployment by using the RDS console, CLI, or API operations.

Time to live (TTL) settings with applications using an RDS Custom for SQL Server Multi-AZ deployment

The failover mechanism automatically changes the Domain Name System (DNS) record of the DB instance to point to the standby DB instance. As a result, you need to re-establish any existing connections to your DB instance. Ensure that any DNS cache time-to-live (TTL) configuration value is low, and validate that your application will not cache DNS for an extended time. A high TTL value might prevent your application from quickly reconnecting to the DB instance after failover.

Backing up and restoring an Amazon RDS Custom for SQL Server DB instance

Like Amazon RDS, RDS Custom creates and saves automated backups of your RDS Custom for SQL Server DB instance when backup retention is enabled. You can also back up your DB instance manually. The automated backups are comprised of snapshot backups and transaction log backups. Snapshot backups are taken for the entire storage volume of DB instance during your specified backup window. Transaction log backups are taken for the PITR-eligible databases on a regular interval period. RDS Custom saves the automated backups of your DB instance according to your specified backup retention period. You can use automated backups to recover your DB instance to a point in time within the backup retention period.

You can also take snapshot backups manually. You can create a new DB instance from these snapshot backups at any time. For more information about manually creating a DB snapshot, see Creating an RDS Custom for SQL Server snapshot.

Although snapshot backups serve operationally as full backups, you are billed only for incremental storage use. The first snapshot of an RDS Custom DB instance contains the data for the full DB instance. Subsequent snapshots of the same database are incremental, which means that only the data that has changed after your most recent snapshot is saved.

Topics

- Creating an RDS Custom for SQL Server snapshot
- Restoring from an RDS Custom for SQL Server DB snapshot
- Restoring an RDS Custom for SQL Server instance to a point in time
- Deleting an RDS Custom for SQL Server snapshot
- Deleting RDS Custom for SQL Server automated backups

Creating an RDS Custom for SQL Server snapshot

RDS Custom for SQL Server creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. When you create a snapshot, specify which RDS Custom for SQL Server DB instance to back up. Give your snapshot a name so you can restore from it later.

When you create a snapshot, RDS Custom for SQL Server creates an Amazon EBS snapshot for volume (`D:`), which is the database volume attached to the DB instance. To make snapshots

easy to associate with a specific DB instance, they're tagged with `DBSnapshotIdentifier`, `DbiResourceId`, and `VolumeType`.

Creating a DB snapshot results in a brief I/O suspension. This suspension can last from a few seconds to a few minutes, depending on the size and class of your DB instance. The snapshot creation time varies with the total count and size of your databases. To learn more about the number of databases eligible for a point in time restore (PITR) operation, see Number of databases eligible for PITR per instance class type.

Because the snapshot includes the entire storage volume, the size of files, such as temporary files, also affects snapshot creation time. To learn more about creating snapshots, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Create an RDS Custom for SQL Server snapshot using the console or the AWS CLI.

Console

To create an RDS Custom snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. In the list of RDS Custom DB instances, choose the instance for which you want to take a snapshot.
4. For **Actions**, choose **Take snapshot**.

   The **Take DB snapshot** window appears.
5. For **Snapshot name**, enter the name of the snapshot.
6. Choose **Take snapshot**.

AWS CLI

You create a snapshot of an RDS Custom DB instance by using the create-db-snapshot AWS CLI command.

Specify the following options:

- `--db-instance-identifier` – Identifies which RDS Custom DB instance you are going to back up

- `--db-snapshot-identifier` – Names your RDS Custom snapshot so you can restore from it later

In this example, you create a DB snapshot called *my-custom-snapshot* for an RDS Custom DB instance called *my-custom-instance*.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-snapshot \
    --db-instance-identifier my-custom-instance \
    --db-snapshot-identifier my-custom-snapshot
```

For Windows:

```
aws rds create-db-snapshot ^
    --db-instance-identifier my-custom-instance ^
    --db-snapshot-identifier my-custom-snapshot
```

Restoring from an RDS Custom for SQL Server DB snapshot

When you restore an RDS Custom for SQL Server DB instance, you provide the name of the DB snapshot and a name for the new instance. You can't restore from a snapshot to an existing RDS Custom DB instance. A new RDS Custom for SQL Server DB instance is created when you restore.

Restoring from a snapshot will restore the storage volume to the point in time at which the snapshot was taken. This will include all the databases and any other files that were present on the `(D:)` volume.

Console

To restore an RDS Custom DB instance from a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.

5. On the **Restore DB instance** page, for **DB instance identifier**, enter the name for your restored RDS Custom DB instance.
6. Choose **Restore DB instance**.

AWS CLI

You restore an RDS Custom DB snapshot by using the restore-db-instance-from-db-snapshot AWS CLI command.

If the snapshot you are restoring from is for a private DB instance, make sure to specify both the correct `db-subnet-group-name` and `no-publicly-accessible`. Otherwise, the DB instance defaults to publicly accessible. The following options are required:

- `db-snapshot-identifier` – Identifies the snapshot from which to restore
- `db-instance-identifier` – Specifies the name of the RDS Custom DB instance to create from the DB snapshot
- `custom-iam-instance-profile` – Specifies the instance profile associated with the underlying Amazon EC2 instance of an RDS Custom DB instance.

The following code restores the snapshot named `my-custom-snapshot` for `my-custom-instance`.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \
  --db-snapshot-identifier my-custom-snapshot \
  --db-instance-identifier my-custom-instance \
  --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance \
  --no-publicly-accessible
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^
  --db-snapshot-identifier my-custom-snapshot ^
  --db-instance-identifier my-custom-instance ^
  --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance ^
  --no-publicly-accessible
```

Restoring an RDS Custom for SQL Server instance to a point in time

You can restore a DB instance to a specific point in time (PITR), creating a new DB instance. To support PITR, your DB instances must have backup retention enabled.

The latest restorable time for an RDS Custom for SQL Server DB instance depends on several factors, but is typically within 5 minutes of the current time. To see the latest restorable time for a DB instance, use the AWS CLI describe-db-instances command and look at the value returned in the `LatestRestorableTime` field for the DB instance. To see the latest restorable time for each DB instance in the Amazon RDS console, choose **Automated backups**.

You can restore to any point in time within your backup retention period. To see the earliest restorable time for each DB instance, choose **Automated backups** in the Amazon RDS console.

For general information about PITR, see Restoring a DB instance to a specified time for Amazon RDS.

Topics

- PITR considerations for RDS Custom for SQL Server
- Number of databases eligible for PITR per instance class type
- Making databases ineligible for PITR
- Transaction logs in Amazon S3
- PITR Restore using the AWS Management Console, the AWS CLI, or the RDS API.

PITR considerations for RDS Custom for SQL Server

In RDS Custom for SQL Server, PITR differs in the following important ways from PITR in Amazon RDS:

- PITR only restores the databases in the DB instance. It doesn't restore the operating system or files on the C: drive.
- For an RDS Custom for SQL Server DB instance, a database is backed up automatically and is eligible for PITR only under the following conditions:
  - The database is online.
  - Its recovery model is set to `FULL`.
  - It's writable.

  - It has its physical files on the D: drive.
  - It's not listed in the `rds_pitr_blocked_databases` table. For more information, see Making databases ineligible for PITR.
- The databases eligible for PITR are determined by the order of their database ID. RDS Custom for SQL Server allows up to 5,000 databases per DB instance. However, the maximum number of databases restored by a PITR operation for an RDS Custom for SQL Server DB instance is dependent on the instance class type. For more information, see Number of databases eligible for PITR per instance class type.

  Other databases that aren't part of PITR can be restored from DB snapshots, including the automated snapshot backups used for PITR.
- Adding a new database, renaming a database, or restoring a database that is eligible for PITR initiates a snapshot of the DB instance.
- The maximum number of databases eligible for PITR changes when the database instance goes through a scale compute operation, depending on the target instance class type. If the instance is scaled up, allowing more databases on the instance to be eligible for PITR, a new snapshot is taken.
- Restored databases have the same name as in the source DB instance. You can't specify a different name.
- `AWSRDSCustomSQLServerIamRolePolicy` requires access to other AWS services. For more information, see Add an access policy to AWSRDSCustomSQLServerInstanceRole.
- Time zone changes aren't supported for RDS Custom for SQL Server. If you change the operating system or DB instance time zone, PITR (and other automation) doesn't work.

Number of databases eligible for PITR per instance class type

The following table shows the maximum number of databases eligible for PITR based on instance class type.

Instance class type	Maximum number of PITR eligible databases				
db.*.large	100				

Instance class type	Maximum number of PITR eligible databases
db.*.xlarge to db.*.2xlarge	150
db.*.4xlarge to db.*.8xlarge	300
db.*.12xlarge to db.*.16xlarge	600
db.*.24xlarge, db.*32xlarge	1000

* *Represents different instance class types.*

The maximum number of databases eligible for PITR on a DB instance depends on the instance class type. The number ranges from 100 on the smallest to 1000 on the largest instance class types supported by RDS Custom for SQL Server. SQL server system databases (`master, model, msdb, tempdb`), aren't included in this limit. When a DB instance is scaled up or down, depending on the target instance class type, RDS Custom will automatically update the number of database eligible for PITR. RDS Custom for SQL Server will send `RDS-EVENT-0352` when the maximum number of databases eligible for PITR changes on a DB instance. For more information, see Custom engine version events.

Note

PITR support for greater than 100 databases is only available on DB instances created after August 26, 2023. For instances created before August 26, 2023, the maximum number of databases eligible for PITR is 100, regardless of the instance class. To enable PITR support for more than 100 databases on DB instances created before August 26, 2023, you can perform the following action:

- Upgrade the DB engine version to 15.00.4322.2.v1 or higher

During a PITR operation, RDS Custom will restore all of the databases that were part of PITR on source DB instance at restore time. Once the target DB instance has completed restore operations, if backup retention is enabled, the DB instance will start backing up based on the maximum number of databases eligible for PITR on target DB instance.

For example, if your DB instance runs on a `db.*.xlarge` that has 200 databases:

1. RDS Custom for SQL Server will choose the first 150 databases, ordered by their database ID, for PITR backup.
2. You modify the instance to scale up to db.*.4xlarge.
3. Once the scale compute operation is completed, RDS Custom for SQL Server will choose the first 300 databases, ordered by their database ID, for PITR backup. Each one of the 200 databases that satisfy the PITR requirement conditions will now be eligible for PITR.
4. You now modify the instance to scale down back to db.*.xlarge.
5. Once the scale compute operation is completed, RDS Custom for SQL Server will again select the first 150 databases, ordered by their database ID, for PITR backup.

Making databases ineligible for PITR

You can choose to exclude individual databases from PITR. To do this, put their `database_id` values into a `rds_pitr_blocked_databases` table. Use the following SQL script to create the table.

To create the rds_pitr_blocked_databases table

- Run the following SQL script.

```
create table msdb..rds_pitr_blocked_databases
(
database_id INT NOT NULL,
database_name SYSNAME NOT NULL,
db_entry_updated_date datetime NOT NULL DEFAULT GETDATE(),
db_entry_updated_by SYSNAME NOT NULL DEFAULT CURRENT_USER,
PRIMARY KEY (database_id)
```

```
);
```

For the list of eligible and ineligible databases, see the `RI.End` file in the `RDSCustomForSQLServer/Instances/`*`DB_instance_resource_ID`*`/TransactionLogMetadata` directory in the Amazon S3 bucket `do-not-delete-rds-custom-`*`$ACCOUNT_ID`*`-`*`$REGION`*`-`*`unique_identifier`*. For more information about the `RI.End` file, see Transaction logs in Amazon S3.

You can also determine the list of eligible databases for PITR using the following SQL script. Set the `@limit` variable to the maximum number of databases on eligible for PITR for the instance class. For more information, see Number of databases eligible for PITR per instance class type.

To determine the list of eligible databases for PITR on a DB instance class

- Run the following SQL script.

```
DECLARE @Limit INT;
SET @Limit = (insert-database-instance-limit-here);

USE msdb;
IF (EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'dbo' AND
 TABLE_NAME = 'rds_pitr_blocked_databases'))
    WITH TABLE0 AS (
        SELECT hdrs.database_id as DatabaseId, sdb.name as DatabaseName,
 'ALWAYS_ON_NOT_WRITABLE_REPLICA' as Reason, NULL as DatabaseNameOnPitrTable
        FROM sys.dm_hadr_database_replica_states hdrs
        INNER JOIN sys.databases sdb ON sdb.database_id = hdrs.database_id
        WHERE (hdrs.is_local = 1 AND hdrs.is_primary_replica = 0)
        OR (sys.fn_hadr_is_primary_replica (sdb.name) = 1 AND DATABASEPROPERTYEX
 (sdb.name, 'Updateability') = 'READ_ONLY')
    ),
    TABLE1 as (
            SELECT dbs.database_id as DatabaseId, sysdbs.name as DatabaseName,
 'OPTOUT' as Reason,
            CASE WHEN dbs.database_name = sysdbs.name THEN NULL ELSE
 dbs.database_name END AS DatabaseNameOnPitrTable
            FROM msdb.dbo.rds_pitr_blocked_databases dbs
            INNER JOIN sys.databases sysdbs ON dbs.database_id = sysdbs.database_id
            WHERE sysdbs.database_id > 4
            ),
    TABLE2 as (
```

```
            SELECT
            db.name AS DatabaseName,
            db.create_date AS CreateDate,
            db.state_desc AS DatabaseState,
            db.database_id AS DatabaseId,
            rs.database_guid AS DatabaseGuid,
            rs.last_log_backup_lsn AS LastLogBackupLSN,
            rs.recovery_fork_guid RecoveryForkGuid,
            rs.first_recovery_fork_guid AS FirstRecoveryForkGuid,
            db.recovery_model_desc AS RecoveryModel,
            db.is_auto_close_on AS IsAutoClose,
            db.is_read_only as IsReadOnly,
            NEWID() as FileName,
            CASE WHEN(db.state_desc = 'ONLINE'
                    AND db.recovery_model_desc != 'SIMPLE'
                    AND((db.is_auto_close_on = 0 and db.collation_name IS NOT NULL)
 OR db.is_auto_close_on = 1))
                    AND db.is_read_only != 1
                    AND db.user_access = 0
                    AND db.source_database_id IS NULL
                    AND db.is_in_standby != 1
                    THEN 1 ELSE 0 END AS IsPartOfSnapshot,
            CASE WHEN db.source_database_id IS NULL THEN 0 ELSE 1 END AS
 IsDatabaseSnapshot
            FROM sys.databases db
            INNER JOIN sys.database_recovery_status rs
            ON db.database_id = rs.database_id
            WHERE DB_NAME(db.database_id) NOT IN('tempdb') AND
            db.database_id NOT IN (SELECT DISTINCT DatabaseId FROM TABLE1) AND
            db.database_id NOT IN (SELECT DISTINCT DatabaseId FROM TABLE0)
        ),
        TABLE3 as(
            Select @Limit+count(DatabaseName) as TotalNumberOfDatabases from TABLE2
 where TABLE2.IsPartOfSnapshot=1 and DatabaseName in ('master','model','msdb')
        )
        SELECT TOP(SELECT TotalNumberOfDatabases from TABLE3)
 DatabaseName,CreateDate,DatabaseState,DatabaseId from TABLE2 where
 TABLE2.IsPartOfSnapshot=1
        ORDER BY TABLE2.DatabaseID ASC
ELSE
    WITH TABLE0 AS (
        SELECT hdrs.database_id as DatabaseId, sdb.name as DatabaseName,
 'ALWAYS_ON_NOT_WRITABLE_REPLICA' as Reason, NULL as DatabaseNameOnPitrTable
        FROM sys.dm_hadr_database_replica_states hdrs
```

```
       INNER JOIN sys.databases sdb ON sdb.database_id = hdrs.database_id
       WHERE (hdrs.is_local = 1 AND hdrs.is_primary_replica = 0)
       OR (sys.fn_hadr_is_primary_replica (sdb.name) = 1 AND DATABASEPROPERTYEX
(sdb.name, 'Updateability') = 'READ_ONLY')
   ),
   TABLE1 as (
           SELECT
           db.name AS DatabaseName,
           db.create_date AS CreateDate,
           db.state_desc AS DatabaseState,
           db.database_id AS DatabaseId,
           rs.database_guid AS DatabaseGuid,
           rs.last_log_backup_lsn AS LastLogBackupLSN,
           rs.recovery_fork_guid RecoveryForkGuid,
           rs.first_recovery_fork_guid AS FirstRecoveryForkGuid,
           db.recovery_model_desc AS RecoveryModel,
           db.is_auto_close_on AS IsAutoClose,
           db.is_read_only as IsReadOnly,
           NEWID() as FileName,
           CASE WHEN(db.state_desc = 'ONLINE'
                   AND db.recovery_model_desc != 'SIMPLE'
                   AND((db.is_auto_close_on = 0 and db.collation_name IS NOT NULL)
OR db.is_auto_close_on = 1))
                   AND db.is_read_only != 1
                   AND db.user_access = 0
                   AND db.source_database_id IS NULL
                   AND db.is_in_standby != 1
                   THEN 1 ELSE 0 END AS IsPartOfSnapshot,
           CASE WHEN db.source_database_id IS NULL THEN 0 ELSE 1 END AS
IsDatabaseSnapshot
           FROM sys.databases db
           INNER JOIN sys.database_recovery_status rs
           ON db.database_id = rs.database_id
           WHERE DB_NAME(db.database_id) NOT IN('tempdb') AND
           db.database_id NOT IN (SELECT DISTINCT DatabaseId FROM TABLE0)
       ),
       TABLE2 as(
           SELECT @Limit+count(DatabaseName) as TotalNumberOfDatabases from TABLE1
where TABLE1.IsPartOfSnapshot=1 and DatabaseName in ('master','model','msdb')
       )
       select top(select TotalNumberOfDatabases from TABLE2)
DatabaseName,CreateDate,DatabaseState,DatabaseId from TABLE1 where
TABLE1.IsPartOfSnapshot=1
```

```
        ORDER BY TABLE1.DatabaseID ASC
```

> **Note**
>
> The databases that are only symbolic links are also excluded from databases eligible for PITR operations. The above query doesn't filter based on this criteria.

Transaction logs in Amazon S3

The backup retention period determines whether transaction logs for RDS Custom for SQL Server DB instances are automatically extracted and uploaded to Amazon S3. A nonzero value means that automatic backups are created, and that the RDS Custom agent uploads the transaction logs to S3 every 5 minutes.

Transaction log files on S3 are encrypted at rest using the AWS KMS key that you provided when you created your DB instance. For more information, see Protecting data using server-side encryption in the *Amazon Simple Storage Service User Guide*.

The transaction logs for each database are uploaded to an S3 bucket named `do-not-delete-rds-custom-`*`$ACCOUNT_ID`*`-`*`$REGION`*`-`*`unique_identifier`*. The `RDSCustomForSQLServer/Instances/`*`DB_instance_resource_ID`* directory in the S3 bucket contains two subdirectories:

- `TransactionLogs` – Contains the transaction logs for each database and their respective metadata.

  The transaction log file name follows the pattern *`yyyyMMddHHmm`*`.`*`database_id`*`.`*`timestamp`*, for example:

  ```
  202110202230.11.1634769287
  ```

  The same file name with the suffix `_metadata` contains information about the transaction log such as log sequence numbers, database name, and `RdsChunkCount`. `RdsChunkCount` determines how many physical files represent a single transaction log file. You might see files with suffixes `_0001`, `_0002`, and so on, which mean the physical chunks of a transaction log file. If you want to use a chunked transaction log file, make sure to merge the chunks after downloading them.

Consider a scenario where you have the following files:

- `202110202230.11.1634769287`
- `202110202230.11.1634769287_0001`
- `202110202230.11.1634769287_0002`
- `202110202230.11.1634769287_metadata`

The `RdsChunkCount` is 3. The order for merging the files is the following: `202110202230.11.1634769287`, `202110202230.11.1634769287_0001`, `202110202230.11.1634769287_0002`.

- `TransactionLogMetadata` – Contains metadata information about each iteration of transaction log extraction.

  The `RI.End` file contains information for all databases that had their transaction logs extracted, and all databases that exist but didn't have their transaction logs extracted. The `RI.End` file name follows the pattern *`yyyyMMddHHmm`*`.RI.End.`*`timestamp`*, for example:

  ```
  202110202230.RI.End.1634769281
  ```

PITR Restore using the AWS Management Console, the AWS CLI, or the RDS API.

You can restore an RDS Custom for SQL Server DB instance to a point in time using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To restore an RDS Custom DB instance to a specified time

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Automated backups**.
3. Choose the RDS Custom DB instance that you want to restore.
4. For **Actions**, choose **Restore to point in time**.

   The **Restore to point in time** window appears.
5. Choose **Latest restorable time** to restore to the latest possible time, or choose **Custom** to choose a time.

If you chose **Custom**, enter the date and time to which you want to restore the instance.

Times are shown in your local time zone, which is indicated by an offset from Coordinated Universal Time (UTC). For example, UTC-5 is Eastern Standard Time/Central Daylight Time.

6. For **DB instance identifier**, enter the name of the target restored RDS Custom DB instance. The name must be unique.
7. Choose other options as needed, such as DB instance class.
8. Choose **Restore to point in time**.

AWS CLI

You restore a DB instance to a specified time by using the restore-db-instance-to-point-in-time AWS CLI command to create a new RDS Custom DB instance.

Use one of the following options to specify the backup to restore from:

- `--source-db-instance-identifier` *`mysourcedbinstance`*
- `--source-dbi-resource-id` *`dbinstanceresourceID`*
- `--source-db-instance-automated-backups-arn` *`backupARN`*

The `custom-iam-instance-profile` option is required.

The following example restores `my-custom-db-instance` to a new DB instance named `my-restored-custom-db-instance`, as of the specified time.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-to-point-in-time \
    --source-db-instance-identifier my-custom-db-instance\
    --target-db-instance-identifier my-restored-custom-db-instance \
    --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance \
    --restore-time 2022-10-14T23:45:00.000Z
```

For Windows:

```
aws rds restore-db-instance-to-point-in-time ^
```

```
    --source-db-instance-identifier my-custom-db-instance ^
    --target-db-instance-identifier my-restored-custom-db-instance ^
    --custom-iam-instance-profile AWSRDSCustomInstanceProfileForRdsCustomInstance ^
    --restore-time 2022-10-14T23:45:00.000Z
```

Deleting an RDS Custom for SQL Server snapshot

You can delete DB snapshots managed by RDS Custom for SQL Server when you no longer need them. The deletion procedure is the same for both Amazon RDS and RDS Custom DB instances.

The Amazon EBS snapshots for the binary and root volumes remain in your account for a longer time because they might be linked to some instances running in your account or to other RDS Custom for SQL Server snapshots. These EBS snapshots are automatically deleted after they're no longer related to any existing RDS Custom for SQL Server resources (DB instances or backups).

Console

To delete a snapshot of an RDS Custom DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB snapshot that you want to delete.
4. For **Actions**, choose **Delete snapshot**.
5. Choose **Delete** on the confirmation page.

AWS CLI

To delete an RDS Custom snapshot, use the AWS CLI command delete-db-snapshot.

The following option is required:

- `--db-snapshot-identifier` – The snapshot to be deleted

The following example deletes the `my-custom-snapshot` DB snapshot.

Example

For Linux, macOS, or Unix:

```
aws rds delete-db-snapshot \
  --db-snapshot-identifier my-custom-snapshot
```

For Windows:

```
aws rds delete-db-snapshot ^
  --db-snapshot-identifier my-custom-snapshot
```

Deleting RDS Custom for SQL Server automated backups

You can delete retained automated backups for RDS Custom for SQL Server when they are no longer needed. The procedure is the same as the procedure for deleting Amazon RDS backups.

Console

To delete a retained automated backup

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Automated backups**.
3. Choose **Retained**.
4. Choose the retained automated backup that you want to delete.
5. For **Actions**, choose **Delete**.
6. On the confirmation page, enter **`delete me`** and choose **Delete**.

AWS CLI

You can delete a retained automated backup by using the AWS CLI command delete-db-instance-automated-backup.

The following option is used to delete a retained automated backup:

- `--dbi-resource-id` – The resource identifier for the source RDS Custom DB instance.

  You can find the resource identifier for the source DB instance of a retained automated backup by using the AWS CLI command describe-db-instance-automated-backups.

The following example deletes the retained automated backup with source DB instance resource identifier `custom-db-123ABCEXAMPLE`.

Example

For Linux, macOS, or Unix:

```
aws rds delete-db-instance-automated-backup \
    --dbi-resource-id custom-db-123ABCEXAMPLE
```

For Windows:

```
aws rds delete-db-instance-automated-backup ^
    --dbi-resource-id custom-db-123ABCEXAMPLE
```

Copying an Amazon RDS Custom for SQL Server DB snapshot

With RDS Custom for SQL Server, you can copy automated backups and manual DB snapshots. After copying a snapshot, the copy you create is a manual snapshot. You can make multiple copies of an automated backup or manual snapshot but each copy must have a unique identifier.

You can only copy a snapshot within the same AWS account across different AWS Regions where RDS Custom for SQL Server is available. The following operations are currently not supported:

- Copying DB snapshots within the same AWS Region.
- Copying DB snapshots across AWS accounts.

RDS Custom for SQL Server supports incremental snapshot copying. For more information, see Considerations for incremental snapshot copying.

Topics

- Limitations
- Handling encryption
- Cross-Region copying
- Snapshots of DB instances created with Custom Engine Versions (CEV)
- Grant required permissions to your IAM principal
- Copying a DB snapshot

Limitations

The following limitations apply to copying a DB snapshot for RDS Custom for SQL Server:

- If you delete a source snapshot before the target snapshot becomes available, the snapshot copy might fail. Verify that the target snapshot has a status of `AVAILABLE` before you delete the source snapshot.
- You cannot specify an option group name or copy an options group in your DB snapshot copy request.
- If you delete any dependent AWS resources of the source DB snapshot before or during the copy process, your copy snapshot request could fail asynchronously.
  - If you delete the Service Master Key (SMK) backup file for your source DB instance stored in the RDS Custom managed S3 bucket in your account, the DB snapshot copy succeeds

asynchronously. However, SQL Server features dependent on SMK such as TDE enabled databases run into issues. For more information, see Troubleshooting PENDING_RECOVERY state for TDE enabled databases in RDS Custom for SQL Server.

- Copying DB snapshots within the same AWS Region is currently not supported.
- Copying DB snapshots across AWS accounts is currently not supported.

The limitations of copying a DB snapshot for Amazon RDS also apply to RDS Custom for SQL Server. For more information, see Limitations.

Handling encryption

All RDS Custom for SQL Server DB instances and DB snapshots are encrypted with KMS keys. You can only copy an encrypted snapshot to an encrypted snapshot, therefore you must specify a KMS key valid in the destination AWS Region for your DB snapshot copy request.

The source snapshot remains encrypted throughout the copy process. Amazon RDS uses envelope encryption to protect data during the copy operation with the specified destination AWS Region KMS key. For more information, see Envelope encryption in the *AWS Key Management Service Developer Guide*.

Cross-Region copying

You can copy DB snapshots across AWS Regions. However, there are certain constraints and considerations for cross-Region snapshot copying.

Authorizing RDS to communicate across AWS Regions for snapshot copying

After a cross-Region DB snapshot copy request is processed successfully, RDS starts the copy. An authorization request for RDS to access the source snapshot is created. This authorization request links the source DB snapshot to the target DB snapshot. This allows RDS to copy only to the specified target snapshot.

RDS verifies the authorization by using the `rds:CrossRegionCommunication` permission in the service-linked IAM role. If the copy is authorized, RDS can communicate with the source Region and complete the copy operation.

RDS doesn't have access to DB snapshots that weren't authorized previously by a CopyDBSnapshot request. The authorization is revoked after the copy completes.

RDS uses the service-linked role to verify the authorization in the source Region. The copy fails if you delete the service-linked role during the copy process.

For more information, see Using service-linked roles in the *AWS Identity and Access Management User Guide*.

Using AWS Security Token Service credentials

Session tokens from the global AWS Security Token Service (AWS STS) endpoint are valid only in AWS Regions that are enabled by default (commercial Regions). If you use credentials from the `assumeRole` API operation in AWS STS, use the regional endpoint if the source Region is an opt-in Region. Otherwise, the request fails. Your credentials must be valid in both Regions, which is true for opt-in Regions only when you use the regional AWS STS endpoint.

To use the global endpoint, make sure that it's enabled for both Regions in the operations. Set the global endpoint to `Valid` in all AWS Regions in the AWS STS account settings.

For more information, see Managing AWS STS in an AWS Region in the *AWS Identity and Access Management User Guide*.

Snapshots of DB instances created with Custom Engine Versions (CEV)

For a DB snapshot of a DB instance using a Custom Engine Version (CEV), RDS associates the CEV with the DB snapshot. To copy a source DB snapshot associated with a CEV across AWS Regions, RDS copies the CEV along with the source DB snapshot to the destination region.

If you are copying multiple DB snapshots associated with the same CEV to the same destination region, the first copy request copies the associated CEV. The copy process of the following requests finds the initially copied CEV and associates it with the following DB snapshot copies. The existing CEV copy must be in `AVAILABLE` state to be associated with the DB snapshot copies.

To copy a DB snapshot associated with a CEV, the requester's IAM policy must have the permissions to authorize both the DB snapshot copying and the associated CEV copying. The following permissions are needed in your requester's IAM policy to allow the associated CEV copying:

- `rds:CopyCustomDBEngineVersion` - Your requester IAM principal needs to have the permission to copy the source CEV to the target region along with the source DB snapshot. The snapshot copy request fails due to authorization errors if your requester IAM principal is not authorized to copy the source CEV.
- `ec2:CreateTags` - The underlying EC2 AMI of the source CEV is copied to the target region as a part of the CEV copy. RDS Custom attempts to tag the AMI with the `AWSRDSCustom` tag before

copying the AMI. Make sure your requester IAM principal has the permission to create the tag against the AMI underlying the source CEV in the source region.

For more information about CEV copying permissions, see Grant required permissions to your IAM principal.

Grant required permissions to your IAM principal

Make sure that you have sufficient access to copy a RDS Custom for SQL Server DB snapshot. The IAM role or user (referred to as the IAM principal) for copying a DB snapshot using the console or CLI must have either of the following policies for successful DB instance creation:

- The `AdministratorAccess` policy or,
- The `AmazonRDSFullAccess` policy with the following additional permissions:

```
s3:CreateBucket
s3:GetBucketPolicy
s3:PutBucketPolicy
kms:CreateGrant
kms:DescribeKey
ec2:CreateTags
```

RDS Custom uses these permissions during snapshot copying across AWS Regions. These permissions configure resources in your account that are required for RDS Custom operations. For more information about the `kms:CreateGrant` permission, see AWS KMS key management.

The following sample JSON policy grants the required permissions in addition to `AmazonRDSFullAccess` policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateS3BucketAndReadWriteBucketPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
```

```
                "s3:PutBucketPolicy",
                "s3:GetBucketPolicy"
            ],
            "Resource": "arn:aws:s3:::do-not-delete-rds-custom-*"
        },
        {
            "Sid": "CreateKmsGrant",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CreateEc2Tags",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "*"
        }
    ]
}
```

Note

Make sure that the listed permissions aren't restricted by service control policies (SCPs), permission boundaries, or session policies associated with the IAM principal.

If you use conditions with context keys in the requester's IAM policy, certain conditions can cause the request to fail. For more information about common pitfalls due to IAM policy conditions, see Requesting a cross-Region DB snapshot copy.

Copying a DB snapshot

Use the following procedures to copy a DB snapshot. For each AWS account, you can copy up to 20 DB snapshots at a time from one AWS Region to another. If you copy a DB snapshot to another AWS Region, you create a manual DB snapshot that is retained in that AWS Region. Copying a

DB snapshot out of the source AWS Region incurs Amazon RDS data transfer charges. For more information about data transfer pricing, see Amazon RDS pricing.

After the DB snapshot copy has been created in the new AWS Region, the DB snapshot copy behaves the same as all other DB snapshots in that AWS Region.

You can copy a DB snapshot using the AWS Management Console, AWS CLI, or the Amazon RDS API.

Console

The following procedure copies a RDS Custom for SQL Server DB snapshot by using the AWS Management Console.

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Select the RDS Custom for SQL Server DB snapshot that you want to copy.
4. In the **Actions** dropwdown , choose **Copy snapshot**.

RDS > Snapshots > Copy snapshot

Copy snapshot

Settings

Source DB snapshot
DB snapshot identifier for the snapshot being copied.
db1-snapshot

New DB snapshot identifier
DB snapshot identifier for the new snapshot.
copied-db1-snapshot
Must start with a letter and only contain letters, digits, or hyphens.

Destination Region Info
Choose a Region

Resource tags Info
Copy tags from the source DB snapshot

Depending on the amount of data to be copied and the Region you choose, this operation can take several hours to complete. The display on the progress bar might be delayed until setup is complete.

Encryption

Encryption Info
Enable Encryption
Choose to encrypt the copy of the source DB snapshot. Master key IDs and aliases appear in the list after they have been created using KMS. You cannot remove encryption from an encrypted DB snapshot.

AWS KMS key Info
Enter a key ARN

Amazon Resource Name (ARN)
[redacted]
Example: arn:aws:kms:<region>:<accountID>:key/<key-id>

Account
[redacted]

KMS key ID
[redacted]

5. To copy the DB snapshot to a different AWS Region, set **Destination Region** to the required value.

   > **Note**
   >
   > The destination AWS Region must have the same database engine version available as the source AWS Region.

6. For **New DB snapshot identifier**, enter a unique name for the DB snapshot. You can make multiple copies of an automated backup or manual snapshot but each copy must have a unique identifier.
7. (Optional) Select **Copy Tags** to copy tags and values from the snapshot to the copy of the snapshot.
8. For **Encryption**, specify the KMS key identifier to use to encrypt the DB snapshot copy.

   > **Note**
   >
   > RDS Custom for SQL Server encrypts all DB snapshots. You can't create an unencrypted DB snapshot.

9. Choose **Copy snapshot**.

RDS Custom for SQL Server creates a DB snapshot copy of your DB instance in the AWS Region of your selection.

AWS CLI

You can copy a RDS Custom for SQL Server DB snapshot by using the AWS CLI command copy-db-snapshot. If you are copying the snapshot to a new AWS Region, run the command in the new AWS Region. The following options are used to copy a DB snapshot. Not all options are required for all scenarios.

- `--source-db-snapshot-identifier` - The identifier for the source DB snapshot.
  - If the source snapshot is in a different AWS Region than the copy, specify a valid DB snapshot ARN. For example, `arn:aws:rds:us-west-2:123456789012:snapshot:instance1-snapshot-12345678`.
- `--target-db-snapshot-identifier` - The identifier for the new copy of the DB snapshot.

- `--kms-key-id` -The KMS key identifier for an encrypted DB snapshot. The KMS key identifier is the Amazon Resource Name (ARN), key identifier, or key alias for the KMS key.
  - If you copy an encrypted snapshot to a different AWS Region, then you must specify a KMS key for the destination AWS Region. KMS keys are specific to the AWS Region that they are created in and you cannot use encryption keys from one AWS Region in another AWS Region unless a multi-Region key is used. For more information on multi-Region KMS keys, see Using multi-Region keys in AWS KMS.
- `--copy-tags` - Include the tags and values from the source snapshot to the copy of the snapshot.

The following options are not supported for copying an RDS Custom for SQL Server DB snapshot:

- `--copy-option-group`
- `--option-group-name`
- `--pre-signed-url`
- `--target-custom-availability-zone`

The following code example copies an encrypted DB snapshot from the US West (Oregon) Region to the US East (N. Virginia) Region. Run the command in the destination (us-east-1) Region.

For Linux, macOS, or Unix:

```
aws rds copy-db-snapshot \
     --region us-east-1 \
    --source-db-snapshot-identifier arn:aws:rds:us-
west-2:123456789012:snapshot:instance1-snapshot-12345678 \
    --target-db-snapshot-identifier mydbsnapshotcopy \
    --kms-key-id a1b2c3d4-1234-5678-wxyz-a1b2c3d4d5e6
```

For Windows:

```
aws rds copy-db-snapshot ^
     --region us-east-1 ^
    --source-db-snapshot-identifier arn:aws:rds:us-
west-2:123456789012:snapshot:instance1-snapshot-12345678 ^
```

```
    --target-db-snapshot-identifier mydbsnapshotcopy ^
    --kms-key-id a1b2c3d4-1234-5678-wxyz-a1b2c3d4d5e6
```

RDS API

You can copy a RDS Custom for SQL Server DB snapshot by using the Amazon RDS API operation CopyDBSnapshot. If you are copying the snapshot to a new AWS Region, perform the action in the new AWS Region. The following parameters are used to copy a DB snapshot. Not all parameters are required:

- `SourceDBSnapshotIdentifier` - The identifier for the source DB snapshot.
  - If the source snapshot is in a different AWS Region than the copy, specify a valid DB snapshot ARN. For example, `arn:aws:rds:us-west-2:123456789012:snapshot:instance1-snapshot-12345678`.
- `TargetDBSnapshotIdentifier` - The identifier for the new copy of the DB snapshot.
- `KmsKeyId` - The KMS key identifier for an encrypted DB snapshot. The KMS key identifier is the Amazon Resource Name (ARN), key identifier, or key alias for the KMS key.
  - If you copy an encrypted snapshot to a different AWS Region, then you must specify a KMS key for the destination AWS Region. KMS keys are specific to the AWS Region that they are created in and you cannot use encryption keys from one AWS Region in another AWS Region unless a multi-Region key is used. For more information on multi-Region KMS keys, see Using multi-Region keys in AWS KMS.
- `CopyTags` - Set this parameter to `true` to copy tags and values from the source snapshot to the copy of the snapshot. The default is `false`.

The following options are not supported copying a RDS Custom for SQL Server DB snapshot:

- `CopyOptionGroup`
- `OptionGroupName`
- `PreSignedUrl`
- `TargetCustomAvailabilityZone`

The following code creates a copy of a snapshot, with the new name `mydbsnapshotcopy`, in the US East (N. Virginia) Region.

```
https://rds.us-east-1.amazonaws.com/
```

```
    ?Action=CopyDBSnapshot
    &KmsKeyId=a1b2c3d4-1234-5678-wxyz-a1b2c3d4d5e6
    &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Asnapshot%3Ainstance1-snapshot-12345678
    &TargetDBSnapshotIdentifier=mydbsnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
    &X-Amz-
Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

Migrating an on-premises database to Amazon RDS Custom for SQL Server

You can use the following process to migrate an on-premises Microsoft SQL Server database to Amazon RDS Custom for SQL Server using native backup and restore:

1. Take a full backup of the database on the on-premises DB instance.
2. Upload the backup file to Amazon S3.
3. Download the backup file from S3 to your RDS Custom for SQL Server DB instance.
4. Restore a database using the downloaded backup file on the RDS Custom for SQL Server DB instance.

This process explains the migration of a database from on-premises to RDS Custom for SQL Server, using native full backup and restore. To reduce the cutover time during the migration process, you might also consider using differential or log backups.

For general information about native backup and restore for RDS for SQL Server, see Importing and exporting SQL Server databases using native backup and restore.

Topics

- Prerequisites
- Backing up the on-premises database
- Uploading the backup file to Amazon S3
- Downloading the backup file from Amazon S3
- Restoring the backup file to the RDS Custom for SQL Server DB instance

Prerequisites

Perform the following tasks before migrating the database:

1. Configure Remote Desktop Connection (RDP) for your RDS Custom for SQL Server DB instance. For more information, see Connecting to your RDS Custom DB instance using RDP.
2. Configure access to Amazon S3 so you can upload and download the database backup file.

Backing up the on-premises database

You use SQL Server native backup to take a full backup of the database on the on-premises DB instance.

The following example shows a backup of a database called `mydatabase`, with the `COMPRESSION` option specified to reduce the backup file size.

To back up the on-premises database

1. Using SQL Server Management Studio (SSMS), connect to the on-premises SQL Server instance.
2. Run the following T-SQL command.

```
backup database mydatabase to
disk ='C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\Backup\mydb-full-compressed.bak'
with compression;
```

Uploading the backup file to Amazon S3

You use the AWS Management Console to upload the backup file `mydb-full-compressed.bak` to Amazon S3.

To upload the backup file to S3

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. For **Buckets**, choose the name of the bucket to which you want to upload your backup file.
3. Choose **Upload**.
4. In the **Upload** window, do one of the following:

   - Drag and drop `mydb-full-compressed.bak` to the **Upload** window.
   - Choose **Add file**, choose `mydb-full-compressed.bak`, and then choose **Open**.

   Amazon S3 uploads your backup file as an S3 object. When the upload completes, you can see a success message on the **Upload: status** page.

Downloading the backup file from Amazon S3

You use the console to download the backup file from S3 to the RDS Custom for SQL Server DB instance.

To download the backup file from S3

1. Using RDP, connect to your RDS Custom for SQL Server DB instance.
2. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
3. In the **Buckets** list, choose the name of the bucket that contains your backup file.
4. Choose the backup file `mydb-full-compressed.bak`.
5. For **Actions**, choose **Download as**.
6. Open the context (right-click) menu for the link provided, then choose **Save As**.
7. Save `mydb-full-compressed.bak` to the `D:\rdsdbdata\BACKUP` directory.

Restoring the backup file to the RDS Custom for SQL Server DB instance

You use SQL Server native restore to restore the backup file to your RDS Custom for SQL Server DB instance.

In this example, the `MOVE` option is specified because the data and log file directories are different from the on-premises DB instance.

To restore the backup file

1. Using SSMS, connect to your RDS Custom for SQL Server DB instance.
2. Run the following T-SQL command.

```
restore database mydatabase from disk='D:\rdsdbdata\BACKUP\mydb-full-
compressed.bak'
with move 'mydatabase' to 'D:\rdsdbdata\DATA\mydatabase.mdf',
move 'mydatabase_log' to 'D:\rdsdbdata\DATA\mydatabase_log.ldf';
```

RDS Custom for SQL Server Operating system updates

RDS Custom for SQL Server provides the following methods to apply operating system updates to your RDS Provided Engine Version (RPEV) instances:

- *system-update maintenance actions*
- *database minor version upgrades*
  - DB minor engine version upgrades using RPEV include up to date Operating System updates. This approach is particularly useful if you want to combine OS updates with SQL Server minor version upgrades. For more information, see Upgrading an Amazon RDS Custom for SQL Server DB instance.

Scenarios for Operating system update

There are two ways to ways to manage Operating system updates for your RDS Custom for SQL Server instances:

- For Single-AZ instances, the instance is unavailable during the Operating system update.
- For Multi-AZ deployments, RDS applies operating system updates in the following manner:
  - First, RDS performs an Operating system update on the standby instance.
  - RDS fails over to the upgraded standby DB instance, making it the new primary DB instance.
  - Lastly, RDS performs an Operating system update on the new standby DB instance.

The downtime for Multi-AZ deployments is the time it takes for the failover.

Applying Operating system updates using system-update maintenance actions

To apply Operating system updates to your Amazon RDS RPEV instances, you can use the AWS Management Console, AWS CLI, or RDS API. For more information, see Operating system updates for RDS DB instances.

Example

For Linux, macOS, or Unix:

Step 1: Check for available updates

Use the `describe-pending-maintenance-actions` command to see if OS updates are available for your instances:

```
aws rds describe-pending-maintenance-actions
```

Example response:

```
{
    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-east-1:111122223333:db:my-sqlserver-
instance",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "system-update",
                    "Description": "New Operating System update is available"
                }
            ]
        }
    ]
}
```

An action type of `system-update` indicates that an OS update is available for your instance.

Step 2: Apply the OS update

Use the `apply-pending-maintenance-action` command to schedule the update:

```
aws rds apply-pending-maintenance-action \
                --resource-identifier arn:aws:rds:us-east-1:111122223333:db:my-
sqlserver-instance \
                --apply-action system-update \
                --opt-in-type immediate
```

The `opt-in-type` input has the following options:

- `immediate`: Apply the update right away
- `next-maintenance`: Apply the update during the next scheduled maintenance window
- `undo-opt-in`: Cancel a previously scheduled update

Example response:

```
{
    "ResourcePendingMaintenanceActions": {
        "ResourceIdentifier": "arn:aws:rds:us-east-1:111122223333:db:my-sqlserver-
instance",
        "PendingMaintenanceActionDetails": [
            {
                "Action": "system-update",
                "AutoAppliedAfterDate": "2024-04-10T20:41:01.695000+00:00",
                "ForcedApplyDate": "2024-04-10T20:41:01.694000+00:00",
                "CurrentApplyDate": "2024-04-10T20:41:01.695000+00:00",
                "Description": "New Operating System update is available"
            }
        ]
    }
}
```

OS update notifications

To be notified when a new, optional operating system patch becomes available, you can subscribe to RDS-EVENT-0230 in the security patching event category. For information about subscribing to RDS events, see Subscribing to Amazon RDS event notification.

Considerations

The following consideations and limitations apply to OS updates:

- Any operating system customizations made to the C:\ drive are not preserved during Operating system updates.
- We recommend taking a manual snapshot before applying updates.

Upgrading an Amazon RDS Custom for SQL Server DB instance

You can upgrade an Amazon RDS Custom for SQL Server DB instance by modifying it to use a new DB engine version. For general information about upgrading DB instances, see Upgrading a DB instance engine version.

Topics

- Overview of upgrades in RDS Custom for SQL Server
- Upgrading major and minor engine version
- Database compatibility level

Overview of upgrades in RDS Custom for SQL Server

Amazon RDS Custom for SQL Server supports major and minor version upgrades. Minor version upgrades can include security patches, bug fixes, and engine improvements. Microsoft releases these updates as cumulative updates (CUs). Major version upgrades introduce new features and engine changes between versions, like upgrading from SQL Server 2019 to 2022. You can apply both upgrades immediately or during scheduled maintenance windows. To prevent potential backward compatibility issues, we recommend testing your applications in a non-production environment before upgrading.

RDS Custom for SQL Server allows you to upgrade an RDS Provided Engine Version (RPEV) or a Custom Engine Version (CEV).

- RDS-provided engine versions (RPEV) contain up-to-date operating system (OS) patches and SQL Server cumulative updates (CU).
- For a custom engine version (CEV), you must follow a two-step process. First, create a new CEV with your target SQL Server version, see Preparing to create a CEV for RDS Custom for SQL Server. This target version must be equal to or newer than your current version. Once the new CEV is created, modify your database instance to use this new version. For more information, see Performing a minor version upgrade for Amazon RDS Custom for SQL Server CEV with Multi-AZ.

  Do not apply SQL Server cumulative updates in-place to your running RDS Custom instance. Once you create a CEV with a specific SQL Server version (for example, SQL Server 2022 CU16), applying a newer cumulative update directly to the instance takes it out of the support perimeter and reports error `SP-S3006`. To patch an existing SQL Server instance using a CEV, create a new

CEV that includes the desired cumulative update, then modify your existing instance to switch to the new CEV.

If you upgrade an RDS Custom for SQL Server DB instance in a Multi-AZ deployment, RDS Custom for SQL Server performs rolling upgrades for your instance. This approach minimizes downtime by upgrading one instance at a time. RDS performs the following actions to perform rolling upgrades:

1. Upgrade the standby DB instance.
2. Failover to the upgraded standby DB instance, making it the new primary DB instance.
3. Upgrade the new standby DB instance.

The DB instance downtime for Multi-AZ deployments is the time it takes for the failover.

The following limitations apply when upgrading an RDS Custom for SQL Server DB instance:

- Custom DB option and parameter groups aren't supported.
- Any additional storage volumes that you attach to your RDS Custom for SQL Server DB instance are not attached after the upgrade.
- For CEVs, in-place application of SQL Server cumulative updates is not supported and results in the instance being taken out of the support perimeter.

Upgrading major and minor engine version

Both major and minor engine version upgrades are irreversible and must always be done to a newer version. To identify available target versions, use the AWS Management Console and choose from the available versions when modifying your DB instance. Alternatively, use the `describe-db-engine-versions` CLI command or DescribeDBEngineVersions RDS API command.

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
    --engine custom-sqlserver-se \
    --engine-version 15.00.4322.2.v1 \
    --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" \
    --output table
```

For Windows:

```
aws rds describe-db-engine-versions ^
    --engine custom-sqlserver-se ^
    --engine-version 15.00.4322.2.v1 ^
    --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
    --output table
```

The output shows the available target engine versions:

```
-------------------------
|DescribeDBEngineVersions|
+------------------------+
|      EngineVersion     |
+------------------------+
|  15.00.4410.1.v1       |
|  15.00.4415.2.v1       |
|  15.00.4430.1.v1       |
|  16.00.4165.4.v1       |
|  16.00.4175.1.v1       |
|  16.00.4185.3.v1       |
+------------------------+
```

After identifying your target version, use the AWS Management Console and follow the instructions in Modifying an RDS Custom for SQL Server DB instance. Alternatively, use `modify-db-instance` CLI command or ModifyDBInstance RDS API command.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier DB_INSTANCE_IDENTIFIER \
    --engine-version ENGINE_VERSION \
    --allow-major-version-upgrade \
    --region Region \
    --no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier DB_INSTANCE_IDENTIFIER ^
    --engine-version ENGINE_VERSION ^
    --allow-major-version-upgrade ^
    --region Region ^
```

```
    --no-apply-immediately
```

> **Note**
>
> You must include the `--allow-major-version-upgrade` parameter to perform major version upgrades.

Database compatibility level

You can use Microsoft SQL Server database compatibility levels to adjust some database behaviors to mimic previous versions of SQL Server. For more information, see Compatibility level in the Microsoft documentation.

When you upgrade your DB instance, all existing databases remain at their original compatibility level. For example, if you upgrade from SQL Server 2019 to SQL Server 2022, all existing databases have a compatibility level of 150. Any new database created after the upgrade have compatibility level 160.

You can change the compatibility level of a database by using the ALTER DATABASE command. For example, to change a database named `customeracct` to be compatible with SQL Server 2022, issue the following command:

```
ALTER DATABASE customeracct SET COMPATIBILITY_LEVEL = 160
```

Troubleshooting DB issues for Amazon RDS Custom for SQL Server

The shared responsibility model of RDS Custom provides OS shell–level access and database administrator access. RDS Custom runs resources in your account, unlike Amazon RDS, which runs resources in a system account. With greater access comes greater responsibility. In the following sections, you can learn how to troubleshoot issues with Amazon RDS Custom for SQL Server DB instances.

> **Note**
>
> This section explains how to troubleshoot RDS Custom for SQL Server. For troubleshooting RDS Custom for Oracle, see Troubleshooting DB issues for Amazon RDS Custom for Oracle.

Topics

- Viewing RDS Custom events
- Subscribing to RDS Custom events
- Troubleshooting CEV errors for RDS Custom for SQL Server
- Fixing unsupported configurations in RDS Custom for SQL Server
- Troubleshooting Storage-Full in RDS Custom for SQL Server
- Troubleshooting PENDING_RECOVERY state for TDE enabled databases in RDS Custom for SQL Server

Viewing RDS Custom events

The procedure for viewing events is the same for RDS Custom and Amazon RDS DB instances. For more information, see Viewing Amazon RDS events.

To view RDS Custom event notification using the AWS CLI, use the `describe-events` command. RDS Custom introduces several new events. The event categories are the same as for Amazon RDS. For the list of events, see Amazon RDS event categories and event messages.

The following example retrieves details for the events that have occurred for the specified RDS Custom DB instance.

```
aws rds describe-events \
    --source-identifier my-custom-instance \
```

```
    --source-type db-instance
```

Subscribing to RDS Custom events

The procedure for subscribing to events is the same for RDS Custom and Amazon RDS DB instances. For more information, see Subscribing to Amazon RDS event notification.

To subscribe to RDS Custom event notification using the CLI, use the `create-event-subscription` command. Include the following required parameters:

- `--subscription-name`
- `--sns-topic-arn`

The following example creates a subscription for backup and recovery events for an RDS Custom DB instance in the current AWS account. Notifications are sent to an Amazon Simple Notification Service (Amazon SNS) topic, specified by `--sns-topic-arn`.

```
aws rds create-event-subscription \
    --subscription-name my-instance-events \
    --source-type db-instance \
    --event-categories '["backup","recovery"]' \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:interesting-events
```

Troubleshooting CEV errors for RDS Custom for SQL Server

When you try to create a CEV, it might fail. In this case, RDS Custom issues the `RDS-EVENT-0198` event message. For more information on viewing RDS events, see Amazon RDS event categories and event messages.

Use the following information to help you address possible causes.

Message	Troubleshooting suggestions		
`Custom Engine Version creation expected a Sysprep'd AMI. Retry creation using a Sysprep'd AMI.`	Run Sysprep on the EC2 instance that you created from the AMI. For more information about prepping an AMI using Sysprep, see Create a standardized Amazon Machine Image (AMI) using Sysprep.		

Message	Troubleshooting suggestions
`EC2 Image permissio ns for image (AMI_ID) weren't found for customer (Customer_ID). Verify customer (Customer_ID) has valid permissions on the EC2 Image.`	Verify that your account and profile used for creation has the required permissions on `create EC2 Instance` and `Describe Images` for the selected AMI.
`Failed to rebuild databases with server collation (collation name) due to missing setup.exe file for SQL Server.`	Verify that the `setup` file is available at `C:\Program Files \Microsoft SQL Server\nnn \Setup Bootstrap\SQLnnnn\ setup.exe` .
`Image (AMI_ID) doesn't exist in your account (ACCOUNT_ID). Verify (ACCOUNT_ID) is the owner of the EC2 image.`	Ensure the AMI exists in the same customer account.
`Image id (AMI_ID) isn't valid. Specify a valid image id, and try again.`	The name of the AMI is incorrect . Ensure the correct AMI ID is provided.

Message	Troubleshooting suggestions
`Image (AMI_ID) operating system platform isn't supported. Specify a valid image, and try again.`	Choose a supported AMI that has Windows Server with SQL Server Enterprise, Standard, or Web edition. Choose an AMI with one of the following usage operation codes from the EC2 Marketplace: • RunInstances:0102 - Windows with SQL Server Enterprise • RunInstances:0006 - Windows with SQL Server Standard • RunInstances:0202 - Windows with SQL Server Web
`SQL Server Web Edition isn't supported for creating a Custom Engine Version using Bring Your Own Media. Specify a valid image, and try again.`	Use an AMI that contains a supported edition of SQL Server. For more information, see Version support for RDS Custom for SQL Server CEVs.
`The custom engine version can't be the same as the OEV engine version. Specify a valid CEV, and try again.`	Classic RDS Custom for SQL Server engine versions aren't supported . For example, version **15.00.4073.23.v1**. Use a supported version number.
`The custom engine version isn't in an active state. Specify a valid CEV, and try again.`	The CEV must be in an `AVAILABLE` state to complete the operation. Modify the CEV from `INACTIVE` to `AVAILABLE` .

Message	Troubleshooting suggestions
`The custom engine version isn't valid for an upgrade. Specify a valid CEV with an engine version greater or equal to (X), and try again.`	The target CEV is not valid. Check the requirements for a valid upgrade path.
`The custom engine version isn't valid. Names can include only lowercase letters (a-z), dashes (-), underscores (_), and periods (.). Specify a valid CEV, and try again.`	Follow the required CEV naming convention. For more information, see Requirements for RDS Custom for SQL Server CEVs.
`The custom engine version isn't valid. Specify valid database engine version, and try again. Example: 15.00.4073.23-cev123.`	An unsupported DB engine version was provided. Use a supported DB engine version.
`The expected architecture is (X) for image (AMI_ID), but architecture (Y) was found.`	Use an AMI built on the **x86_64** architecture.
`The expected owner of image (AMI_ID) is customer account ID (ACCOUNT_ID), but owner (ACCOUNT_ID) was found.`	Create the EC2 instance from the AMI that you have permission for. Run Sysprep on the EC2 instance to create and save a base image.
`The expected platform is (X) for image (AMI_ID), but platform (Y) was found.`	Use an AMI built with the Windows platform.

Message	Troubleshooting suggestions
`The expected root device type is (X) for image %s, but root device type (Y) was found.`	Create the AMI with the EBS device type.
`The expected SQL Server edition is (X), but (Y) was found.`	Choose a supported AMI that has Windows Server with SQL Server Enterprise, Standard, or Web edition. Choose an AMI with one of the following usage operation codes from the EC2 Marketplace: • RunInstances:0102 - Windows with SQL Server Enterprise • RunInstances:0006 - Windows with SQL Server Standard • RunInstances:0202 - Windows with SQL Server Web
`The expected state is (X) for image (AMI_ID), but the following state was found: (Y).`	Ensure the AMI is in a state of `AVAILABLE` .
`The provided Windows OS name (X) isn't valid. Make sure the OS is one of the following: (Y).`	Use a supported Windows OS.
`Unable to find bootstrap log file in path.`	Verify that the log file is available at `C:\Program Files\Microsoft SQL Server\nnn\Setup Bootstrap\Log\Summary.txt` .

Message	Troubleshooting suggestions
`RDS expected a Windows build version greater than or equal to (X), but found version (Y)..`	Use an AMI with a minimum OS build version of **14393**.
`RDS expected a Windows major version greater than or equal to (X), but found version (Y)..`	Use an AMI with a minimum OS major version of **10.0** or higher.

Fixing unsupported configurations in RDS Custom for SQL Server

Because of the shared responsibility model, it's your responsibility to fix configuration issues that put your RDS Custom for SQL Server DB instance into the `unsupported-configuration` state. If the issue is with the AWS infrastructure, you can use the console or the AWS CLI to fix it. If the issue is with the operating system or the database configuration, you can log in to the host to fix it.

> **Note**
>
> This section explains how to fix unsupported configurations in RDS Custom for SQL Server. For information about RDS Custom for Oracle, see Fixing unsupported configurations in RDS Custom for Oracle.

In the following tables, you can find descriptions of the notifications and events that the support perimeter sends and how to fix them. These notifications and the support perimeter are subject to change. For background on the support perimeter, see RDS Custom support perimeter. For event descriptions, see Amazon RDS event categories and event messages.

Event Code	Configuration area	RDS event message	Validation process
`SP-S0000`	Manual Unsupported Configuration	The RDS Custom DB instance status is set to [Unsuppor	To resolve this issue, create a support case.

Event Code	Configuration area	RDS event message	Validation process
		ted configuration] because of: X.	

AWS resource (infrastructure)

Event Code	Configuration area	RDS event message	Validation process
SP-S1001	EC2 Instance State	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The underlying EC2 instance %s has been stopped without stopping the RDS instance. You can resolve this by starting the underlying EC2 instance and ensuring that the binary and data volumes are attached. If your intention is to stop the RDS instance, make sure that underlying EC2 instance is in the AVAILABLE state first and then use	To check the status of a DB instance, use the console or run the following AWS CLI command: `aws rds describe-db-instances \ --db-instance-identifier db-instance-name |grep DBInstanceStatus`

Event Code	Configuration area	RDS event message	Validation process
		the RDS console or CLI to stop the RDS instance.	

SP-S1002	EC2 Instance State	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The RDS DB instance status is set to STOPPED but the underlying EC2 instance %s has been started. You can resolve this by stopping the underlying EC2 instance. If your intention is to start the RDS instance, use the console or CLI.	Use the following AWS CLI command to check the status of a DB instance:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep DBInstanceStatus`
			You can also check the status of the EC2 instance using the EC2 console.
			To start a DB instance, use the console or run the following AWS CLI command:
			`aws rds start-db-instance \ --db-instance-identifier db-instance-name`

Event Code	Configuration area	RDS event message	Validation process
SP-S1003	EC2 Instance Class	The RDS Custom DB instance status is set to [Unsupported configuration] because of: There is a mismatch between the expected and configured DB instance class of the EC2 host. You can resolve this by modifying the DB instance class to its original class type.	Use the following CLI command to check the expected DB instance class: `aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep DBInstanceClass`
SP-S1004	EBS Storage Volume Not Accessible	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The original EBS storage volume %s that was associated with the EC2 instance is currently not accessible.	

Event Code	Configuration area	RDS event message	Validation process
SP-S1005	EBS Storage Volume Detached	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The original EBS storage volume "volume-id" isn't attached. You can resolve this by attaching the EBS volume associated to the EC2 instance.	After re-attaching the EBS volume, use the following CLI commands to check if the EBS volume 'volume-id' is properly attached to the RDS instance: `aws ec2 describe-volumes \ --volume-ids volume-id | grep InstanceId`

SP-S1006	EBS Storage Volume Size	The RDS Custom DB instance status is set to [Unsupported configuration] because of: There is a mismatch between the expected and configured settings of EBS storage volume "volume-id". The volume size has been changed manually at EC2 level from its original value(s) of [%s]. To resolve this issue, create a support case.	Use the following CLI command to compare the volume size of the EBS volume 'volume-id' details and the RDS instance details:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep AllocatedStorage`
			Use the following CLI command to view the actual allocated volume size:
			`aws ec2 describe-volumes \ --volume-ids |grep Size`

Event Code	Configuration area	RDS event message	Validation process

SP-S1007	EBS Storage Volume Configuration	The RDS Custom DB instance status is set to [Unsupported configuration] because of: There is a mismatch between the expected and configured settings of EBS storage volume "volume-id". You can resolve this by modifying the EBS storage volume configuration [IOPS, Throughput, Volume type] to its original value(s) of [IOPS: %s, Throughput: %s, Volume type: %s] at the EC2 level. For future storage modifications, use the RDS console or CLI. The volume size has also been changed manually at EC2 level from its original value(s) of [%s]. To resolve this issue, create a support case.	Use the following CLI command to compare the volume type of the EBS volume 'volume-id' details and the RDS instance details. Make sure that the values at the EBS level matches the values at the RDS level:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep StorageType`
			To get the expected value for Storage Throughput at the RDS level:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep StorageThroughput`
			To get the expected value for Volume IOPS at the RDS level:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep Iops`
			To get the current Storage Type at the EC2 Level:

Event Code	Configuration area	RDS event message	Validation process

			`aws ec2 describe-volumes \ --volume-ids |grep VolumeType`
			
			To get the current value for Storage Throughput at the EC2 Level:
			
			`aws ec2 describe-volumes \ --volume-ids |grep Throughput`
			
			To get the current value for Volume IOPS at the EC2 Level:
			
			`aws ec2 describe-volumes \ --volume-ids |grep Iops`

Event Code	Configuration area	RDS event message	Validation process

SP-S1008	EBS Storage Volume Size and Configuration	The RDS Custom DB instance status is set to [Unsupported configuration] because of: There is a mismatch between the expected and configured settings of EBS storage volume "volume-id". You can resolve this by modifying the EBS storage volume configuration [IOPS, Throughput, Volume type] to its original value(s) of [IOPS: %s, Throughput: %s, Volume type: %s] at the EC2 level. For future storage modifications, use the RDS console or CLI. The volume size has also been changed manually at EC2 level from its original value(s) of [%s]. To resolve this issue, create a support case.	Use the following CLI command to compare the volume type of the EBS volume 'volume-id' details and the RDS instance details. Make sure that the values at the EBS level matches the values at the RDS level:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep StorageType`
			To get the expected value for Storage Throughput at the RDS level:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep StorageThroughput`
			To get the expected value for Volume IOPS at the RDS level:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep Iops`
			To get the current Storage Type at the EC2 Level:

Event Code	Configuration area	RDS event message	Validation process

			`aws ec2 describe-volumes \ --volume-ids |grep VolumeType`
			
			To get the current value for Storage Throughput at the EC2 Level:
			
			`aws ec2 describe-volumes \ --volume-ids |grep Throughput`
			
			To get the current value for Volume IOPS at the EC2 Level:
			
			`aws ec2 describe-volumes \ --volume-ids |grep Iops`
			
			To get the expected Allocated Volume Size:
			
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name | grep AllocatedStorage`
			
			To get the actual Allocated Volume Size:
			
			`aws ec2 describe-volumes \ --volume-ids |grep Size`

Event Code	Configuration area	RDS event message	Validation process
SP-S1009	SQS Permissions	The RDS Custom DB instance status is set to [Unsupported configuration] because of: Amazon Simple Queue Service (SQS) permissions are missing for the IAM instance profile. You can resolve this by making sure the IAM profile associated with the host has the following permissions: ["SQS:SendMessage","SQS:ReceiveMessage","SQS:DeleteMessage","SQS:GetQueueUrl"].	

Event Code	Configuration area	RDS event message	Validation process
SP-S1010	SQS VPC Endpoint	The RDS Custom DB instance status is set to [Unsupported configuration] because of: A VPC endpoint policy is blocking the Amazon Simple Queue Service (SQS) operations. You can resolve this by modifying your VPC endpoint policy to allow the required SQS actions.	

Event Code	Configuration area	RDS event message	Validation process
SP-S1011	Event bus policy	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The resource-based policy for your event bus arn:aws:events:region-1:123456789012:event-bus/default denies Amazon CloudWatch events:PutEvents actions. Resolve this by modifying your resource-based policy to allow events:PutEvents actions for EventBus %s.	

Event Code	Configuration area	RDS event message	Validation process
SP-S1012	CloudWatch VPC permissions	The RDS Custom DB instance status is set to [Unsupported configuration] because of: A VPC endpoint policy is missing permissions to access Amazon CloudWatch events. Resolve this by modifying your VPC endpoint policy to allow events:PutEvents on EventBus arn:aws:events:region-1:123456789012:event-bus/default.	

Event Code	Configuration area	RDS event message	Validation process
SP-S1013	Service control policy	The RDS Custom DB instance status is set to [Unsupported configuration] because of: A service control policy in your AWS Organizations is missing permissions to access Amazon CloudWatch events. Resolve this by modifying your service control policy to allow events:PutEvents on EventBus arn:aws:events:region-1:123456789012:event-bus/default.	

Event Code	Configuration area	RDS event message	Validation process
SP-S1014	IAM instance profile	The RDS Custom DB instance status is set to [Unsupported configuration] because of: Your IAM instance profile %s permissions deny Amazon CloudWatch events. Resolve this by setting ["events:PutEvents"] to 'Allow' and allowing events:PutEvents on EventBus arn:aws:events:region-1:123456789012:event-bus/default in your IAM profile associated with the instance.	

Event Code	Configuration area	RDS event message	Validation process
SP-S1015	IAM instance profile	The RDS Custom DB instance status is set to [Unsupported configuration] because of: Your IAM instance profile %s is missing Amazon CloudWatch event permissions. Resolve this by including the ["events:PutEvents"] permissions and allowing events:PutEvents on EventBus arn:aws:events:region-1:123456789012:event-bus/default in your IAM profile associated with the instance.	

Event Code	Configuration area	RDS event message	Validation process
SP-S1016	IAM permissions boundary	The RDS Custom DB instance status is set to [Unsupported configuration] because of: Your IAM instance profile %s has permissions boundary that deny Amazon CloudWatch events. Resolve this by setting ["events:PutEvents"] to 'Allow' for the EventBus arn:aws:events:region-1:123456789012:event-bus/default in your IAM instance profile permissions boundary.	

Operating system

Event Code	Configuration area	RDS event message	Validation process
SP-S2001	SQL Service Status	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The SQL Server service isn't started. You	

Event Code	Configuration area	RDS event message	Validation process
		can resolve this by restarting the SQL Server service on the host. If this DB instance is a Multi-AZ DB instance and restart fails, then stop and start the host to initiate a failover.	

Event Code	Configuration area	RDS event message	Validation process

SP-S2002	RDS Custom Agent Status	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The RDS Custom Agent service isn't installed or couldn't be started. You can resolve this by reviewing the Windows Event Log to determine why the service won't start, and take appropriate steps to fix the issue. For additional assistance, create a support case.	Log in to the host and make sure that the RDS Custom agent is running.
			
			You can use the following commands to view the agent status.
			
			`$name = "RDSCustomAgent"`
			`$service = Get-Service $name`
			`Write-Host $service.Status`
			
			If the status isn't Running, you can start the service with the following command:
			
			`Start-Service $name`
			
			If the agent can't start, check the Windows Events to see why it can't start. The agent requires a Windows user to start the service. Ensure a Windows user exists and has privileges to run the service.

Event Code	Configuration area	RDS event message	Validation process
SP-S2003	SSM Agent Status	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The Amazon SSM Agent service is unreachable. You can troubleshoot this by checking the service status with the `Get-Service AmazonSSMAgent` PowerShell command, or starting the service with `Start-Service AmazonSSMAgent`. Ensure that HTTPS (port 443) outbound traffic to the **ssm**, **ssmmessages**, and **ec2messages** regional endpoints is allowed.	For more information, see Troubleshooting SSM Agent. To troubleshoot SSM endpoints, see Unable to connect to SSM endpoints and Use ssm-cli to troubleshoot managed node availability.

Event Code	Configuration area	RDS event message	Validation process
SP-S2004	RDS Custom Agent Login	The RDS Custom DB instance status is set to [Unsupported configuration] because of: An unexpected issue occurred with the SQL login "$HOSTNAME/RDSAgent" . To resolve this issue, create a support case.	

Event Code	Configuration area	RDS event message	Validation process
SP-S2005	Timezone	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The timezone on the Amazon EC2 Instance [%s] was changed. You can resolve this by modifying the time zone back to the setting specified during instance creation. If you would like to create an instance with a specific timezone, see the RDS Custom documentation.	Run the `Get-Timezone` PowerShell command to confirm the timezone. For more information, see Local time zone for RDS Custom for SQL Server DB instances.

Event Code	Configuration area	RDS event message	Validation process
SP-S2006	High Availability Software Solution Version	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The high availability software solution of the current instance is different from the expected version. To resolve this issue, create a support case.	

Event Code	Configuration area	RDS event message	Validation process
SP-S2007	High Availability Software Solution Configuration	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The configuration settings of the high availability software solution have been modified to unexpected values on the instance %s. To fix this issue, reboot the EC2 instance. When you reboot the EC2 instance, it automatically updates the settings to the required configuration for the high availability software solution.	

Event Code	Configuration area	RDS event message	Validation process

SP-S2008	SQL Server Service	The RDS Custom DB instance is set to [Unsupported configuration]: SQLServer (MSSQLServer) service doesn't exist on the host. To resolve this, create a support case.	You can use the following commands to view the agent status.
			`$name = "MSSQLServer"`
			`$service = Get-Service $name`
			`Write-Host $service.Status`

SP-S2009	SSL Certificate	The RDS Custom DB instance is set to [Unsupported configuration] because of: Non self-signed SSL certificate(s) causing disruption in RDS. To resolve this issue, remove the non self-signed certificate(s) from the trusted root certificate store.	Run the following PowerShell command to review non self-signed certificate(s). `Get-ChildItem cert:\LocalMachine\root -Recurse | Where-Object {$_.Issuer -ne $_.Subject -and $_.Issuer -notlike "*RDSCustomAgentCA*"}` For more information, see HTTP Error 403.16 when you try to access a website that's hosted on IIS.

Event Code	Configuration area	RDS event message	Validation process
SP-S2010	Root Volume Storage Status	The RDS Custom DB instance is set to [Unsupported configuration] because of: Root volume storage is full. To resolve this issue, free up at least 500 MiB of storage space in the root EBS volume "volume-id" or increase the volume size and resize the C drive on the EC2 instance "instance-id". The root volume size changes do not persist when you replace the EC2 instance.	Use the following command to view available storage on the root (C:) volume. `(Get-PSDrive -Name C).Free / 1MB` For more information on modifying the EBS root volume, see How

Database

Event Code	Configuration area	RDS event message	Validation process
SP-S3001	SQL Server Shared Memory Protocol	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The SQL Server shared	You can validate this by checking: **SQL Server Configuration Manager > SQL Server Network Configuration > Protocols for MSSQLSERVER> Shared Memory** as Enabled. After you enable the

Event Code	Configuration area	RDS event message	Validation process
		memory protocol is disabled. You can resolve this by enabling the shared memory protocol in SQL Server Configuration Manager.	protocol, restart the SQL Server process.
SP-S3002	Service Master Key	The RDS Custom DB instance status is set to [Unsupported configuration] because of: RDS Automation is unable to take the backup of Service Master Key (SMK) as part of the new SMK generation. To resolve this issue, create a support case.	

Event Code	Configuration area	RDS event message	Validation process
SP-S3003	Service Master Key	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The metadata related to the Service Master Key (SMK) is missing or incomplete. To resolve this issue, create a support case.	

Event Code	Configuration area	RDS event message	Validation process

SP-S3004	DB Engine Version and Edition	The RDS Custom DB instance status is set to [Unsupported configuration] because of: There is a mismatch between the expected and installed SQL Server version and edition: Modifying the SQL Server edition is not supported on RDS Custom for SQL Server. Also, manually changing the SQL Server version on the RDS Custom EC2 instance is not supported. To resolve this issue, create a support case.	Run the following query to get the SQL version:
			`select @@version`
			Run the following AWS CLI command to get the RDS SQL engine version and edition:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name |grep EngineVersion`
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name |grep Engine`
			For more information, see Modifying an RDS Custom for SQL Server DB instance and Upgrading a DB instance engine version.

Event Code	Configuration area	RDS event message	Validation process

SP-S3005	DB Engine Edition	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The current SQL Server edition doesn't match the expected SQL Server edition [%s]: Modifying the SQL Server edition is not supported on RDS Custom for SQL Server. To resolve this issue, create a support case.	Run the following query to get the SQL edition:
			Example
			`select @@version`
			Run the following AWS CLI command to get the RDS SQL engine edition:
			`aws rds describe-db-instances \ --db-instance-identifier db-instance-name |grep Engine`

Event Code	Configuration area	RDS event message	Validation process

SP-S3006	DB Engine Version	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The current SQL Server version doesn't match the expected SQL Server version [%s]: You can't manually change the SQL Server version on the RDS Custom EC2 instance. To resolve this issue, create a support case. For any future modifications to SQL Server version, you can modify the instance from the AWS RDS console or through the modify-db-instance CLI command.	Run the following query to get the SQL version:
			Example
			`select @@version`
			Run the following AWS CLI command to get the RDS SQL engine version:
			`aws rds describe-db-instances \ --db-instance-identifier *db-instance-name* |grep EngineVersion`
			For more information, see Modifying an RDS Custom for SQL Server DB instance and Upgrading a DB instance engine version.

Event Code	Configuration area	RDS event message	Validation process

SP-S3007	Database file location	The RDS Custom DB instance status is set to [Unsupported configuration] because of: Database files are configured outside of the D:\ drive. You can resolve this by making sure that all database files, including ROW, LOG, FILESTREAM, etc... are stored on the D:\ drive.	Run the following query to list the location of database files that aren't in the default path:
			`USE master;`
			`SELECT physical_name as files_not_in_default_path`
			`FROM sys.master_files`
			`WHERE SUBSTRING(physical_name,1,3)!='D:\';`

SP-S3008	Database Count Limit Exceeded	The RDS Custom DB instance status is set to [Unsupported configuration] because of: The total number of databases on the DB instance exceeds the maximum limit of 5000. To resolve this, reduce the number of databases below the maximum supported limit.	Use the following command to view total database count:
			`SELECT COUNT(name) as databaseCount`
			`FROM sys.databases`
			`WHERE name not in ('tempdb','master','model','msdb','DWDiagnostics','DWConfiguration','DWQueue');`

Troubleshooting `Storage-Full` in RDS Custom for SQL Server

RDS Custom also monitors the root (C:) volume. The RDS Custom for SQL Server DB instance moves to the `unsupported-configuration` state when the root volume has less than 500 MiB disk space available. See `Event SP-S2010` in Fixing unsupported configurations in RDS Custom for SQL Server.

Troubleshooting PENDING_RECOVERY state for TDE enabled databases in RDS Custom for SQL Server

SQL Server databases with transparent data encryption (TDE) enabled might remain in `PENDING_RECOVERY` state if the automatic decryption runs into issues. This typically occurs after a DB instance restore if the source DB instance Service Master Key (SMK) backup file stored in the RDS Custom managed S3 bucket in your account has been deleted prior to the restore completion.

To enable the automatic decryption and bring the TDE enabled databases online, you need to open the Database Master Key (DMK) with its password and ecrypt the DMK using the SMK.

Use the following SQL Server commands for reference:

```
-- Identify PENDING_RECOVERY TDE databases
USE MASTER;
GO
SELECT name, is_encrypted, state_desc FROM sys.databases;
GO

-- Open DMK using password
OPEN MASTER KEY DECRYPTION BY PASSWORD = '<password>';
GO

-- Encrypt DMK using SMK
ALTER MASTER KEY ADD ENCRYPTION BY SERVICE MASTER KEY;
GO

-- Close SMK
CLOSE MASTER KEY;
GO

-- Bring the TDE databases online
ALTER DATABASE <database_name> SET ONLINE;
GO
```

```
-- Verify TDE databases are now in ONLINE state
SELECT name, is_encrypted, state_desc FROM sys.databases;
GO
```

Amazon RDS on AWS Outposts

Amazon RDS on AWS Outposts extends RDS for SQL Server, RDS for MySQL, and RDS for PostgreSQL databases to AWS Outposts Rack Gen1 and Gen2 environments. AWS Outposts uses the same hardware as in public AWS Regions to bring AWS services, infrastructure, and operation models on-premises. With RDS on Outposts, you can provision managed DB instances close to the business applications that must run on-premises. For more information about AWS Outposts, see the AWS Outposts documentation and the AWS Outposts product page.

You use the same AWS Management Console, AWS CLI, and RDS API to provision and manage on-premises RDS on Outposts DB instances as you do for RDS DB instances running in the AWS Cloud. RDS on Outposts automates tasks, such as database provisioning, operating system and database patching, backup, and long-term archival in Amazon S3.

RDS on Outposts supports automated backups of DB instances. Network connectivity between your Outpost and your AWS Region is required to back up and restore DB instances. All DB snapshots and transaction logs from an Outpost are stored in your AWS Region. From your AWS Region, you can restore a DB instance from a DB snapshot to a different Outpost. For more information, see Introduction to backups.

RDS on Outposts supports automated maintenance and upgrades of DB instances. For more information, see Maintaining a DB instance.

RDS on Outposts uses encryption at rest for DB instances and DB snapshots using your AWS KMS key. For more information about encryption at rest, see Encrypting Amazon RDS resources.

By default, EC2 instances in Outposts subnets can use the Amazon Route 53 DNS Service to resolve domain names to IP addresses. You might encounter longer DNS resolution times with Route 53, depending on the path latency between your Outpost and the AWS Region. In such cases, you can use the DNS servers installed locally in your on-premises environment. For more information, see DNS in the *AWS Outposts User Guide*.

When network connectivity to the AWS Region isn't available, your DB instance continues to run locally. You can continue to access DB instances using DNS name resolution by configuring a local DNS server as a secondary server. However, you can't create new DB instances or modify existing DB instances. Automatic backups don't occur when there is no connectivity. If there is a DB instance failure, the DB instance isn't automatically replaced until connectivity is restored. We recommend restoring network connectivity as soon as possible.

Topics

- Prerequisites for Amazon RDS on AWS Outposts
- Amazon RDS on AWS Outposts support for Amazon RDS features
- Supported DB instance classes for Amazon RDS on AWS Outposts
- Customer-owned IP addresses for Amazon RDS on AWS Outposts
- Working with Multi-AZ deployments for Amazon RDS on AWS Outposts
- Creating DB instances for Amazon RDS on AWS Outposts
- Creating read replicas for Amazon RDS on AWS Outposts
- Considerations for restoring DB instances on Amazon RDS on AWS Outposts

Prerequisites for Amazon RDS on AWS Outposts

The following are prerequisites for using Amazon RDS on AWS Outposts:

- Install AWS Outposts in your on-premises data center. For more information, see Installing an AWS Outposts server in the *AWS Outposts Server installation guide*.
- Make sure that you have at least one subnet available for RDS on Outposts. You can use the same subnet for other workloads.
- Make sure that you have a reliable network connection between your Outpost and an AWS Region.

Amazon RDS on AWS Outposts support for Amazon RDS features

The following table describes the Amazon RDS features supported by Amazon RDS on AWS Outposts.

Feature	Supported	Notes	More information

DB instance provisioning	Yes	You can only create DB instances for RDS for SQL Server, RDS for MySQL, and RDS for PostgreSQL DB engines. The following versions are supported:
		• Microsoft SQL Server:
		  • 16.00.4085.2.v1 and higher 2022 versions
		  • 15.00.4043.16.v1 and higher 2019 versions
		  • 14.00.3294.2.v1 and higher 2017 versions
		  • 13.00.5820.21.v1 and higher 2016 versions
		• MySQL 8.0 and 8.4 versions
		• All PostgreSQL 16 & 15 & 14 & 13 versions, and PostgreSQL version 12.5 and higher PostgreSQL 12 versions
			Creating DB instances for Amazon RDS on AWS Outposts

Connect to a Microsoft SQL Server DB instance	Yes	Some TLS versions and encryption ciphers might not be secure. To turn them off, follow the	Connecting to your Microsoft SQL Server DB instance

Feature	Supported	Notes	More information
with Microsoft SQL Server Management Studio		instructions in Configuring SQL Server security protocols and ciphers.	
Modifying the master user password	Yes	None	Modifying an Amazon RDS DB instance
Renaming a DB instance	Yes	None	Modifying an Amazon RDS DB instance
Rebooting a DB instance	Yes	None	Rebooting a DB instance
Stopping a DB instance	Yes	None	Stopping an Amazon RDS DB instance temporarily
Starting a DB instance	Yes	None	Starting an Amazon RDS DB instance that was previously stopped
Multi-AZ deployments	Yes	Multi-AZ deployments are supported on MySQL and PostgreSQL DB instances. Multi-AZ deployments do not support Direct VPC Routing (DVR).	Creating DB instances for Amazon RDS on AWS Outposts Configuring and managing a Multi-AZ deployment for Amazon RDS
DB parameter groups	Yes	None	Parameter groups for Amazon RDS

Feature	Supported	Notes	More information
Read replicas	Yes	Read replicas are supported for MySQL and PostgreSQL DB instances. Read replicas do not support Direct VPC Routing (DVR).	Creating read replicas for Amazon RDS on AWS Outposts
Encryption at rest	Yes	RDS on Outposts doesn't support unencrypted DB instances.	Encrypting Amazon RDS resources
AWS Identity and Access Management (IAM) database authentication	No	None	IAM database authentic ationfor MariaDB, MySQL, and PostgreSQL
Associating an IAM role with a DB instance	No	None	add-role-to-db-instance AWS CLI command AddRoleToDBInstance RDS API operation
Kerberos authentication	No	None	Kerberos authentication
Tagging Amazon RDS resources	Yes	None	Tagging Amazon RDS resources
Option groups	Yes	None	Working with option groups
Modifying the maintenance window	Yes	None	Maintaining a DB instance

Feature	Supported	Notes	More information
Automatic minor version upgrade	Yes	None	Automatically upgrading the minor engine version
Modifying the backup window	Yes	None	Introduction to backups Modifying an Amazon RDS DB instance
Changing the DB instance class	Yes	None	Modifying an Amazon RDS DB instance
Changing the allocated storage	Yes	None	Modifying an Amazon RDS DB instance
Storage autoscaling	Yes	None	Managing capacity automatically with Amazon RDS storage autoscaling

Feature	Supported	Notes	More information
Manual and automatic DB instance snapshots	Yes	You can store automated backups and manual snapshots in your AWS Region. Or you can store them locally on your Outpost. Local backups are supported on MySQL and PostgreSQL DB instances. To store backups on your Outpost, make sure that you have Amazon S3 on Outposts configured. Local backups are not supported for Multi-AZ instance deployments.	Creating DB instances for Amazon RDS on AWS Outposts Amazon S3 on Outposts Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS
Restoring from a DB snapshot	Yes	You can store automated backups and manual snapshots for the restored DB instance in the parent AWS Region or locally on your Outpost.	Considerations for restoring DB instances on Amazon RDS on AWS Outposts Restoring to a DB instance
Restoring a DB instance from Amazon S3	No	None	Restoring a backup into an Amazon RDS for MySQL DB instance
Exporting snapshot data to Amazon S3	No	None	Exporting DB snapshot data to Amazon S3 for Amazon RDS

Feature	Supported	Notes	More information
Point-in-time recovery	Yes	You can store automated backups and manual snapshots for the restored DB instance in the parent AWS Region or locally on your Outpost, with one exception.	Considerations for restoring DB instances on Amazon RDS on AWS Outposts Restoring a DB instance to a specified time for Amazon RDS
Enhanced monitoring	No	None	Monitoring OS metrics with Enhanced Monitoring
Amazon CloudWatch monitoring	Yes	You can view the same set of metrics that are available for your databases in the AWS Region.	Monitoring Amazon RDS metrics with Amazon CloudWatch
Publishing database engine logs to CloudWatch Logs	Yes	None	Publishing database logs to Amazon CloudWatch Logs
Event notification	Yes	None	Working with Amazon RDS event notification
Amazon RDS Performance Insights	No	None	Monitoring DB load with Performance Insights on Amazon RDS

Feature	Supported	Notes	More information
Viewing or downloading database logs	No	RDS on Outposts doesn't support viewing database logs using the console or describing database logs using the AWS CLI or RDS API. RDS on Outposts doesn't support downloading database logs using the console or downloading database logs using the AWS CLI or RDS API.	Monitoring Amazon RDS log files
Amazon RDS Proxy	No	None	Amazon RDS Proxy
Stored procedures for Amazon RDS for MySQL	Yes	None	RDS for MySQL stored procedure reference
Replication with external databases for RDS for MySQL	No	None	Configuring binary log file position replication with an external source instance
Native backup and restore for Amazon RDS for Microsoft SQL Server	Yes	None	Importing and exporting SQL Server databases using native backup and restore

Supported DB instance classes for Amazon RDS on AWS Outposts

Amazon RDS on AWS Outposts supports the following DB instance classes:

- General purpose DB instance classes
  - db.m7i.48xlarge
  - db.m7i.24xlarge
  - db.m7i.16xlarge
  - db.m7i.12xlarge
  - db.m7i.8xlarge
  - db.m7i.4xlarge
  - db.m7i.2xlarge
  - db.m7i.xlarge
  - db.m7i.large
  - db.m5.24xlarge
  - db.m5.16xlarge
  - db.m5.12xlarge
  - db.m5.8xlarge
  - db.m5.4xlarge
  - db.m5.2xlarge
  - db.m5.xlarge
  - db.m5.large
- Memory optimized DB instance classes
  - db.r7i.48xlarge
  - db.r7i.24xlarge
  - db.r7i.16xlarge
  - db.r7i.12xlarge
  - db.r7i.8xlarge
  - db.r7i.4xlarge
  - db.r7i.2xlarge

- db.r7i.xlarge
- db.r7i.large
- db.r5.24xlarge
- db.r5.16xlarge
- db.r5.12xlarge
- db.r5.8xlarge
- db.r5.4xlarge
- db.r5.2xlarge
- db.r5.xlarge
- db.r5.large

Depending on how you've configured your Outpost, you might not have all of these classes available. For example, if you haven't purchased the db.r5 classes for your Outpost, you can't use them with RDS on Outposts.

Only general purpose SSD storage is supported for RDS on Outposts DB instances. For more information about DB instance classes, see DB instance classes.

Amazon RDS manages maintenance and recovery for your DB instances and requires active capacity on the Outpost to do so. We recommend that you configure N+1 EC2 instances for each DB instance class in your production environments. RDS on Outposts can use the extra capacity of these EC2 instances for maintenance and repair operations. For example, if your production environments have 3 db.m5.large and 5 db.r5.xlarge DB instance classes, then we recommend that they have at least 4 m5.large EC2 instances and 6 r5.xlarge EC2 instances. For more information, see Resilience in AWS Outposts in the *AWS Outposts User Guide*.

Customer-owned IP addresses for Amazon RDS on AWS Outposts

Amazon RDS on AWS Outposts uses information that you provide about your on-premises network to create an address pool. This pool is known as a *customer-owned IP address pool* (CoIP pool). *Customer-owned IP addresses* (CoIPs) provide local or external connectivity to resources in your Outpost subnets through your on-premises network. For more information about CoIPs, see Customer-owned IP addresses in the *AWS Outposts User Guide*.

Each RDS on Outposts DB instance has a private IP address for traffic inside its virtual private cloud (VPC). This private IP address isn't publicly accessible. You can use the **Public** option to set whether the DB instance also has a public IP address in addition to the private IP address. Using the public IP address for connections routes them through the internet and can result in high latencies in some cases.

Instead of using these private and public IP addresses, RDS on Outposts supports using CoIPs for DB instances through their subnets. When you use a CoIP for an RDS on Outposts DB instance, you connect to the DB instance with the DB instance endpoint. RDS on Outposts then automatically uses the CoIP for all connections from both inside and outside of the VPC.

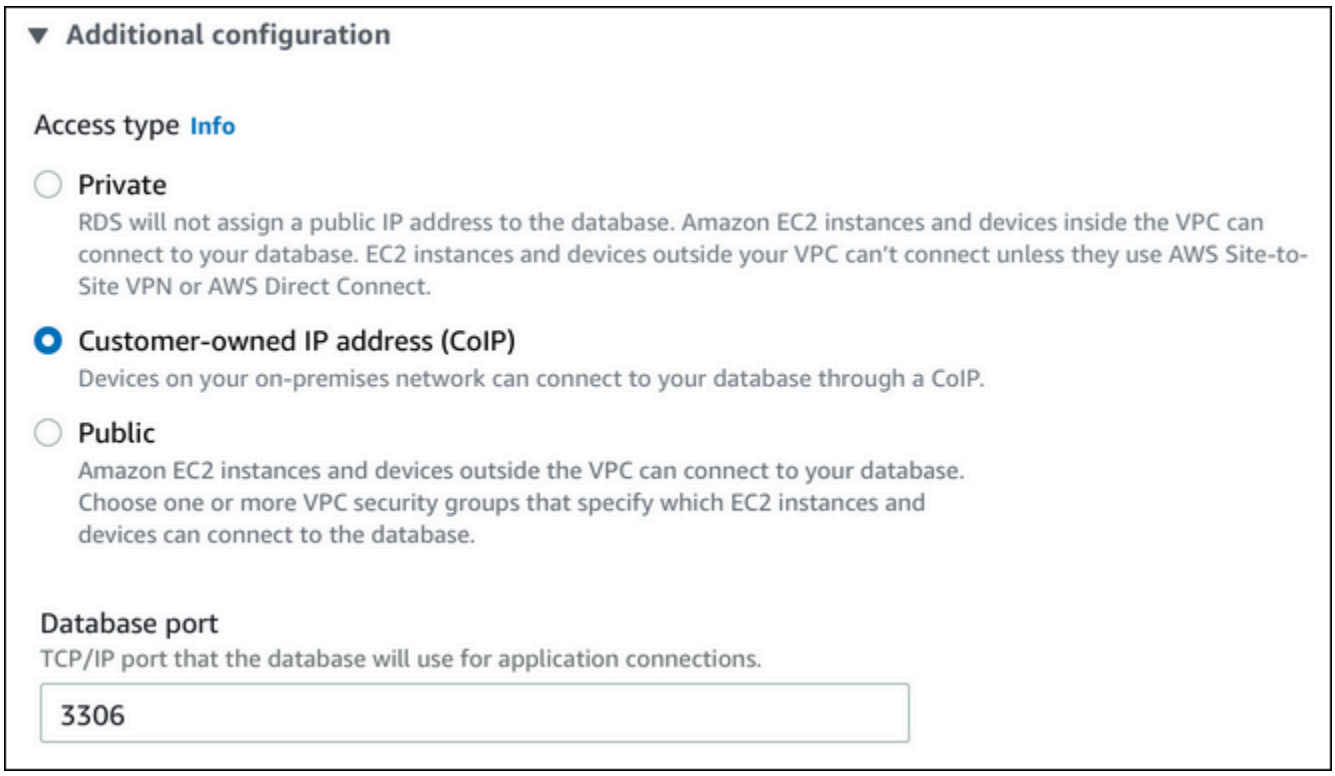
CoIPs can provide the following benefits for RDS on Outposts DB instances:

- Lower connection latency
- Enhanced security

Using CoIPs

You can turn CoIPs on or off for an RDS on Outposts DB instance using the AWS Management Console, the AWS CLI, or the RDS API:

- With the AWS Management Console, choose the **Customer-owned IP address (CoIP)** setting in **Access type** to use CoIPs. Choose one of the other settings to turn them off.



- With the AWS CLI, use the `--enable-customer-owned-ip | --no-enable-customer-owned-ip` option.
- With the RDS API, use the `EnableCustomerOwnedIp` parameter.

You can turn CoIPs on or off when you perform any of the following actions:

- Create a DB instance

  For more information, see Creating DB instances for Amazon RDS on AWS Outposts.
- Modify a DB instance

  For more information, see Modifying an Amazon RDS DB instance.
- Create a read replica

  For more information, see Creating read replicas for Amazon RDS on AWS Outposts.
- Restore a DB instance from a snapshot

  For more information, see Restoring to a DB instance.
- Restore a DB instance to a specified time

For more information, see Restoring a DB instance to a specified time for Amazon RDS.

> **Note**
>
> In some cases, you might turn on CoIPs for a DB instance but Amazon RDS isn't able to allocate a CoIP for the DB instance. In such cases, the DB instance status is changed to **incompatible-network**. For more information about the DB instance status, see Viewing Amazon RDSDB instance status.

Limitations

The following limitations apply to CoIP support for RDS on Outposts DB instances:

- When using a CoIP for a DB instance, make sure that public accessibility is turned off for that DB instance.
- Make sure that the inbound rules for your VPC security groups include the CoIP address range (CIDR block). For more information about setting up security groups, see Provide access to your DB instance in your VPC by creating a security group.
- You can't assign a CoIP from a CoIP pool to a DB instance. When you use a CoIP for a DB instance, Amazon RDS automatically assigns a CoIP from a CoIP pool to the DB instance.
- You must use the AWS account that owns the Outpost resources (owner) or share the following resources with other AWS accounts (consumers) in the same organization:
  - The Outpost
  - The local gateway (LGW) route table for the DB instance's VPC
  - The CoIP pool or pools for the LGW route table

  For more information, see Working with shared AWS Outposts resources in the *AWS Outposts User Guide*.

Working with Multi-AZ deployments for Amazon RDS on AWS Outposts

For Multi-AZ deployments, Amazon RDS creates a primary DB instance on one AWS Outpost. RDS synchronously replicates the data to a standby DB instance on a different Outpost.

Multi-AZ deployments on AWS Outposts operate like Multi-AZ deployments in AWS Regions, but with the following differences:

- They require a local connection between two or more Outposts.
- They require customer-owned IP (CoIP) pools. For more information, see Customer-owned IP addresses for Amazon RDS on AWS Outposts.
- Replication runs on your local network.

Multi-AZ on AWS Outposts is available for all supported versions of MySQL and PostgreSQL on RDS on Outposts. Local backups aren't supported for Multi-AZ deployments. For more information, see Creating DB instances for Amazon RDS on AWS Outposts.

Working with the shared responsibility model

Although AWS uses commercially reasonable efforts to provide DB instances configured for high availability, the availability uses a shared responsibility model. The ability of RDS on Outposts to fail over and repair DB instances requires each of your Outposts to be connected to its AWS Region.

RDS on Outposts also requires connectivity between the Outpost that is hosting the primary DB instance and the Outpost that is hosting the standby DB instance for synchronous replication. Any impact to this connection can prevent RDS on Outposts from performing a failover.

You might see elevated latencies for a standard DB instance deployment as a result of the synchronous data replication. The bandwidth and latency of the connection between the Outpost hosting the primary DB instance and the Outpost hosting the standby DB instance directly affect latencies. For more information, see Prerequisites.

Improving availability

We recommend the following actions to improve availability:

- Allocate enough additional capacity for your mission-critical applications to allow recovery and failover if there is an underlying host issue. This applies to all Outposts that contain subnets in your DB subnet group. For more information, see Resilience in AWS Outposts.
- Provide redundant network connectivity for your Outposts.
- Use more than two Outposts. Having more than two Outposts allows Amazon RDS to recover a DB instance. RDS does this recovery by moving the DB instance to another Outpost if the current Outpost experiences a failure.
- Provide dual power sources and redundant network connectivity for your Outpost.

We recommend the following for your local networks:

- The round trip time (RTT) latency between the Outpost hosting your primary DB instance and the Outpost hosting your standby DB instance directly affects write latency. Keep the RTT latency between the AWS Outposts in the low single-digit milliseconds. We recommend not more than 5 milliseconds, but your requirements might vary.

  You can find the net impact to network latency in the Amazon CloudWatch metrics for `WriteLatency`. For more information, see Amazon CloudWatch metrics for Amazon RDS.
- The availability of the connection between the Outposts affects the overall availability of your DB instances. Have redundant network connectivity between the Outposts.

Prerequisites

Multi-AZ deployments on RDS on Outposts have the following prerequisites:

- Have at least two Outposts, connected over local connections and attached to different Availability Zones in an AWS Region.
- Make sure that your DB subnet groups contain the following:
  - At least two subnets in at least two Availability Zones in a given AWS Region.
  - Subnets only in Outposts.
  - At least two subnets in at least two Outposts within the same virtual private cloud (VPC).
- Associate your DB instance's VPC with all of your local gateway route tables. This association is necessary because replication runs over your local network using your Outposts' local gateways.

For example, suppose that your VPC contains subnet-A in Outpost-A and subnet-B in Outpost-B. Outpost-A uses LocalGateway-A (LGW-A), and Outpost-B uses LocalGateway-B (LGW-B). LGW-A has RouteTable-A, and LGW-B has RouteTable-B. You want to use both RouteTable-A and RouteTable-B for replication traffic. To do this, associate your VPC with both RouteTable-A and RouteTable-B.

For more information about how to create an association, see the Amazon EC2 create-local-gateway-route-table-vpc-association AWS CLI command.

- Make sure that your Outposts use customer-owned IP (CoIP) routing. Each route table must also each have at least one address pool. Amazon RDS allocates an additional IP address each for the primary and standby DB instances for data synchronization.
- Make sure that the AWS account that owns the RDS DB instances owns the local gateway route tables and CoIP pools. Or make sure it's part of a Resource Access Manager share with access to the local gateway route tables and CoIP pools.
- Make sure that the IP addresses in your CoIP pools can be routed from one Outpost local gateway to the others.
- Make sure that the VPC's CIDR blocks (for example, 10.0.0.0/4) and your CoIP pool CIDR blocks don't contain IP addresses from Class E (240.0.0.0/4). RDS uses these IP addresses internally.
- Make sure that you correctly set up outbound and related inbound traffic.

  RDS on Outposts establishes a virtual private network (VPN) connection between the primary and standby DB instances. For this to work correctly, your local network must allow outbound and related inbound traffic for Internet Security Association and Key Management Protocol (ISAKMP). It does so using User Datagram Protocol (UDP) port 500 and IP Security (IPsec) Network Address Translation Traversal (NAT-T) using UDP port 4500.

For more information on CoIPs, see Customer-owned IP addresses for Amazon RDS on AWS Outposts in this guide, and Customer-owned IP addresses in the *AWS Outposts User Guide*.

Working with API operations for Amazon EC2 permissions

Regardless of whether you use CoIPs for your DB instance on AWS Outposts, RDS requires access to your CoIP pool resources. RDS can call the following EC2 permissions API operations for CoIPs on your behalf for Multi-AZ deployments:

- `CreateCoipPoolPermission` – When you create a Multi-AZ DB instance on RDS on Outposts

- `DeleteCoipPoolPermission` – When you delete a Multi-AZ DB instance on RDS on Outposts

These API operations grant to, or remove from, internal RDS accounts the permission to allocate elastic IP addresses from the CoIP pool specified by the permission. You can view these IP addresses using the `DescribeCoipPoolUsage` API operation. For more information on CoIPs, see Customer-owned IP addresses for Amazon RDS on AWS Outposts and Customer-owned IP addresses in the *AWS Outposts User Guide*.

RDS can also call the following EC2 permission API operations for local gateway route tables on your behalf for Multi-AZ deployments:

- `CreateLocalGatewayRouteTablePermission` – When you create a Multi-AZ DB instance on RDS on Outposts
- `DeleteLocalGatewayRouteTablePermission` – When you delete a Multi-AZ DB instance on RDS on Outposts

These API operations grant to, or remove from, internal RDS accounts the permission to associate internal RDS VPCs with your local gateway route tables. You can view these route table–VPC associations using the `DescribeLocalGatewayRouteTableVpcAssociations` API operations.

Creating DB instances for Amazon RDS on AWS Outposts

Creating an Amazon RDS on AWS Outposts DB instance is similar to creating an Amazon RDS DB instance in the AWS Cloud. However, make sure that you specify a DB subnet group that is associated with your Outpost.

A virtual private cloud (VPC) based on the Amazon VPC service can span all of the Availability Zones in an AWS Region. You can extend any VPC in the AWS Region to your Outpost by adding an Outpost subnet. To add an Outpost subnet to a VPC, specify the Amazon Resource Name (ARN) of the Outpost when you create the subnet.

Before you create an RDS on Outposts DB instance, you can create a DB subnet group that includes one subnet that is associated with your Outpost. When you create an RDS on Outposts DB instance, specify this DB subnet group. You can also choose to create a new DB subnet group when you create your DB instance.

For information about configuring AWS Outposts, see the AWS Outposts User Guide.

Console

Creating a DB subnet group

Create a DB subnet group with one subnet that is associated with your Outpost.

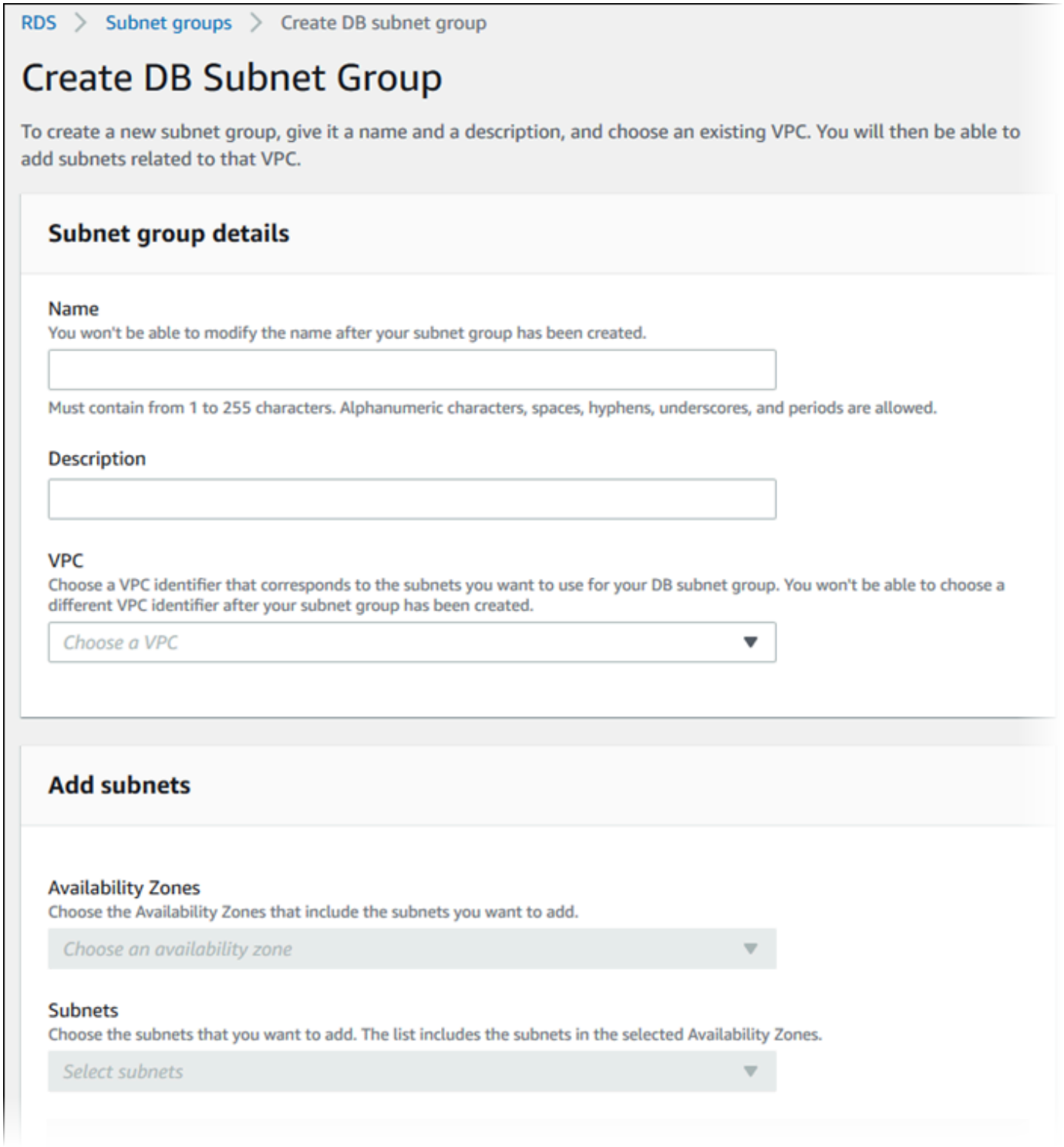
You can also create a new DB subnet group for the Outpost when you create your DB instance. If you want to do so, then skip this procedure.

> **Note**
>
> To create a DB subnet group for the AWS Cloud, specify at least two subnets.

To create a DB subnet group for your Outpost

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region where you want to create the DB subnet group.
3. Choose **Subnet groups**, and then choose **Create DB Subnet Group**.

The **Create DB subnet group** page appears.



4. For **Name**, choose the name of the DB subnet group.
5. For **Description**, choose a description for the DB subnet group.
6. For **VPC**, choose the VPC that you're creating the DB subnet group for.

7. For **Availability Zones**, choose the Availability Zone for your Outpost.
8. For **Subnets**, choose the subnet for use by RDS on Outposts.
9. Choose **Create** to create the DB subnet group.

Creating the RDS on Outposts DB instance

Create the DB instance, and choose the Outpost for your DB instance.

To create an RDS on Outposts DB instance using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region where the Outpost on which you want to create the DB instance is attached.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.

   The AWS Management Console detects available Outposts that you have configured and presents the **On-premises** option in the **Database location** section.

   > **Note**
   >
   > If you haven't configured any Outposts, either the **Database location** section doesn't appear or the **RDS on Outposts** option isn't available in the **Choose an on-premises creation method** section.

5. For **Database location**, choose **On-premises**.
6. For **On-premises creation method**, choose **RDS on Outposts**.
7. Specify your settings for **Outposts Connectivity**. These settings are for the Outpost that uses the VPC that has the DB subnet group for your DB instance. Your VPC must be based on the Amazon VPC service.

   a. For **Virtual Private Cloud (VPC)**, choose the VPC that contains the DB subnet group for your DB instance.

   b. For **VPC security group**, choose the Amazon VPC security group for your DB instance.

   c. For **DB subnet group**, choose the DB subnet group for your DB instance.

You can choose an existing DB subnet group that's associated with the Outpost—for example, if you performed the procedure in Creating a DB subnet group.

You can also create a new DB subnet group for the Outpost.

8. For **Multi-AZ deployment**, choose **Create a standby instance (recommended for production usage)** to create a standby DB instance in another Outpost.

> **Note**
>
> This option isn't available for Microsoft SQL Server.
> If you choose to create a Multi-AZ deployment, you can't store backups on your Outpost.

9. Under **Backup**, do the following:

   a. For **Backup target**, choose one of the following:

   - **AWS Cloud** to store automated backups and manual snapshots in the parent AWS Region.
   - **Outposts (on-premises)** to create local backups.

   > **Note**
   >
   > To store backups on your Outpost, your Outpost must have Amazon S3 capability. For more information, see Amazon S3 on Outposts.
   > Local backups aren't supported for Multi-AZ deployments or read replicas.

   b. Choose **Enable automated backups** to create point-in-time snapshots of your DB instance.

   If you turn on automated backups, then you can choose values for **Backup retention period** and **Backup window**, or leave the default values.

10. Specify other DB instance settings as needed.

    For information about each setting when creating a DB instance, see Settings for DB instances.

11. Choose **Create database**.

The **Databases** page appears. A banner tells you that your DB instance is being created, and displays the **View credential details** button.

Viewing DB instance details

After you create your DB instance, you can view credentials and other details for it.

To view DB instance details

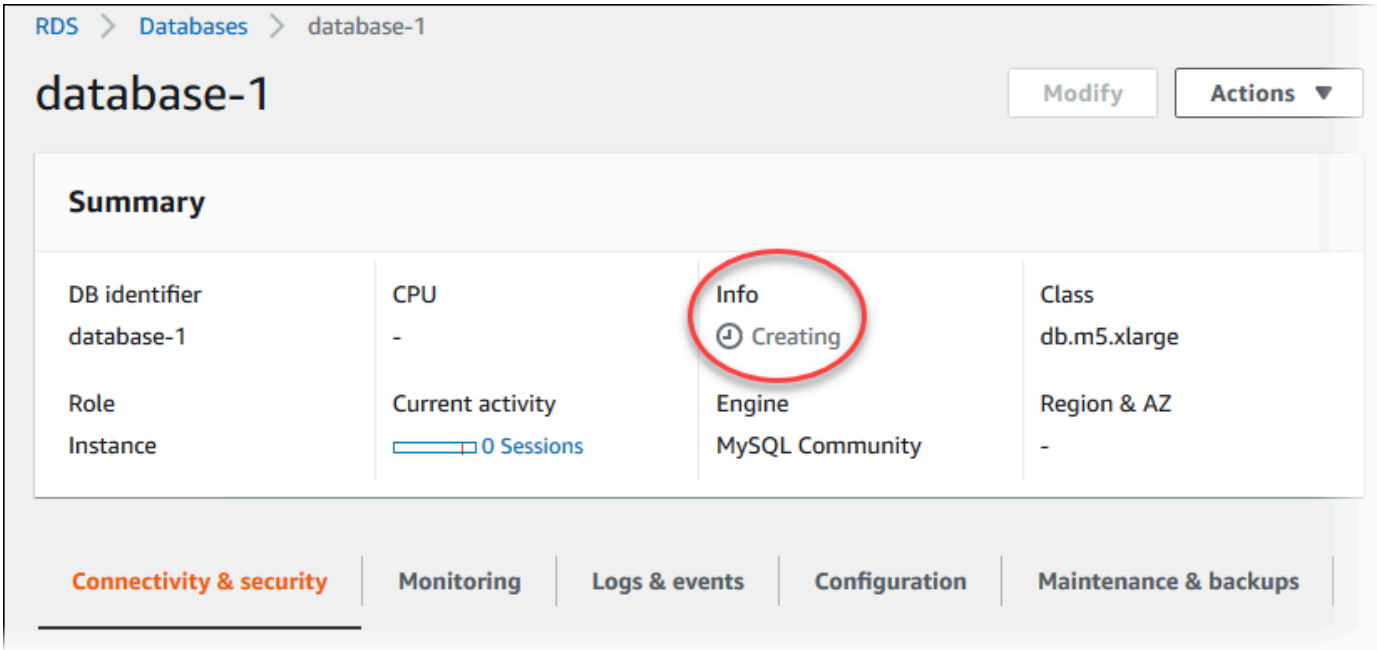
1. To view the master user name and password for the DB instance, choose **View credential details** on the **Databases** page.

   You can connect to the DB instance as the master user by using these credentials.

   > **Important**
   >
   > You can't view the master user password again. If you don't record it, you might have to change it. To change the master user password after the DB instance is available, modify the DB instance. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

2. Choose the name of the new DB instance on the **Databases** page.

   On the RDS console, the details for the new DB instance appear. The DB instance has a status of **Creating** until the DB instance is created and ready for use. When the state changes to **Available**, you can connect to the DB instance. Depending on the DB instance class and storage allocated, it can take several minutes for the new DB instance to be available.



After the DB instance is available, you can manage it the same way that you manage RDS DB instances in the AWS Cloud.

AWS CLI

Before you create a new DB instance in an Outpost with the AWS CLI, first create a DB subnet group for use by RDS on Outposts.

To create a DB subnet group for your Outpost

- Use the create-db-subnet-group command. For `--subnet-ids`, specify the subnet group in the Outpost for use by RDS on Outposts.

  For Linux, macOS, or Unix:

```
aws rds create-db-subnet-group \
    --db-subnet-group-name myoutpostdbsubnetgr \
    --db-subnet-group-description "DB subnet group for RDS on Outposts" \
    --subnet-ids subnet-abc123
```

  For Windows:

```
aws rds create-db-subnet-group ^
    --db-subnet-group-name myoutpostdbsubnetgr ^
```

```
    --db-subnet-group-description "DB subnet group for RDS on Outposts" ^
    --subnet-ids subnet-abc123
```

To create an RDS on Outposts DB instance using the AWS CLI

- Use the create-db-instance command. Specify an Availability Zone for the Outpost, an Amazon VPC security group associated with the Outpost, and the DB subnet group you created for the Outpost. You can include the following options:
  - `--db-instance-identifier`
  - `--db-instance-class`
  - `--engine` – The database engine. Use one of the following values:
    - MySQL – Specify `mysql`.
    - PostgreSQL – Specify `postgres`.
    - Microsoft SQL Server – Specify `sqlserver-ee`, `sqlserver-se`, or `sqlserver-web`.
  - `--availability-zone`
  - `--vpc-security-group-ids`
  - `--db-subnet-group-name`
  - `--allocated-storage`
  - `--max-allocated-storage`
  - `--master-username`
  - `--master-user-password`
  - `--multi-az | --no-multi-az` – (Optional) Whether to create a standby DB instance in a different Availability Zone. The default is `--no-multi-az`.

    The `--multi-az` option isn't available for SQL Server.
  - `--backup-retention-period`
  - `--backup-target` – (Optional) Where to store automated backups and manual snapshots. Use one of the following values:
    - `outposts` – Store them locally on your Outpost.
    - `region` – Store them in the parent AWS Region. This is the default value.

If you use the `--multi-az` option, you can't use `outposts` for `--backup-target`. In addition, the DB instance can't have read replicas if you use `outposts` for `--backup-target`.

- `--storage-encrypted`
- `--kms-key-id`

Example

The following example creates a MySQL DB instance named `myoutpostdbinstance` with backups stored on your Outpost.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier myoutpostdbinstance \
    --engine-version 8.0.17 \
    --db-instance-class db.m5.large \
    --engine mysql \
    --availability-zone us-east-1d \
    --vpc-security-group-ids outpost-sg \
    --db-subnet-group-name myoutpostdbsubnetgr \
    --allocated-storage 100 \
    --max-allocated-storage 1000 \
    --master-username masterawsuser \
    --manage-master-user-password \
    --backup-retention-period 3 \
    --backup-target outposts \
    --storage-encrypted \
    --kms-key-id mykey
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier myoutpostdbinstance ^
    --engine-version 8.0.17 ^
    --db-instance-class db.m5.large ^
    --engine mysql ^
    --availability-zone us-east-1d ^
    --vpc-security-group-ids outpost-sg ^
    --db-subnet-group-name myoutpostdbsubnetgr ^
```

```
    --allocated-storage 100 ^
    --max-allocated-storage 1000 ^
    --master-username masterawsuser ^
    --manage-master-user-password ^
    --backup-retention-period 3 ^
    --backup-target outposts ^
    --storage-encrypted ^
    --kms-key-id mykey
```

For information about each setting when creating a DB instance, see Settings for DB instances.

RDS API

To create a new DB instance in an Outpost with the RDS API, first create a DB subnet group for use by RDS on Outposts by calling the CreateDBSubnetGroup operation. For `SubnetIds`, specify the subnet group in the Outpost for use by RDS on Outposts.

Next, call the CreateDBInstance operation with the following parameters. Specify an Availability Zone for the Outpost, an Amazon VPC security group associated with the Outpost, and the DB subnet group you created for the Outpost.

- `AllocatedStorage`
- `AvailabilityZone`
- `BackupRetentionPeriod`
- `BackupTarget`

  If you are creating a Multi-AZ DB instance deployment, you can't use `outposts` for `BackupTarget`. In addition, the DB instance can't have read replicas if you use `outposts` for `BackupTarget`.
- `DBInstanceClass`
- `DBInstanceIdentifier`
- `VpcSecurityGroupIds`
- `DBSubnetGroupName`
- `Engine`
- `EngineVersion`
- `MasterUsername`
- `MasterUserPassword`

- `MaxAllocatedStorage` (optional)
- `MultiAZ` (optional)
- `StorageEncrypted`
- `KmsKeyID`

For information about each setting when creating a DB instance, see Settings for DB instances.

Creating read replicas for Amazon RDS on AWS Outposts

Amazon RDS on AWS Outposts uses the MySQL and PostgreSQL DB engines' built-in replication functionality to create a read replica from a source DB instance. The source DB instance becomes the primary DB instance. Updates made to the primary DB instance are asynchronously copied to the read replica. You can reduce the load on your primary DB instance by routing read queries from your applications to the read replica. Using read replicas, you can elastically scale out beyond the capacity constraints of a single DB instance for read-heavy database workloads.

When you create a read replica from an RDS on Outposts DB instance, the read replica uses a customer-owned IP address (CoIP). For more information, see Customer-owned IP addresses for Amazon RDS on AWS Outposts.

Read replicas on RDS on Outposts have the following limitations:

- You can't create read replicas for RDS for SQL Server on RDS on Outposts DB instances.
- Cross-Region read replicas aren't supported on RDS on Outposts.
- Cascading read replicas aren't supported on RDS on Outposts.
- The source RDS on Outposts DB instance can't have local backups. The backup target for the source DB instance must be your AWS Region.
- Read replicas require customer-owned IP (CoIP) pools. For more information, see Customer-owned IP addresses for Amazon RDS on AWS Outposts.
- Read replicas on RDS on Outposts can only be created in the same virtual private cloud (VPC) as the source DB instance.
- Read replicas on RDS on Outposts can be located on the same Outpost or another Outpost in the same VPC as the source DB instance.
- You can't create read replicas for DB instances encrypted with AWS KMS External Key Store (XKS).

You can create a read replica from an RDS on Outposts DB instance using the AWS Management Console, AWS CLI, or RDS API. For more information on read replicas, see Working with DB instance read replicas.

Console

To create a read replica from a source DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to use as the source for a read replica.
4. For **Actions**, choose **Create read replica**.
5. For **DB instance identifier**, enter a name for the read replica.
6. Specify your settings for **Outposts Connectivity**. These settings are for the Outpost that uses the virtual private cloud (VPC) that has the DB subnet group for your DB instance. Your VPC must be based on the Amazon VPC service.
7. Choose your **DB instance class**. We recommend that you use the same or larger DB instance class and storage type as the source DB instance for the read replica.
8. For **Multi-AZ deployment**, choose **Create a standby instance (recommended for production usage)** to create a standby DB instance in a different Availability Zone.

   Creating your read replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ DB instance.
9. (Optional) Under **Connectivity**, set values for **Subnet Group** and **Availability Zone**.

   If you specify values for both **Subnet Group** and **Availability Zone**, the read replica is created on an Outpost that is associated with the Availability Zone in the DB subnet group.

   If you specify a value for **Subnet Group** and **No preference** for **Availability Zone**, the read replica is created on a random Outpost in the DB subnet group.
10. For **AWS KMS key**, choose the AWS KMS key identifier of the KMS key.

    The read replica must be encrypted.
11. Choose other options as needed.
12. Choose **Create read replica**.

After the read replica is created, you can see it on the **Databases** page in the RDS console. It shows **Replica** in the **Role** column.

AWS CLI

To create a read replica from a source MySQL or PostgreSQL DB instance, use the AWS CLI command create-db-instance-read-replica.

You can control where the read replica is created by specifying the `--db-subnet-group-name` and `--availability-zone` options:

- If you specify both the `--db-subnet-group-name` and `--availability-zone` options, the read replica is created on an Outpost that is associated with the Availability Zone in the DB subnet group.
- If you specify the `--db-subnet-group-name` option and don't specify the `--availability-zone` option, the read replica is created on a random Outpost in the DB subnet group.
- If you don't specify either option, the read replica is created on the same Outpost as the source RDS on Outposts DB instance.

The following example creates a replica and specifies the location of the read replica by including `--db-subnet-group-name` and `--availability-zone` options.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance-read-replica \
    --db-instance-identifier myreadreplica \
    --source-db-instance-identifier mydbinstance \
    --availability-zone us-west-2a
```

For Windows:

```
aws rds create-db-instance-read-replica ^
    --db-instance-identifier myreadreplica ^
    --source-db-instance-identifier mydbinstance ^
    --availability-zone us-west-2a
```

RDS API

To create a read replica from a source MySQL or PostgreSQL DB instance, call the Amazon RDS API CreateDBInstanceReadReplica operation with the following required parameters:

- `DBInstanceIdentifier`
- `SourceDBInstanceIdentifier`

You can control where the read replica is created by specifying the `DBSubnetGroupName` and `AvailabilityZone` parameters:

- If you specify both the `DBSubnetGroupName` and `AvailabilityZone` parameters, the read replica is created on an Outpost that is associated with the Availability Zone in the DB subnet group.
- If you specify the `DBSubnetGroupName` parameter and don't specify the `AvailabilityZone` parameter, the read replica is created on a random Outpost in the DB subnet group.
- If you don't specify either parameter, the read replica is created on the same Outpost as the source RDS on Outposts DB instance.

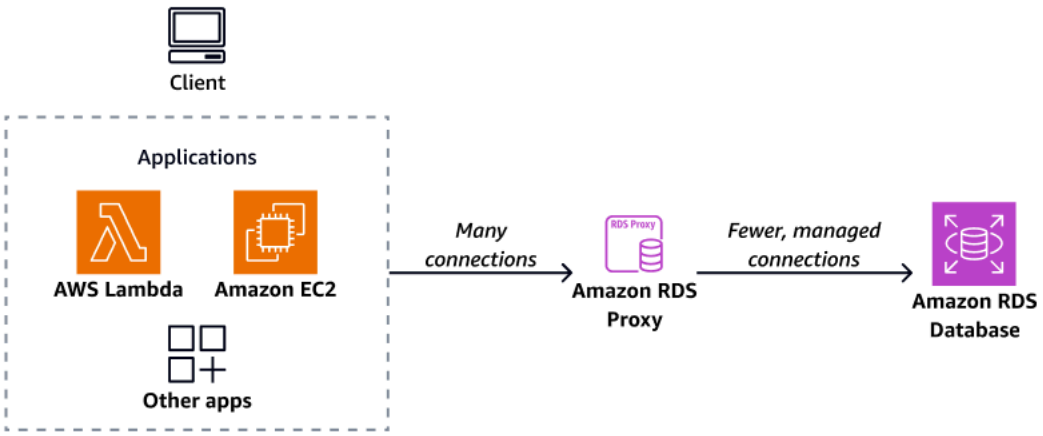
Considerations for restoring DB instances on Amazon RDS on AWS Outposts

When you restore a DB instance in Amazon RDS on AWS Outposts, you can generally choose the storage location for automated backups and manual snapshots of the restored DB instance.

- When restoring from a manual DB snapshot, you can store backups either in the parent AWS Region or locally on your Outpost.
- When restoring from an automated backup (point-in-time recovery), you have fewer choices:
  - If restoring from the parent AWS Region, you can store backups either in the AWS Region or on your Outpost.
  - If restoring from your Outpost, you can store backups only on your Outpost.

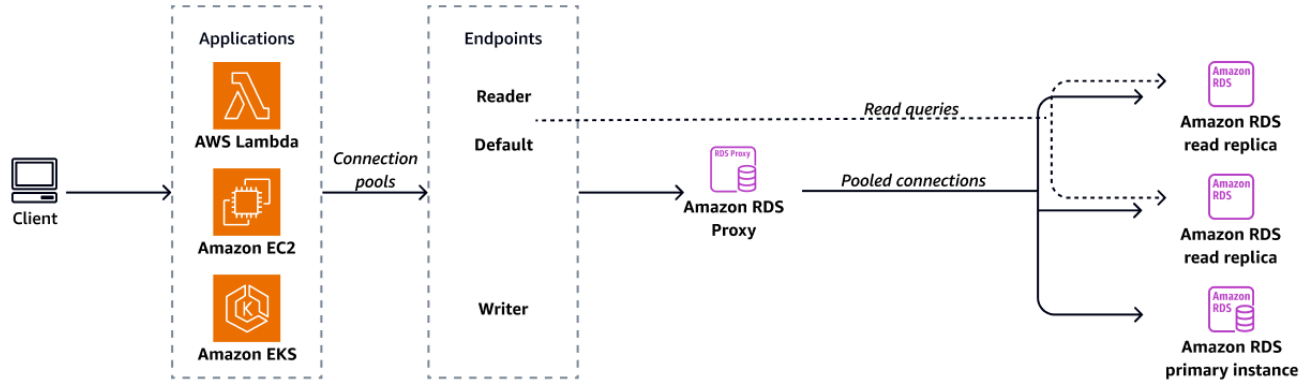
Amazon RDS Proxy

By using Amazon RDS Proxy, you can allow your applications to pool and share database connections to improve their ability to scale. RDS Proxy makes applications more resilient to database failures by automatically connecting to a standby DB instance while preserving application connections. By using RDS Proxy, you can enforce AWS Identity and Access Management (IAM) authentication for clients connecting to the proxy, and the proxy can connect to databases using either IAM database authentication or credentials stored in AWS Secrets Manager.



Using RDS Proxy, you can handle unpredictable surges in database traffic. Otherwise, these surges might cause issues due to oversubscribing connections or new connections being created at a fast rate. RDS Proxy establishes a database connection pool and reuses connections in this pool. This approach avoids the memory and CPU overhead of opening a new database connection each time. To protect a database against oversubscription, you can control the number of database connections that are created.

RDS Proxy queues or throttles application connections that can't be served immediately from the connection pool. Although latencies might increase, your application can continue to scale without abruptly failing or overwhelming the database. If connection requests exceed the limits you specify, RDS Proxy rejects application connections (that is, it sheds load). At the same time, it maintains predictable performance for the load that RDS can serve with the available capacity.



You can reduce the overhead to process credentials and establish a secure connection for each new connection. RDS Proxy can handle some of that work on behalf of the database.

RDS Proxy is fully compatible with the engine versions that it supports. You can enable RDS Proxy for most applications with no code changes.

Topics

- Region and version availability
- Quotas and limitations for RDS Proxy
- Planning where to use RDS Proxy
- RDS Proxy concepts and terminology
- Getting started with RDS Proxy
- Managing an RDS Proxy
- Working with Amazon RDS Proxy endpoints
- Monitoring RDS Proxy metrics with Amazon CloudWatch
- Working with RDS Proxy events
- Troubleshooting for RDS Proxy
- Using RDS Proxy with AWS CloudFormation

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability of Amazon RDS with RDS Proxy, see Supported Regions and DB engines for Amazon RDS Proxy.

Quotas and limitations for RDS Proxy

The following quotas and limitations apply to RDS Proxy:

- Each AWS account ID is limited to 20 proxies. If your application requires more proxies, request an increase via the **Service Quotas** page within the AWS Management Console. In the **Service Quotas** page, select **Amazon Relational Database Service (Amazon RDS)** and locate **Proxies** to request a quota increase. AWS can automatically increase your quota or pending review of your request by Support.
- Each proxy can have up to 200 associated Secrets Manager secrets, thus limiting connections to up to 200 different user accounts when using secrets.
- Each proxy has a default endpoint. You can also add up to 20 proxy endpoints for each proxy. You can create, view, modify, and delete these endpoints.
- For RDS DB instances in replication configurations, you can associate a proxy only with the writer DB instance, not a read replica.
- Your RDS Proxy must be in the same virtual private cloud (VPC) as the database. The proxy can't be publicly accessible, although the database can be. For example, if you're prototyping your database on a local host, you can't connect to your proxy unless you set up the necessary network requirements to allow connection to the proxy. This is because your local host is outside of the proxy's VPC.
- You can't use RDS Proxy with a VPC that has its tenancy set to `dedicated`.
- You can't use RDS Proxy in a VPC that has encryption controls with `Enforce Mode` enabled.
- For IPv6 endpoint network types, configure your VPC and subnets to support only IPv6. For both IPv4 and IPv6 target connection network types, configure your VPC and subnets to support dual-stack mode.
- If you use RDS Proxy with an RDS DB instance that has IAM authentication enabled, the proxy can connect to the database using either IAM authentication or credentials stored in Secrets Manager. Clients connecting to the proxy must authenticate using IAM credentials. For detailed configuration instructions, see Setting up database credentials for RDS Proxy and Configuring IAM authentication for RDS Proxy
- You can't use RDS Proxy with custom DNS when using SSL hostname validation.
- Each proxy can be associated with a single target DB instance . However, you can associate multiple proxies with the same DB instance .
- Any statement with a text size greater than 16 KB causes the proxy to pin the session to the current connection.

- Certain Regions have Availability-Zone (AZ) restrictions to consider while creating your proxy. US East (N. Virginia) Region does not support RDS Proxy in the `use1-az3` Availability Zone. US West (N. California) Region does not support RDS Proxy in the `usw1-az2` Availability Zone. When selecting subnets while creating your proxy, make sure that you don't select subnets in the Availability Zones mentioned above.
- Currently, RDS Proxy doesn't support any global condition context keys.

  For more information about global condition context keys, see AWS global condition context keys in the *IAM User Guide*.
- You can't use RDS Proxy with RDS Custom for SQL Server.
- To reflect any database parameter group modification to your proxy, an instance reboot is required even if your chose to apply your changes immediately. For cluster-level parameters, a cluster-wide reboot is required.
- Your proxy automatically creates the `rdsproxyadmin` DB user when you register a proxy target. This is a protected user that is essential for proxy functionality. You should avoid tampering with the `rdsproxyadmin` user in any capacity. Deleting or modifying the `rdsproxyadmin` user or its permissions can result in complete unavailability of the proxy to your application.

For additional limitations for each DB engine, see the following sections:

- Additional limitations for RDS for MariaDB
- Additional limitations for RDS for Microsoft SQL Server
- Additional limitations for RDS for MySQL
- Additional limitations for RDS for PostgreSQL

Additional limitations for RDS for MariaDB

The following additional limitations apply to RDS Proxy with RDS for MariaDB databases:

- Currently, all proxies listen on port 3306 for MariaDB. The proxies still connect to your database using the port that you specified in the database settings.
- You can't use RDS Proxy with self-managed MariaDB databases in Amazon EC2 instances.
- You can't use RDS Proxy with an RDS for MariaDB DB instance that has the `read_only` parameter in its DB parameter group set to 1.

- RDS Proxy doesn't support MariaDB compressed mode. For example, it doesn't support the compression used by the `--compress` or `-C` options of the `mysql` command.
- Some SQL statements and functions can change the connection state without causing pinning. For the most current pinning behavior, see Avoiding pinning an RDS Proxy.
- RDS Proxy doesn't support the MariaDB `auth_ed25519` plugin.
- RDS Proxy doesn't support Transport Layer Security (TLS) version 1.3 for MariaDB databases.
- Database connections processing a `GET DIAGNOSTIC` command might return inaccurate information when RDS Proxy reuses the same database connection to run another query. This can happen when RDS Proxy multiplexes database connections. For more information, see Overview of RDS Proxy concepts.
- RDS Proxy currently doesn't support the `caching_sha2_password` option for `ClientPasswordAuthType` for MariaDB.

> **Important**
>
> For proxies associated with MariaDB databases, don't set the configuration parameter `sql_auto_is_null` to `true` or a nonzero value in the initialization query. Doing so might cause incorrect application behavior.

Additional limitations for RDS for Microsoft SQL Server

The following additional limitations apply to RDS Proxy with RDS for Microsoft SQL Server databases:

- The number of Secrets Manager secrets that you need to create for a proxy depends on the collation that your DB instance uses. For example, suppose that your DB instance uses case-sensitive collation. If your application accepts both "Admin" and "admin," then your proxy needs two separate secrets. For more information about collation in SQL Server, see the Microsoft SQL Server documentation.
- RDS Proxy doesn't support connections that use Active Directory.
- You can't use IAM authentication with clients that don't support token properties. For more information, see Considerations for connecting to Microsoft SQL Server.

- The results of `@@IDENTITY`, `@@ROWCOUNT`, and `SCOPE_IDENTITY` aren't always accurate. As a work-around, retrieve their values in the same session statement to ensure that they return the correct information.
- If the connection uses multiple active result sets (MARS), RDS Proxy doesn't run the initialization queries. For information about MARS, see the Microsoft SQL Server documentation.
- Currently, RDS Proxy does not support RDS for SQL Server DB instances that run on major version *SQL Server 2022*.
- RDS Proxy does not support RDS for SQL Server DB instances that run on major version *SQL Server 2014*.
- RDS Proxy does not support client applications that can't handle multiple response messages in one TLS record.
- RDS Proxy does not support end-to-end IAM authentication for RDS for SQL Server.

Additional limitations for RDS for MySQL

The following additional limitations apply to RDS Proxy with RDS for MySQL databases:

- RDS Proxy support for `caching_sha2_password` authentication requires a secure (TLS) connection.
- RDS Proxy support for `caching_sha2_password` is known to have compatibility issues with certain go-sql driver versions.
- When using the MySQL 8.4 C driver, the `mysql_stmt_bind_named_param` API might form malformed packets if parameter count exceeds placeholder count in a prepared statements. This results in incorrect responses. For more information, see MySQL bug report.
- Currently, all proxies listen on port 3306 for MySQL. The proxies still connect to your database using the port that you specified in the database settings.
- You can't use RDS Proxy with self-managed MySQL databases in EC2 instances.
- You can't use RDS Proxy with an RDS for MySQL DB instance that has the `read_only` parameter in its DB parameter group set to 1.
- RDS Proxy doesn't support MySQL compressed mode. For example, it doesn't support the compression used by the `--compress` or `-C` options of the `mysql` command.
- Database connections processing a `GET DIAGNOSTIC` command might return inaccurate information when RDS Proxy reuses the same database connection to run another query. This can happen when RDS Proxy multiplexes database connections.

- Some SQL statements and functions such as `SET LOCAL` can change the connection state without causing pinning. For the most current pinning behavior, see Avoiding pinning an RDS Proxy.
- Using the `ROW_COUNT()` function in a multi-statement query is not supported.
- RDS Proxy does not support client applications that can't handle multiple response messages in one TLS record.
- RDS Proxy does not support the MySQL dual passwords.
- RDS Proxy might not work as expected when you configure the `init_connect` parameter in your RDS DB parameter group to set session state variables. Instead, set the initialization query for your proxy to run session initialization statements when using proxy to connect to your database.

> **Important**
>
> For proxies associated with MySQL databases, don't set the configuration parameter `sql_auto_is_null` to `true` or a nonzero value in the initialization query. Doing so might cause incorrect application behavior.

Additional limitations for RDS for PostgreSQL

The following additional limitations apply to RDS Proxy with RDS for PostgreSQL databases:

- RDS Proxy doesn't support session pinning filters for PostgreSQL.
- Currently, all proxies listen on port 5432 for PostgreSQL.
- For PostgreSQL, RDS Proxy doesn't currently support canceling a query from a client by issuing a `CancelRequest`. This is the case, for example, when you cancel a long-running query in an interactive psql session by using Ctrl+C.
- The results of the PostgreSQL function lastval aren't always accurate. As a work-around, use the INSERT statement with the `RETURNING` clause.
- RDS Proxy currently doesn't support streaming replication mode.
- With RDS for PostgreSQL 16, modifications to the `scram_iterations` value exclusively impact the authentication process between the proxy and the database. Specifically, if you configure `ClientPasswordAuthType` to `scram-sha-256`, any customizations made to the

`scram_iterations` value doesn't influence client-to-proxy password authentication. Instead, the iteration value for client-to-proxy password authentication is fixed at 4096.

- The default `postgres` database must exist on the RDS for PostgreSQL instance for RDS Proxy to function. Don't delete this database even if your application uses different databases.
- If you use `ALTER ROLE` to change the user role with `SET ROLE`, subsequent connections as that user to the proxy might not use this role setting, if those connections encounter pinning. To prevent this, when using proxy, use `SET ROLE` in the initialization query of the proxy. For more information, see **Initialization query** in Creating a proxy for Amazon RDS.

> **Important**
>
> For existing proxies with PostgreSQL databases, if you modify the database authentication to use `SCRAM` only, the proxy becomes unavailable for up to 60 seconds. To avoid the issue, do one of the following:
>
> - Ensure that the database allows both `SCRAM` and `MD5` authentication.
> - To use only `SCRAM` authentication, create a new proxy, migrate your application traffic to the new proxy, then delete the proxy previously associated with the database.

Planning where to use RDS Proxy

You can determine which of your DB instances, clusters, and applications might benefit the most from using RDS Proxy. To do so, consider these factors:

- Any DB instance that encounters "too many connections" errors is a good candidate for associating with a proxy. This is often characterized by a high value of the `ConnectionAttempts` CloudWatch metric. The proxy enables applications to open many client connections, while the proxy manages a smaller number of long-lived connections to the DB instance .
- For DB instances that use smaller AWS instance classes, such as T2 or T3, using a proxy can help avoid out-of-memory conditions. It can also help reduce the CPU overhead for establishing connections. These conditions can occur when dealing with large numbers of connections.
- You can monitor certain Amazon CloudWatch metrics to determine whether a DB instance is approaching certain types of limit. These limits are for the number of connections and the memory associated with connection management. You can also monitor certain CloudWatch

metrics to determine whether a DB instance is handling many short-lived connections. Opening and closing such connections can impose performance overhead on your database. For information about the metrics to monitor, see Monitoring RDS Proxy metrics with Amazon CloudWatch.

- AWS Lambda functions can also be good candidates for using a proxy. These functions make frequent short database connections that benefit from connection pooling offered by RDS Proxy. You can take advantage of any IAM authentication you already have for Lambda functions, instead of managing database credentials in your Lambda application code.
- Applications that typically open and close large numbers of database connections and don't have built-in connection pooling mechanisms are good candidates for using a proxy.
- Applications that keep a large number of connections open for long periods are typically good candidates for using a proxy. Applications in industries such as software as a service (SaaS) or ecommerce often minimize the latency for database requests by leaving connections open. With RDS Proxy, an application can keep more connections open than it can when connecting directly to the DB instance.
- You might not have adopted IAM authentication and Secrets Manager due to the complexity of setting up such authentication for all DB instances . The proxy can enforce the authentication policies for client connections for particular applications. You can take advantage of any IAM authentication you already have for Lambda functions, instead of managing database credentials in your Lambda application code.
- RDS Proxy can help make applications more resilient and transparent to database failures. RDS Proxy bypasses Domain Name System (DNS) caches to reduce failover times by up to 66% for Amazon RDS Multi-AZ DB instances. RDS Proxy also automatically routes traffic to a new database instance while preserving application connections. This makes failovers more transparent for applications.

RDS Proxy concepts and terminology

You can simplify connection management for your Amazon RDS DB instances by using RDS Proxy.

RDS Proxy handles the network traffic between the client application and the database. It does so in an active way first by understanding the database protocol. It then adjusts its behavior based on the SQL operations from your application and the result sets from the database.

RDS Proxy reduces the memory and CPU overhead for connection management on your database. The database needs less memory and CPU resources when applications open many simultaneous

connections. It also doesn't require logic in your applications to close and reopen connections that stay idle for a long time. Similarly, it requires less application logic to reestablish connections in case of a database problem.

The infrastructure for RDS Proxy is highly available and deployed over multiple Availability Zones (AZs). The computation, memory, and storage for RDS Proxy are independent of your RDS DB instance. This separation helps lower overhead on your database servers, so that they can devote their resources to serving database workloads. The RDS Proxy compute resources are serverless, automatically scaling based on your database workload.

Topics

- Overview of RDS Proxy concepts
- Connection pooling
- RDS Proxy security
- Failover
- Transactions

Overview of RDS Proxy concepts

RDS Proxy handles the infrastructure to perform connection pooling and the other features described in the sections that follow. You see the proxies represented in the RDS console on the **Proxies** page.

Each proxy handles connections to a single RDS DB instance. The proxy automatically determines the current writer instance for RDS Multi-AZ DB instance or cluster.

The connections that a proxy keeps open and available for your database applications to use make up the *connection pool*.

By default, RDS Proxy can reuse a connection after each transaction in your session. This transaction-level reuse is called *multiplexing*. When RDS Proxy temporarily removes a connection from the connection pool to reuse it, that operation is called *borrowing* the connection. When it's safe to do so, RDS Proxy returns that connection to the connection pool.

In some cases, RDS Proxy can't be sure that it's safe to reuse a database connection outside of the current session. In these cases, it keeps the session on the same connection until the session ends. This fallback behavior is called *pinning*.

A proxy has a default endpoint. You connect to this endpoint when you work with an Amazon RDS DB instance. You do so instead of connecting to the read/write endpoint that connects directly to the instance . For RDS DB clusters, you can also create additional read/write and read-only endpoints. For more information, see Overview of proxy endpoints.

For example, you can still connect to the cluster endpoint for read/write connections without connection pooling. You can still connect to the reader endpoint for load-balanced read-only connections. You can still connect to the instance endpoints for diagnosis and troubleshooting of specific DB instances within a cluster. If you use other AWS services such as AWS Lambda to connect to RDS databases, change their connection settings to use the proxy endpoint. For example, you specify the proxy endpoint to allow Lambda functions to access your database while taking advantage of RDS Proxy functionality.

Each proxy contains a target group. This *target group* embodies the RDS DB instance that the proxy can connect to. The RDS DB instance associated with a proxy are called the *targets* of that proxy. For convenience, when you create a proxy through the console, RDS Proxy also creates the corresponding target group and registers the associated targets automatically.

An *engine family* is a related set of database engines that use the same DB protocol. You choose the engine family for each proxy that you create.

Connection pooling

Each proxy performs connection pooling separately for the writer and reader instance of its associated RDS database . *Connection pooling* is an optimization that reduces the overhead associated with opening and closing connections and with keeping many connections open simultaneously. This overhead includes memory needed to handle each new connection. It also involves CPU overhead to close each connection and open a new one. Examples include Transport Layer Security/Secure Sockets Layer (TLS/SSL) handshaking, authentication, negotiating capabilities, and so on. Connection pooling simplifies your application logic. You don't need to write application code to minimize the number of simultaneous open connections.

Each proxy also performs connection multiplexing, also known as connection reuse. With *multiplexing*, RDS Proxy performs all the operations for a transaction using one underlying database connection. RDS then can use a different connection for the next transaction. You can open many simultaneous connections to the proxy, and the proxy keeps a smaller number of connections open to the DB instance or cluster. Doing so further minimizes the memory overhead for connections on the database server. This technique also reduces the chance of "too many connections" errors.

RDS Proxy security

RDS Proxy uses the existing RDS security mechanisms such as TLS/SSL and AWS Identity and Access Management (IAM). For general information about those security features, see Security in Amazon RDS . Also, make sure to familiarize yourself with how RDS work with authentication, authorization, and other areas of security.

RDS Proxy can act as an additional layer of security between client applications and the underlying database. For example, you can connect to the proxy using TLS 1.3, even if the underlying DB instance supports an older version of TLS. You can connect to the proxy using an IAM role even if the proxy connects to the database using the database user and password authentication method. By using this technique, you can enforce strong authentication requirements for database applications without a costly migration effort for the DB instances themselves.

You can use the following methods of authentication with RDS Proxy:

- **Database credentials**
- **Standard IAM authentication**
- **End-to-end IAM authentication**

Using IAM with RDS Proxy

RDS Proxy offers two methods of IAM authentication:

- **Standard IAM authentication**: Enforce IAM authentication for connections to your proxy while the proxy connects to the database using credentials stored in Secrets Manager. This enforces IAM authentication for database access even if the databases use native password authentication. The proxy retrieves the database credentials from Secrets Manager and handles the authentication to the database on behalf of your application.
- **End-to-end IAM authentication**: Enforces IAM authentication for connections directly from your applications to your database through the proxy. End-to-end IAM authentication simplifies your security configuration and avoids database credential management in Secrets Manager. This additional layer of security enforces IAM-based access control from the client application to the database.

To use standard IAM authentication, configure your proxy to use Secrets Manager secrets for authentication and enable IAM authentication for client connections. Your applications

authenticate to the proxy using IAM, while the proxy authenticates to the database using the credentials retrieved from Secrets Manager.

To use end-to-end IAM authentication, configure your proxy to use IAM authentication when setting the default authentication scheme when creating or modifying your proxy.

For end-to-end IAM authentication, you must update the IAM role associated with the proxy to grant the `rds-db:connect` permission. With end-to-end IAM authentication, this eliminates the need to register individual database users with the proxy through Secrets Manager secrets.

Using TLS/SSL with RDS Proxy

You can connect to RDS Proxy using the TLS/SSL protocol.

> **Note**
>
> RDS Proxy uses certificates from the AWS Certificate Manager (ACM). If you are using RDS Proxy, you don't need to download Amazon RDS certificates or update applications that use RDS Proxy connections.

To enforce TLS for all connections between the proxy and your database, you can specify a setting **Require Transport Layer Security** when you create or modify a proxy in the AWS Management Console.

RDS Proxy can also ensure that your session uses TLS/SSL between your client and the RDS Proxy endpoint. To have RDS Proxy do so, specify the requirement on the client side. SSL session variables are not set for SSL connections to a database using RDS Proxy.

- For RDS for MySQL, specify the requirement on the client side with the `--ssl-mode` parameter when you run the `mysql` command.
- For Amazon RDS PostgreSQL, specify `sslmode=require` as part of the `conninfo` string when you run the `psql` command.

RDS Proxy supports TLS protocol version 1.0, 1.1, 1.2, and 1.3. You can connect to the proxy using a higher version of TLS than you use in the underlying database.

By default, client programs establish an encrypted connection with RDS Proxy, with further control available through the `--ssl-mode` option. From the client side, RDS Proxy supports all SSL modes.

For the client, the SSL modes are the following:

PREFERRED

SSL is the first choice, but it isn't required.

DISABLED

No SSL is allowed.

REQUIRED

Enforce SSL.

VERIFY_CA

Enforce SSL and verify the certificate authority (CA).

VERIFY_IDENTITY

Enforce SSL and verify the CA and CA hostname.

When using a client with `--ssl-mode VERIFY_CA` or `VERIFY_IDENTITY`, specify the `--ssl-ca` option pointing to a CA in `.pem` format. For the `.pem` file to use, download all root CA PEMs from Amazon Trust Services and place them into a single `.pem` file.

RDS Proxy uses wildcard certificates, which apply to both a domain and its subdomains. If you use the `mysql` client to connect with SSL mode `VERIFY_IDENTITY`, currently you must use the MySQL 8.0-compatible `mysql` command.

Failover

Failover is a high-availability feature that replaces a database instance with another one when the original instance becomes unavailable. A failover might happen because of a problem with a database instance. It might also be part of normal maintenance procedures, such as during a database upgrade. Failover applies to RDS DB instances in a Multi-AZ configuration.

Connecting through a proxy makes your applications more resilient to database failovers. When the original DB instance becomes unavailable, RDS Proxy connects to the standby database without dropping idle application connections. This helps speed up and simplify the failover process. This is less disruptive to your application than a typical reboot or database problem.

Without RDS Proxy, a failover involves a brief outage. During the outage, you can't perform write operations on the database in failover. Any existing database connections are disrupted, and your

application must reopen them. The database becomes available for new connections and write operations when a read-only DB instance is promoted in place of one that's unavailable.

During DB failovers, RDS Proxy continues to accept connections at the same IP address and automatically directs connections to the new primary DB instance. Clients connecting through RDS Proxy are not susceptible to the following:

- Domain Name System (DNS) propagation delays on failover.
- Local DNS caching.
- Connection timeouts.
- Uncertainty about which DB instance is the current writer.
- Waiting for a query response from a former writer that became unavailable without closing connections.

For applications that maintain their own connection pool, going through RDS Proxy means that most connections stay alive during failovers or other disruptions. Only connections that are in the middle of a transaction or SQL statement are canceled. RDS Proxy immediately accepts new connections. When the database writer is unavailable, RDS Proxy queues up incoming requests.

For applications that don't maintain their own connection pools, RDS Proxy offers faster connection rates and more open connections. It offloads the expensive overhead of frequent reconnects from the database. It does so by reusing database connections maintained in the RDS Proxy connection pool. This approach is particularly important for TLS connections, where setup costs are significant.

Transactions

All the statements within a single transaction always use the same underlying database connection. The connection becomes available for use by a different session when the transaction ends. Using the transaction as the unit of granularity has the following consequences:

- Connection reuse can happen after each individual statement when the RDS for MySQL `autocommit` setting is turned on.
- Conversely, when the `autocommit` setting is turned off, the first statement you issue in a session begins a new transaction. For example, suppose that you enter a sequence of `SELECT`, `INSERT`, `UPDATE`, and other data manipulation language (DML) statements. In this case, connection reuse doesn't happen until you issue a `COMMIT`, `ROLLBACK`, or otherwise end the transaction.

- Entering a data definition language (DDL) statement causes the transaction to end after that statement completes.

RDS Proxy detects when a transaction ends through the network protocol used by the database client application. Transaction detection doesn't rely on keywords such as `COMMIT` or `ROLLBACK` appearing in the text of the SQL statement.

In some cases, RDS Proxy might detect a database request that makes it impractical to move your session to a different connection. In these cases, it turns off multiplexing for that connection the remainder of your session. The same rule applies if RDS Proxy can't be certain that multiplexing is practical for the session. This operation is called *pinning*. For ways to detect and minimize pinning, see Avoiding pinning an RDS Proxy.

Getting started with RDS Proxy

Use the information in the following pages to set up and manage Amazon RDS Proxy and set related security options. The security options control who can access each proxy and how each proxy connects to DB instances.

If you're new to RDS Proxy, we recommend following the pages in the order that we present them.

Topics

- Setting up network prerequisites for RDS Proxy
- Setting up database credentials for RDS Proxy
- Configuring IAM authentication for RDS Proxy
- Creating a proxy for Amazon RDS
- Viewing a proxy
- Connecting to a database through RDS Proxy

Setting up network prerequisites for RDS Proxy

Using RDS Proxy requires you to have a common virtual private cloud (VPC) between your RDS DB instance and RDS Proxy. This VPC should have a minimum of two subnets that are in different Availability Zones. Your account can either own these subnets or share them with other accounts. For information about VPC sharing, see Work with shared VPCs.

For IPv6 support, additional network configuration is required:

- **IPv6 endpoint network types** – Your VPC and subnets must be configured to support IPv6. This includes having IPv6 CIDR blocks assigned to your VPC and subnets.
- **Dual-stack endpoint network types** – Your VPC and subnets must support both IPv4 and IPv6 addressing.
- **IPv6 target connection network types** – Your database must be configured for dual-stack mode to support IPv6 connections from the proxy.

Your client application resources such as Amazon EC2, Lambda, or Amazon ECS can be in the same VPC as the proxy. Or they can be in a separate VPC from the proxy. If you successfully connected to any RDS DB instances , you already have the required network resources.

Topics

- Getting information about your subnets
- Planning for IP address capacity

Getting information about your subnets

To create a proxy, you must provide the subnets and the VPC that the proxy operates within. The following Linux example shows AWS CLI commands that examine the VPCs and subnets owned by your AWS account. In particular, you pass subnet IDs as parameters when you create a proxy using the CLI.

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

The following Linux example shows AWS CLI commands to determine the subnet IDs corresponding to a specific RDS DB instance. Find the VPC ID for the DB instance. Examine the VPC to find its subnets. The following Linux example shows how.

```
$ #From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
 IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]||[0]||[0]||[Subnets]||[0]||[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
...
```

```
$ #From the DB instance, find the VPC.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]||[0]||[0].VpcId' --output text
```

```
my_vpc_id
```

```
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].
[SubnetId]' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

Planning for IP address capacity

An RDS Proxy automatically adjusts its capacity as needed based on the size and number of DB instances registered with it. Certain operations might also require additional proxy capacity such as increasing the size of a registered database or internal RDS Proxy maintenance operations. During these operations, your proxy might need more IP addresses to provision the extra capacity. These additional addresses allow your proxy to scale without affecting your workload. A lack of free IP addresses in your subnets prevents a proxy from scaling up. This can lead to higher query latencies or client connection failures. RDS notifies you through event `RDS-EVENT-0243` when there aren't enough free IP addresses in your subnets. For information about this event, see Working with RDS Proxy events.

> **Note**
>
> RDS Proxy doesn't consume more than 215 IP addresses for each proxy in a VPC.

Reserve the following minimum numbers of free IP addresses in your subnets for your proxy, based on DB instance class sizes.

DB instance class	Minimum free IP addresses
db.*.xlarge or smaller	10
db.*.2xlarge	15
db.*.4xlarge	25
db.*.8xlarge	45
db.*.12xlarge	60
db.*.16xlarge	75
db.*.24xlarge	110

These recommended numbers of IP addresses are estimates for a proxy with only the default endpoint. A proxy with additional endpoints or read replicas might need more free IP addresses. For each additional endpoint, we recommend that you reserve three more IP addresses. For each read replica, we recommend that you reserve additional IP addresses as specified in the table based on that read replica's size.

Setting up database credentials for RDS Proxy

RDS Proxy in Amazon RDS uses AWS Secrets Manager to store and manage database credentials securely. Instead of embedding credentials in your application, you associate a proxy with a Secrets Manager secret that contains the necessary authentication details. You create a separate Secrets Manager secret for each database user account that the proxy connects to on the RDS DB instance.

Alternatively, you can configure RDS Proxy to use end-to-end IAM authentication, which eliminates the need to store database credentials in Secrets Manager. RDS Proxy uses IAM authentication for both client-to-proxy and proxy-to-database connections. This provides a fully integrated IAM-based authentication solution that doesn't require managing secrets or passwords. For information about adding a new IAM DB user, see Creating a database account using IAM authentication.

Topics

- Creating secrets to use with RDS Proxy

Creating secrets to use with RDS Proxy

Before you create a proxy, you must first create at least one secret that stores your database credentials.

Console

To create a secret

1. Open the Secrets Manager console at https://console.aws.amazon.com/secretsmanager/.
2. Choose **Store a new secret**.
3. Choose **Credentials for Amazon RDS database**.
4. Enter a user name and password. The credentials that you enter must match the credentials of a database user that exists in the associated RDS database. RDS Proxy uses these credentials to authenticate and establish connections to the database on behalf of applications.

   If there's a mismatch, you can update the secret to match the database password. Until you update the secret, attempts to connect through the proxy using that secret fail, but connections using other valid secrets still work.

   > **Note**
   >
   > For RDS for SQL Server, RDS Proxy requires a case-sensitive secret in Secrets Manager, regardless of the DB instance collation settings. If your application allows usernames with different capitalizations, such as "Admin" and "admin," you must create separate secrets for each. RDS Proxy doesn't support case-insensitive username authentication between the client and proxy.
   > For more information about collation in SQL Server, see the Microsoft SQL Server documentation.

5. For **Database**, select the Amazon RDS database that the secret will access.
6. Fill in other settings for the secret, then choose **Store**. For comprehensive instructions, see Creating an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

AWS CLI

When you create a proxy through the AWS CLI, you specify the Amazon Resource Names (ARNs) of the corresponding secrets. You do so for all the DB user accounts that the proxy can access. In the AWS Management Console, you choose the secrets by their descriptive names.

- To create a Secrets Manager secret for use with RDS Proxy, use the create-secret command:

```
aws secretsmanager create-secret \
  --name "secret_name" \
  --description "secret_description" \
  --region region_name \
  --secret-string '{"username":"db_user","password":"db_user_password"}'
```

- You can also create a custom key to encrypt your Secrets Manager secret. The following command creates an example key.

```
aws kms create-key --description "test-key" --policy '{
  "Id":"kms-policy",
  "Version": "2012-10-17",
  "Statement":
    [
      {
        "Sid":"Enable IAM User Permissions",
        "Effect":"Allow",
        "Principal":{"AWS":"arn:aws:iam::account_id:root"},
        "Action":"kms:*","Resource":"*"
      },
      {
        "Sid":"Allow access for Key Administrators",
        "Effect":"Allow",
        "Principal":
          {
            "AWS":
              ["$USER_ARN","arn:aws:iam:account_id::role/Admin"]
          },
        "Action":
          [
            "kms:Create*",
            "kms:Describe*",
            "kms:Enable*",
            "kms:List*",
            "kms:Put*",
```

```
            "kms:Update*",
            "kms:Revoke*",
            "kms:Disable*",
            "kms:Get*",
            "kms:Delete*",
            "kms:TagResource",
            "kms:UntagResource",
            "kms:ScheduleKeyDeletion",
            "kms:CancelKeyDeletion"
          ],
        "Resource":"*"
      },
      {
        "Sid":"Allow use of the key",
        "Effect":"Allow",
        "Principal":{"AWS":"$ROLE_ARN"},
        "Action":["kms:Decrypt","kms:DescribeKey"],
        "Resource":"*"
      }
    ]
}'
```

For example, the following commands create Secrets Manager secrets for two database users:

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

To create these secrets encrypted with your custom AWS KMS key, use the following commands:

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}' \
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}' \
```

```
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id
```

To see the secrets owned by your AWS account, use the list-secrets command:

```
aws secretsmanager list-secrets
```

When you create a proxy using the CLI, you pass the Amazon Resource Names (ARNs) of one or more secrets to the `--auth` parameter. The following example shows how to prepare a report with only the name and ARN of each secret owned by your AWS account. This example uses the `--output table` parameter that is available in AWS CLI version 2. If you are using AWS CLI version 1, use `--output text` instead.

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

To confirm that the secret contains the correct credentials in the proper format, use the get-secret-value command. Replace *your_secret_name* with the secret's short name or ARN.

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

The output contains a line with a JSON-encoded value similar to the following:

```
...
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
...
```

Configuring IAM authentication for RDS Proxy

To set up AWS Identity and Access Management (IAM) authentication for RDS Proxy in Amazon RDS, create and configure an IAM policy that grants the necessary permissions.

This topic provides the steps to configure IAM authentication for RDS Proxy, including creating the required IAM policy and attaching it to an IAM role.

> **Tip**
>
> This procedure is only necessary if you want to create your own IAM role. Otherwise, RDS can automatically create the required role when you set up the proxy, so you can skip these steps.

Prerequisites

Before you set up IAM authentication for RDS Proxy, make sure that you have the following:

- **AWS Secrets Manager** – At least one stored secret that contains database credentials. For instructions to create secrets, see the section called "Setting up database credentials".

  This is not required if you are using end-to-end IAM authentication.
- **IAM permissions** – An IAM role or user with permissions to create and manage IAM policies, roles, and secrets in AWS Secrets Manager.

Creating an IAM policy for end-to-end IAM authentication

When using end-to-end IAM authentication, RDS Proxy connects to your database using IAM authentication instead of retrieving credentials from Secrets Manager. This requires configuring your IAM role with `rds-db:connect` permissions for the database accounts you want to use with the proxy.

To authenticate your RDS Proxy to the database using IAM, create an IAM role with a policy that grants the necessary database connection permissions.

Console

To create a role for end-to-end IAM authentication with your proxy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Create a permissions policy for the role. For general steps, see Create IAM policies (console).

   Paste this policy into the JSON editor and make the following changes:

   - Substitute your own account ID.
   - Substitute `us-east-2` with the where the proxy must reside.
   - Substitute the database resource IDs and user names with the ones you want to use. The resource ID format differs between RDS instances and Aurora clusters.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "rds-db:connect",
            "Resource": [
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_1",
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_2"
            ]
        }
    ]
}
```

3. Create the role and attach the permissions policy to it. For general steps, see Create a role to delegate permissions to an AWS service.

   For the **Trusted entity type**, choose **AWS service**. Under **Use case**, select **RDS** and choose **RDS - Add Role to Database** for the use case.

4. For **Permissions policies**, choose the policy that you created.
5. For **Select trusted entities**, enter the following trust policy for the role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS CLI

To create the role using the AWS CLI, send the following request:

```
aws iam create-role \
  --role-name my_e2e_iam_role_name \
```

```
  --assume-role-policy-document '{"Version":"2012-10-17",
     "Statement":[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'
```

Then, attach the policy to the role:

```
aws iam put-role-policy \
  --role-name my_e2e_iam_role_name \
  --policy-name e2e_iam_db_connect_policy \
  --policy-document '{

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds-db:connect",
            "Resource": [
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_1",
                "arn:aws:rds-db:us-
east-2:account_id:dbuser:db_instance_resource_id/db_user_name_2"
            ]
        }
    ]
}'
```

With the IAM role and permissions configured for end-to-end IAM authentication, you can now create a proxy with `DefaultAuthScheme` set to `IAM_AUTH`. This proxy directly authenticates to the database using IAM without requiring Secrets Manager secrets. For instructions, see the section called “Creating a proxy”.

When using end-to-end IAM authentication, ensure that your database users are configured for IAM authentication as described in Creating a database account using IAM authentication.

Creating an IAM policy for Secrets Manager access

To allow RDS Proxy to retrieve database credentials from Secrets Manager, create an IAM role with a policy that grants the necessary permissions.

Console

To create a role to access your secrets for use with your proxy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Create a permissions policy for the role. For general steps, see Create IAM policies (console).

   Paste this policy into the JSON editor and make the following changes:

   - Substitute your own account ID.
   - Substitute `us-east-2` with the Region where the proxy will reside.
   - Substitute the secret names with the ones you created. For more information, see Specifying KMS keys in IAM policy statements.
   - Substitue the KMS key ID with the one you used to encrypt the Secrets Manager secrets, either the default key or your own key.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-
east-2:111122223333:secret:secret_name_1",
                "arn:aws:secretsmanager:us-
east-2:111122223333:secret:secret_name_2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:111122223333:key/key_id",
            "Condition": {
                "StringEquals": {
```

```
                    "kms:ViaService": "secretsmanager.us-
east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

3. Create the role and attach the permissions policy to it. For general steps, see Create a role to delegate permissions to an AWS service.

   For the **Trusted entity type**, choose **AWS service**. Under **Use case**, select **RDS** and choose **RDS - Add Role to Database** for the use case.

4. For **Permissions policies**, choose the policy that you created.

5. For **Select trusted entities**, enter the following trust policy for the role:

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

AWS CLI

To create the role using the AWS CLI, send the following request:

```
aws iam create-role \
  --role-name my_role_name \
```

```
  --assume-role-policy-document '{"Version": "2012-10-17",
      "Statement":[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'
```

Then, attach the policy to the role:

```
aws iam put-role-policy \
  --role-name my_role_name \
  --policy-name secret_reader_policy \
  --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}'
```

With the IAM role and permissions configured, you can now create a proxy and associate it with this role. This allows the proxy to retrieve database credentials securely from AWS Secrets Manager and enable IAM authentication for your applications. For instructions, see the section called “Creating a proxy”.

Creating a proxy for Amazon RDS

You can associate a proxy with an RDS for MariaDB, RDS for Microsoft SQL Server, RDS for MySQL, or RDS for PostgreSQL DB instance.

Console

To create a proxy

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Proxies**.
3. Choose **Create proxy**.
4. Configure the following settings for your proxy.

Setting	Description
Engine family	The database network protocol the proxy recognizes when it interprets network traffic to and from the database. **Note** To use RDS for PostgreSQL, make sure to retain the `postgres` database in your instance. See Troubleshooting deleted `postgres` database.
Proxy identifier	A name that is unique within your AWS account ID and current AWS Region.
Idle client connection timeout	The proxy closes a client connection if it remains idle for a set period. By default, this is 1,800 seconds (30 minutes). A connection is idle when the application doesn't submit a new request within the specified time after completing the previous request. The proxy keeps the underlying database connection open and returns it to the connection pool, making it available for new client connections.

Setting	Description
	To proactively remove stale connections, reduce the idle client connection timeout. To minimize connection costs during workload spikes, increase the timeout.
Database	The RDS DB instance to access through this proxy. The list only includes DB instances and clusters with compatible database engines, engine versions, and other settings. If the list is empty, create a new DB instance or cluster that's compatible with RDS Proxy. To do so, follow the procedure in Creating an Amazon RDS DB instance. Then, try creating the proxy again.
Connection pool maximum connections	A value between 1 and 100 to define the percentage of the `max_connections` limit that RDS Proxy can use. If you only intend to use one proxy with this DB instance or cluster, set this value to 100. For more information about how RDS Proxy uses this setting, see the section called “MaxConnectionsPercent”.

Setting	Description
Session pinning filters	Prevents RDS Proxy from pinning certain detected session states, which bypasses default safety measures for multiplexing connections. Currently, PostgreSQL doesn't support this setting, and the only available option is `EXCLUDE_VARIABLE_SETS`. Enabling it might cause session variables from one connection to affect others, leading to errors or correctness issues if queries rely on session variables set outside the current transaction. Use this option only after confirming that your applications can safely share database connections. The following patterns are considered safe: • `SET` statements where there is no change to the effective session variable value. In other words, there is no change to the session variable. • You change the session variable value and execute a statement in the same transaction. For more information, see Avoiding pinning an RDS Proxy.
Connection borrow timeout	If you expect the proxy to use all available database connections, set the wait time before it returns a timeout error. You can specify up to five minutes. This setting applies only when the proxy has reached the maximum number of connections and all are in use.

Setting	Description
Initialization query	(Optional) Add an initialization query, or modify the current one. You can specify one or more SQL statements for the proxy to run when opening each new database connection. The setting is typically used with SET statements to make sure that each connection has identical settings. Make sure that the query you add is valid. To include multiple variables in a single SET statement, use comma separators. For example: `SET variable1 =value1, variable2 =value2` For multiple statements, use semicolons as the separator. **Important** Since you can access initialization query as part of target group configuration, it is not protected by authentication or cryptographic methods. Anyone with access to view or manage your proxy target group configuration can view the initialization query. You should not add sensitive data, such as passwords or long-lived encryption keys, to this option.
AWS Identity and Access Management (IAM) role	An IAM role with permission to access the Secrets Manager secrets, which represent the credentials for database user accounts that the proxy can use. Alternatively, you can create a new IAM role from the AWS Management Console.

Setting	Description
Secrets Manager secrets	Create or choose Secrets Manager secrets representing the credentials for database users accounts that can use the proxy. When **Default authentication scheme** is set to **None**, this field is required. When **Default authentication scheme** is set to **IAM authentication**, this field becomes optional and is marked as such in the console. You can choose one or more secrets from the dropdown or create a new secret using the **Create a new secret** link.
Client authentication type	The type of authentication the proxy uses for connections from clients. Your choice applies to all Secrets Manager secrets that you associate with this proxy. If you need to specify a different client authentication type for each secret, create your proxy by using the AWS CLI or the API instead. Specify this option only when your client connection uses database credentials for authentication.
IAM authentication	Specify **Required**, **Allowed**, or **Not Allowed** for IAM authentic ation for connections to your proxy. The **Allowed** option is only valid for proxies for RDS for SQL Server. Your choice applies to all Secrets Manager secrets that you associate with this proxy. If you need to specify a different IAM authentication for each secret, create your proxy by using the AWS CLI or the API instead.

Setting	Description
Default authentication scheme	Choose the default type of authentication that the proxy uses for client connections to the proxy and the connections from the proxy to the underlying database. You have the following options: • **None** (default) - The proxy retrieves database credentials from Secrets Manager secrets. • **IAM authentication** - The proxy uses IAM authentication to connect to the database, enabling end-to-end IAM authentic ation. When you select **IAM authentication**, an information alert appears reminding you to enable IAM database authentication for the databases in the target group configuration. **Note** This option is supported for MySQL, PostgreSQL, and MariaDB engine families only.

Setting	Description
Database accounts for IAM authentication	This field appears only when **Default authentication scheme** is set to **IAM authentication** and **Identity and access management (IAM) role** is set to **Create IAM role**. Name the database user accounts for the proxy to use with IAM authentication. This is a required field. Specify multiple accounts by: • Typing a database user name to add it as a tag • Using specific database user names (e.g., `db_user`, `jane_doe`) • Using wildcard patterns for multiple users (e.g., `db_test_*`) Each account appears as a removable tag that you can delete by clicking the X icon. The console uses these values to create the appropriate `rds-db:connect` permissions in the IAM role policy.
Require Transport Layer Security	Enforces TLS/SSL for all client connections. The proxy uses the same encryption setting for its connection to the underlyin g database, whether the client connection is encrypted or unencrypted.

Setting	Description
Target connection network type	The IP version that the proxy uses to connect to the target database. Choose from the following options: • **IPv4** – The proxy connects to the database using IPv4 addresses. • **IPv6** – The proxy connects to the database using IPv6 addresses. The default is IPv4. To use IPv6, your database must support dual-stack mode. Dual-stack mode is not available for target connections.
Endpoint network type	The IP version for the proxy endpoint that clients use to connect to the proxy. Choose from the following options: • **IPv4** – The proxy endpoint uses IPv4 addresses only. • **IPv6** – The proxy endpoint uses IPv6 addresses only. • **Dual-stack** – The proxy endpoint supports both IPv4 and IPv6 addresses. The default is IPv4. To use IPv6 or dual-stack, your VPC and subnets must be configured to support the selected network type.
Subnets	This field is prepopulated with all subnets associated with your VPC. You can remove any subnets not needed for the proxy, but you must leave at least two subnets. For IPv6 or dual-stac k endpoint network types, ensure that the selected subnets support the chosen network type.

Setting	Description
VPC security group	Choose an existing VPC security group or create a new one from the AWS Management Console. Configure the inbound rules to allow your applications to access the proxy and the outbound rules to permit traffic from your database targets. **Note** The security group must allow connections from the proxy to the database. It serves both for ingress from your applications to the proxy and egress from the proxy to the database. For example, if you use the same security group for both the database and the proxy, make sure that resources within that security group can communicate with each other. When you use a shared VPC, avoid using the default security group for the VPC or one associated with another account. Instead, select a security group that belongs to your account. If none exists, create one. For more information, see Work with shared VPCs. RDS deploys a proxy across multiple Availability Zones to ensure high availability. To enable cross-AZ communication, the network access control list (ACL) for your proxy subnet must allow egress on the engine port and ingress on all ports. For more information about network ACLs, see Control traffic to subnets using network ACLs. If the network ACL for your proxy and target are identical, you must add a **TCP** protocol ingress rule where the **Source** is set to the VPC CIDR. You must also add an engine port specific **TCP** protocol egress rule where the **Destination** is set to the VPC CIDR.

Setting	Description
Activate enhanced logging	Enable this setting to troubleshoot proxy compatibility or performance issues. When enabled, RDS Proxy logs detailed performance information to help you debug SQL behavior or proxy connection performance and scalability. Only enable this setting for debugging and ensure proper security measures are in place to protect sensitive informati on in the logs. To minimize overhead, RDS Proxy automatically disables this setting 24 hours after activation. Use it temporari ly to troubleshoot specific issues.

5. Choose **Create proxy**.

AWS CLI

To create a proxy by using the AWS CLI, call the create-db-proxy command with the following required parameters:

- `--db-proxy-name`
- `--engine-family`
- `--role-arn`
- `--vpc-subnet-ids`

The `--engine-family` value is case-sensitive.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } \
    --role-arn iam_role \
    --vpc-subnet-ids space_separated_list \
    [--default-auth-scheme { NONE | IAM_AUTH }] \
    [--auth ProxyAuthenticationConfig_JSON_string] \
```

```
    [--vpc-security-group-ids space_separated_list] \
    [--require-tls | --no-require-tls] \
    [--idle-client-timeout value] \
    [--debug-logging | --no-debug-logging] \
    [--endpoint-network-type { IPV4 | IPV6 | DUAL }] \
    [--target-connection-network-type { IPV4 | IPV6 }] \
    [--tags comma_separated_list]
```

For Windows:

```
aws rds create-db-proxy ^
    --db-proxy-name proxy_name ^
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } ^
    --role-arn iam_role ^
    --vpc-subnet-ids space_separated_list ^
    [--default-auth-scheme { NONE | IAM_AUTH }] ^
    [--auth ProxyAuthenticationConfig_JSON_string] ^
    [--vpc-security-group-ids space_separated_list] ^
    [--require-tls | --no-require-tls] ^
    [--idle-client-timeout value] ^
    [--debug-logging | --no-debug-logging] ^
    [--endpoint-network-type { IPV4 | IPV6 | DUAL }] ^
    [--target-connection-network-type { IPV4 | IPV6 }] ^
    [--tags comma_separated_list]
```

The following is an example of the JSON value for the `--auth` option. This example applies a different client authentication type to each secret.

```
[
  {
    "Description": "proxy description 1",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789123:secret/1234abcd-12ab-34cd-56ef-1234567890ab",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_SCRAM_SHA_256"
  },

  {
    "Description": "proxy description 2",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122223333:secret/1234abcd-12ab-34cd-56ef-1234567890cd",
```

```
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_MD5"

  },

  {
    "Description": "proxy description 3",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122221111:secret/1234abcd-12ab-34cd-56ef-1234567890ef",
    "IAMAuth": "REQUIRED"
  }

]
```

The `--endpoint-network-type` parameter specifies the IP version for the proxy endpoint that clients use to connect to the proxy. Valid values are:

- `IPV4` – The proxy endpoint uses IPv4 addresses only (default).
- `IPV6` – The proxy endpoint uses IPv6 addresses only.
- `DUAL` – The proxy endpoint supports both IPv4 and IPv6 addresses.

The `--target-connection-network-type` parameter specifies the IP version that the proxy uses to connect to the target database. Valid values are:

- `IPV4` – The proxy connects to the database using IPv4 addresses (default).
- `IPV6` – The proxy connects to the database using IPv6 addresses.

To use IPv6 or dual-stack endpoint network types, your VPC and subnets must be configured to support the selected network type. To use IPv6 target connection network type, your database must support dual-stack mode.

Tip

If you don't already know the subnet IDs to use for the `--vpc-subnet-ids` parameter, see Setting up network prerequisites for RDS Proxy for examples of how to find them.

Note

The security group must allow access to the database the proxy connects to. The same security group is used for ingress from your applications to the proxy, and for egress from the proxy to the database. For example, suppose that you use the same security group for your database and your proxy. In this case, make sure that you specify that resources in that security group can communicate with other resources in the same security group.
When using a shared VPC, you can't use the default security group for the VPC, or one that belongs to another account. Choose a security group that belongs to your account. If one doesn't exist, create one. For more information about this limitation, see Work with shared VPCs.

To create the right associations for the proxy, you also use the register-db-proxy-targets command. Specify the target group name `default`. RDS Proxy automatically creates a target group with this name when you create each proxy.

```
aws rds register-db-proxy-targets
    --db-proxy-name value
    [--target-group-name target_group_name]
    [--db-instance-identifiers space_separated_list]  # rds db instances, or
    [--db-cluster-identifiers cluster_id]        # rds db cluster (all instances)
```

RDS API

To create an RDS proxy, call the Amazon RDS API operation CreateDBProxy. You pass a parameter with the AuthConfig data structure.

RDS Proxy automatically creates a target group named `default` when you create each proxy. You associate an RDS DB instance with the target group by calling the function RegisterDBProxyTargets.

Important

When you select **IAM authentication** for the default authentication scheme:

- You must enable IAM database authentication on your target database instances or clusters before the proxy can successfully connect.

- If you choose **Create IAM role**, the **Database accounts for IAM authentication** field is required.
- If you select an existing IAM role, the console does not automatically update the role with database connection permissions. Check that the role has the necessary `rds-db:connect` permissions.

Viewing a proxy

After you create one or more RDS proxies, you can view and manage them in the AWS Management Console, the AWS CLI, or the RDS API. You can review their configuration details, monitor performance, and determine which proxies to modify or delete as needed.

To enable database applications to route traffic through a proxy, you must specify the proxy endpoint in the connection string.

Console

To view a proxy in the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Proxies**.
3. Select the proxy name to view its details.
4. On the details page, the **Target groups** section shows how the proxy is linked to a specific RDS DB instance. You can navigate to the default target group page for a deeper view of this association, including configuration settings defined during proxy creation. These settings include the maximum connection percentage, connection borrow timeout, engine family, and session pinning filters.

CLI

To view your proxy using the CLI, use the describe-db-proxies command. By default, the request returns all proxies owned by your AWS account. To see details for a single proxy, specify its name with the `--db-proxy-name` parameter.

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

To view other information associated with the proxy, use the following commands.

```
aws rds describe-db-proxy-target-groups  --db-proxy-name proxy_name

aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

Use the following sequence of commands to see more detail about the things that are associated with the proxy:

1. To get a list of proxies, run describe-db-proxies.
2. To show connection parameters such as the maximum percentage of connections that the proxy can use, run describe-db-proxy-target-groups `--db-proxy-name`. Use the name of the proxy as the parameter value.
3. To see the details of the RDS DB instance associated with the returned target group, run describe-db-proxy-targets.

RDS API

To view your proxies using the RDS API, use the DescribeDBProxies operation. It returns values of the DBProxy data type.

To see details of the connection settings for the proxy, use the proxy identifiers from this return value with the DescribeDBProxyTargetGroups operation. It returns values of the DBProxyTargetGroup data type.

To see the RDS instance or Aurora DB cluster associated with the proxy, use the DescribeDBProxyTargets operation. It returns values of the DBProxyTarget data type.

Connecting to a database through RDS Proxy

The way to connect to an RDS DB instance through a proxy or by connecting to the database is generally the same. For more information, see Overview of proxy endpoints.

Topics

- Connecting to a database using database credentials
- Connecting to a database using IAM authentication
- Considerations for connecting to Microsoft SQL Server

- Considerations for connecting to PostgreSQL

Connecting to a database using database credentials

Use the following steps to connect to a proxy using database credentials:

1. Find the proxy endpoint. In the AWS Management Console, you can find the endpoint on the details page for the corresponding proxy. With the AWS CLI, you can use the describe-db-proxies command. The following example shows how.

```
# Add --output text to get output as a simple tab-separated list.
$ aws rds describe-db-proxies --query '*[*].
{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
        },
        {
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. Specify the endpoint as the host parameter in the connection string for your client application. For example, specify the proxy endpoint as the value for the `mysql -h` option or `psql -h` option.
3. Supply the same database user name and password as you usually do.

Connecting to a database using IAM authentication

When you use IAM authentication with RDS Proxy, you have two options for authentication between your client and proxy:

- Set up your database users to authenticate with regular user names and passwords. RDS Proxy retrieves the user name and password credentials from Secrets Manager. The connection from RDS Proxy to the underlying database doesn't go through IAM.
- You can also use end-to-end IAM authentication, which connects to your database through the proxy using IAM without requiring database credentials.

To connect to RDS Proxy using IAM authentication, use the same general connection procedure as for IAM authentication with an RDS DB instance. For general information about using IAM, see Security in Amazon RDS . If you are using end-to-end IAM authentication, provide the IAM authentication plugin to your DB user. See Creating a database account using IAM authentication.

The major differences in IAM usage for RDS Proxy include the following:

- With standard IAM authentication, database users have regular credentials within the database. You set up Secrets Manager secrets containing these user names and passwords, and authorize RDS Proxy to retrieve the credentials from Secrets Manager. The IAM authentication applies to the connection between your client program and the proxy. The proxy then authenticates to the database using the user name and password credentials retrieved from Secrets Manager.
- With end-to-end IAM authentication, you don't need to configure Secrets Manager secrets for database credentials. The IAM authentication applies to the connection between the client to the proxy and proxy to the database.
- Instead of the instance, cluster, or reader endpoint, you specify the proxy endpoint. For details about the proxy endpoint, see Connecting to your DB instance using IAM authentication.
- Make sure that you use Transport Layer Security (TLS)/Secure Sockets Layer (SSL) when connecting to a proxy using IAM authentication.

You can grant a specific user access to the proxy by modifying the IAM policy. An example follows.

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

Tip

When configuring IAM authentication for RDS Proxy connections, follow these important guidelines to avoid connection issues:

- Do not grant the `rds_iam` role while maintaining general password authentication for the same database user or role.
- Remember that while clients connect to RDS Proxy using IAM authentication, RDS Proxy always connects to the database using password authentication through Secrets Manager.
- If you experience frequent connection terminations and reconnections, remove any existing `rds_iam` grants from the user or role and use only password authentication.
- Ensure your password policy satisfies SCRAM-SHA-256 safe character requirements.

Mixing IAM and password authentication methods for the same database user can cause connection instability.

Considerations for connecting to Microsoft SQL Server

For connecting to a proxy using IAM authentication, you don't use the password field. Instead, you provide the appropriate token property for each type of database driver in the token field. For example, use the `accessToken` property for JDBC, or the `sql_copt_ss_access_token` property for ODBC. Or use the `AccessToken` property for the .NET SqlClient driver. You can't use IAM authentication with clients that don't support token properties.

Under some conditions, a proxy can't share a database connection and instead pins the connection from your client application to the proxy to a dedicated database connection. For more information about these conditions, see Avoiding pinning an RDS Proxy.

Considerations for connecting to PostgreSQL

If you create a new PostgreSQL database user for connecting to RDS Proxy, make sure that you grant the user `CONNECT` privilege on the database. Without this, the user can't establish a connection. For more information, see the section called “Adding a new database user to a PostgreSQL database when using RDS Proxy”.

When a client starts a connection to a PostgreSQL database, it sends a startup message. This message includes pairs of parameter name and value strings. For details, see the `StartupMessage` in PostgreSQL message formats in the PostgreSQL documentation.

When you connect through an RDS proxy, the startup message can include the following currently recognized parameters:

- `user`
- `database`

The startup message can also include the following additional runtime parameters:

- `application_name`
- `client_encoding`
- `DateStyle`
- `TimeZone`
- `extra_float_digits`
- `search_path`

For more information about PostgreSQL messaging, see the Frontend/Backend protocol in the PostgreSQL documentation.

For PostgreSQL, if you use JDBC, we recommend the following to avoid pinning:

- Set the JDBC connection parameter `assumeMinServerVersion` to at least `9.0` to avoid pinning. This prevents the JDBC driver from performing an extra round trip during connection startup when it runs `SET extra_float_digits = 3`.
- Set the JDBC connection parameter `ApplicationName` to *`any/your-application-name`* to avoid pinning. Doing this prevents the JDBC driver from performing an extra round trip during connection startup when it runs `SET application_name = "PostgreSQL JDBC Driver"`. Note the JDBC parameter is `ApplicationName` but the PostgreSQL `StartupMessage` parameter is `application_name`.

For more information, see Avoiding pinning an RDS Proxy. For more information about connecting using JDBC, see Connecting to the database in the PostgreSQL documentation.

Managing an RDS Proxy

This section provides information on how to manage RDS Proxy operation and configuration. These procedures help your application make the most efficient use of database connections and achieve maximum connection reuse. The more that you can take advantage of connection reuse, the more CPU and memory overhead that you can save. This in turn reduces latency for your application and enables the database to devote more of its resources to processing application requests.

Topics

- Modifying an RDS Proxy
- Adding a new database user when using RDS Proxy
- Moving from standard IAM authentication to end-to-end IAM authentication for RDS Proxy
- RDS Proxy connection considerations
- Avoiding pinning an RDS Proxy
- Deleting an RDS Proxy

Modifying an RDS Proxy

You can change specific settings associated with a proxy after you create the proxy. You do so by modifying the proxy itself, its associated target group, or both. Each proxy has an associated target group.

AWS Management Console

Important

The values in the **Client authentication type** and **IAM authentication** fields apply to all Secrets Manager secrets that are associated with this proxy. To specify different values for each secret, modify your proxy by using the AWS CLI or the API instead.

To modify the settings for a proxy

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Proxies**.

3. In the list of proxies, choose the proxy whose settings you want to modify or go to its details page.
4. For **Actions**, choose **Modify**.
5. Enter or choose the properties to modify. You can modify the following:

   - **Proxy identifier** – Rename the proxy by entering a new identifier.
   - **Idle client connection timeout** – Enter a time period for the idle client connection timeout.
   - **IAM role** – Change the IAM role used to retrieve the secrets from Secrets Manager.

     > **Note**
     >
     > You can't create a new IAM role if you set **Default authentication scheme** to **IAM authentication**.

   - **Secrets Manager secrets** – Add or remove Secrets Manager secrets. These secrets correspond to database user names and passwords.
   - **Client authentication type** – Change the type of authentication for client connections to the proxy.
   - **IAM authentication** – Require or disallow IAM authentication for connections to the proxy.
   - **Default authentication scheme** – Change the default authentication scheme the proxy uses for client connections to the proxy and connections from the proxy to the underlying database.
   - **Require Transport Layer Security** – Turn the requirement for Transport layer Security (TLS) on or off.
   - **VPC security group** – Add or remove VPC security groups for the proxy to use.
   - **Enable enhanced logging** – Enable or disable enhanced logging.
6. Choose **Modify**.

If you didn't find the settings listed that you want to change, use the following procedure to update the target group for the proxy. The *target group* associated with a proxy controls the settings related to the physical database connections. Each proxy has one associated target group named `default`, which is created automatically along with the proxy. You can't rename the default target group.

You can only modify the target group from the proxy details page, not from the list on the **Proxies** page.

To modify the settings for a proxy target group

1. On the **Proxies** page, go to the details page for a proxy.
2. For **Target groups**, choose the `default` link. Currently, all proxies have a single target group named `default`.
3. On the details page for the **default** target group, choose **Modify**.
4. Choose new settings for the properties that you can modify:

   - **Database** – Choose a different RDS DB instance or cluster.
   - **Connection pool maximum connections** – Adjust what percentage of the maximum available connections the proxy can use.
   - **Session pinning filters** – (Optional) Choose a session pinning filter. This circumvents the default safety measures for multiplexing database connections across client connections. Currently, the setting isn't supported for PostgreSQL. The only choice is `EXCLUDE_VARIABLE_SETS`.

     Enabling this setting can cause session variables of one connection to impact other connections. This can cause errors or correctness issues if your queries depend on session variable values set outside of the current transaction. Consider using this option after verifying it is safe for your applications to share database connections across client connections.

     The following patterns can be considered safe:

     - `SET` statements where there is no change to the effective session variable value, i.e., there is no change to the session variable.
     - You change the session variable value and execute a statement in the same transaction.

     For more information, see Avoiding pinning an RDS Proxy.
   - **Connection borrow timeout** – Adjust the connection borrow timeout interval. This setting applies when the maximum number of connections is already being used for the proxy. The setting determines how long the proxy waits for a connection to become available before returning a timeout error.
   - **Initialization query**. (Optional) Add an initialization query, or modify the current one. You can specify one or more SQL statements for the proxy to run when opening each new

database connection. The setting is typically used with SET statements to make sure that each connection has identical settings. Make sure that the query you add is valid. To include multiple variables in a single SET statement, use comma separators. For example:

```
SET variable1=value1, variable2=value2
```

For multiple statements, use semicolons as the separator.

You can't change certain properties, such as the target group identifier and the database engine.

5. Choose **Modify target group**.

AWS CLI

To modify a proxy using the AWS CLI, use the commands modify-db-proxy, modify-db-proxy-target-group, deregister-db-proxy-targets, and register-db-proxy-targets.

With the `modify-db-proxy` command, you can change properties such as the following:

- The set of Secrets Manager secrets used by the proxy.
- Whether TLS is required.
- The idle client timeout.
- Whether to log additional information from SQL statements for debugging.
- The IAM role used to retrieve Secrets Manager secrets.
- The security groups used by the proxy.
- The default authentication scheme associated with the proxy.

The following example shows how to rename an existing proxy.

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

To modify connection-related settings or rename the target group, use the `modify-db-proxy-target-group` command. Currently, all proxies have a single target group named `default`. When you work with this target group, you specify the name of the proxy and `default` for the name of the target group. You can't rename the default target group.

The following example shows how to first check the `MaxIdleConnectionsPercent` setting for a proxy and then change it, using the target group.

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name
 default --connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'

{
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        },
        "TargetGroupName": "default",
        "CreatedDate": "2019-11-30T16:49:27.940Z",
        "DBProxyName": "the-proxy",
        "IsDefault": true
    }
```

```
}
```

With the `deregister-db-proxy-targets` and `register-db-proxy-targets` commands, you change which RDS DB instances the proxy is associated with through its target group. Currently, each proxy can connect to one RDS DB instance. The target group tracks the connection details for all the RDS DB instances in a Multi-AZ configuration.

The following example starts with a proxy that is associated with an Aurora MySQL cluster named `cluster-56-2020-02-25-1399`. The example shows how to change the proxy so that it can connect to a different cluster named `provisioned-cluster`.

When you work with an RDS DB instance, you specify the `--db-instance-identifier` option.

The following example modifies an Aurora MySQL proxy. An Aurora PostgreSQL proxy has port 5432.

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-8898"
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
```

```
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 cluster-56-2020-02-25-1399

aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": []
}

aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

RDS API

To modify a proxy using the RDS API, you use the operations ModifyDBProxy, ModifyDBProxyTargetGroup, DeregisterDBProxyTargets, and RegisterDBProxyTargets operations.

With `ModifyDBProxy`, you can change properties such as the following:

- The set of Secrets Manager secrets used by the proxy.
- Whether TLS is required.
- The idle client timeout.
- Whether to log additional information from SQL statements for debugging.
- The IAM role used to retrieve Secrets Manager secrets.
- The security groups used by the proxy.

With `ModifyDBProxyTargetGroup`, you can modify connection-related settings. Currently, all proxies have a single target group named `default`. When you work with this target group, you specify the name of the proxy and `default` for the name of the target group. You can't rename the default target group.

With `DeregisterDBProxyTargets` and `RegisterDBProxyTargets`, you change which RDS DB instance the proxy is associated with through its target group. Currently, each proxy can connect to one RDS DB instance . The target group tracks the connection details for the RDS DB instances in a Multi-AZ configuration .

Adding a new database user when using RDS Proxy

In some cases, you might add a new database user to an RDS DB instance or cluster that's associated with a proxy. Proceed depending on whether you're using standard authentication with Secrets Manager secrets or end-to-end IAM authentication.

If you are using the standard IAM authentication, follow these instructions:

1. Create a new Secrets Manager secret, using the procedure described in Setting up database credentials for RDS Proxy.
2. Update the IAM role to give RDS Proxy access to the new Secrets Manager secret. To do so, update the resources section of the IAM role policy.
3. Modify the RDS Proxy to add the new Secrets Manager secret under **Secrets Manager secrets**.

4. If the new user takes the place of an existing one, update the credentials stored in the proxy's Secrets Manager secret for the existing user.

If you're using end-to-end IAM authentication, you need to create the database user and configure IAM permissions. To do this, run through the following steps:

1. Create a new database user in your database that matches the IAM user or role name you want to use for authentication.
2. Ensure the database user is configured with IAM authentication plugin at the database. See Creating a database account using IAM authentication.
3. Update the IAM policy to grant the `rds-db:connect` permission to the IAM user or role, as described in Creating an IAM policy for end-to-end IAM authentication.
4. Ensure your proxy is configured to use IAM authentication as the default authentication scheme.

With end-to-end IAM authentication, you don't need to manage database credentials in Secrets Manager secrets, as IAM credentials are used for authentication from the client to the proxy and from the proxy to the database.

Adding a new database user to a PostgreSQL database when using RDS Proxy

When adding a new user to your PostgreSQL database, if you have run the following command:

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
```

Grant the `rdsproxyadmin` user the `CONNECT` privilege so the user can monitor connections on the target database.

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

You can also allow other target database users to perform health checks by changing `rdsproxyadmin` to the database user in the command above.

Changing the password for a database user when using RDS Proxy

In some cases, you might change the password for a database user in an RDS DB instance that's associated with a proxy. If so, update the corresponding Secrets Manager secret with the new password.

If you're using end-to-end IAM authentication, you don't need to update any passwords in Secrets Manager secrets.

Moving from standard IAM authentication to end-to-end IAM authentication for RDS Proxy

If you currently use standard IAM authentication for RDS Proxy, where clients authenticate to the proxy using IAM but the proxy connects to the database using secrets, you can migrate to end-to-end IAM authentication where both client-to-proxy and proxy-to-database connections use IAM authentication.

To move to end-to-end IAM authentication

1. **Update RDS Proxy IAM role permissions**

   Create an updated proxy permission policy that includes both Secrets Manager and `rds:db-connect` permissions:

```
# Create updated proxy permission policy
cat > updated-proxy-policy.json # EOF
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetSecretsValue",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:secretName-1234f"
      ]
    },
    {
      "Sid": "RdsDBConnect",
      "Action": [
        "rds-db:connect"
      ],
      "Effect": "Allow",
      "Resource": [
```

```
        "arn:aws:rds-db:us-east-1:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/
jane_doe"
      ]
    }
  ]
}
```

Update proxy your role policy:

```
aws iam put-role-policy \
            --role-name RDSProxyRole \
            --policy-name UpdatedProxyPermissions \
            --policy-document file://updated-proxy-policy.json
```

2. Modify your RDS Proxy to enable end-to-end IAM authentication

```
aws rds modify-db-proxy \
  --db-proxy-name my-database-proxy \
  --default-auth-scheme IAM_AUTH \
  --region us-east-1
```

Verify that RDS Proxy status is **Available** and `DefaultAuthScheme` is `IAM_AUTH` before proceeding to ensure zero downtime during migration.

```
aws rds describe-db-proxies --db-proxy-name my-database-proxy --region us-east-1
```

Expected output:

```
{
  "DBProxies": [
    {
      "DBProxyName": "my-database-proxy",
      "DBProxyArn": "arn:aws:rds:us-east-1:123456789012:db-
proxy:prx-0123456789abcdef",
      "Status": "available",
      ...
      "DefaultAuthScheme": "IAM_AUTH"
    }
  ]
}
```

3. Enable IAM authentication on database

```
aws rds modify-db-cluster \
  --db-cluster-identifier my-database-cluster \
  --enable-iam-database-authentication \
  --region us-east-1
```

4. Configure database user for IAM authentication

   For RDS for PostgreSQL:

```
GRANT rds_iam TO jane_doe;
```

   For RDS for MySQL and RDS for MariaDB:

```
ALTER USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

5. Your client application code doesn't need to change. The connection process remains the same:

   For RDS for PostgreSQL:

```
# Generate authentication token
export PGPASSWORD=$(aws rds generate-db-auth-token \
  --hostname my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 \
  --port 5432 \
  --username jane_doe \
  --region us-east-1)

# Connect to database through proxy
psql "host=my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 port=5432 user=jane_doe dbname=postgres password=$PGPASSWORD sslmode=require
 sslrootcert=us-east-1-bundle.pem"
```

   For RDS for MySQL and RDS for MariaDB:

```
# Generate authentication token
export MYSQL_PWD=$(aws rds generate-db-auth-token \
  --hostname my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com
 \
```

```
  --port 3306 \
  --username jane_doe \
  --region us-east-1)

# Connect to database through proxy
mysql -h my-database-proxy.proxy-ABCDEFGHIJKL01234.us-east-1.rds.amazonaws.com \
  -P 3306 \
  -u jane_doe \
  --ssl-ca=us-east-1-bundle.pem \
  --enable-cleartext-plugin
```

RDS Proxy connection considerations

Configuring connection settings

To adjust RDS Proxy's connection pooling, you can modify the following settings:

- IdleClientTimeout
- MaxConnectionsPercent
- MaxIdleConnectionsPercent
- ConnectionBorrowTimeout

IdleClientTimeout

You can specify how long a client connection can be idle before the proxy closes it. The default is 1,800 seconds (30 minutes).

A client connection is considered *idle* when the application doesn't submit a new request within the specified time after the previous request completed. The underlying database connection stays open and is returned to the connection pool. Thus, it's available to be reused for new client connections. If you want the proxy to proactively remove stale connections, then lowering the idle client connection timeout. If your workload establishes frequent connections with the proxy, then raise the idle client connection timeout to save the cost of establishing connections.

This setting is represented by the **Idle client connection timeout** field in the RDS console and the `IdleClientTimeout` setting in the AWS CLI and the API. To learn how to change the value of the **Idle client connection timeout** field in the RDS console, see AWS Management Console. To learn how to change the value of the `IdleClientTimeout` setting, see the CLI command modify-db-proxy or the API operation ModifyDBProxy.

MaxConnectionsPercent

You can limit the number of connections that an RDS Proxy can establish with the target database. You specify the limit as a percentage of the maximum connections available for your database. This setting is represented by the **Connection pool maximum connections** field in the RDS console and the `MaxConnectionsPercent` setting in the AWS CLI and the API.

The `MaxConnectionsPercent` value is expressed as a percentage of the `max_connections` setting for the RDS DB instance used by the target group. The proxy doesn't create all of these connections in advance. This setting allows the proxy to establish these connections as the workload needs them.

For example, for a registered database target with `max_connections` set to 1000, and `MaxConnectionsPercent` set to 95, RDS Proxy sets 950 connections as the upper limit for concurrent connections to that database target.

A common side-effect of your workload reaching the maximum number of allowed database connections is an increase in overall query latency, along with an increase in the `DatabaseConnectionsBorrowLatency` metric. You can monitor currently used and total allowed database connections by comparing the `DatabaseConnections` and `MaxDatabaseConnectionsAllowed` metrics.

When setting this parameter, note the following best practices:

- Allow sufficient connection headroom for changes in workload pattern. It is recommended to set the parameter at least 30% above your maximum recent monitored usage. As RDS Proxy redistributes database connection quotas across multiple nodes, internal capacity changes might require at least 30% headroom for additional connections to avoid increased borrow latencies.
- RDS Proxy reserves a certain number of connections for active monitoring to support fast failover, traffic routing and internal operations. The `MaxDatabaseConnectionsAllowed` metric does not include these reserved connections. It represents the number of connections available to serve the workload, and can be lower than the value derived from the `MaxConnectionsPercent` setting.

  Minimal recommended `MaxConnectionsPercent` values

  - db.t3.small: 30
  - db.t3.medium or above: 20

To learn how to change the value of the **Connection pool maximum connections** field in the RDS console, see AWS Management Console. To learn how to change the value of the `MaxConnectionsPercent` setting, see the CLI command modify-db-proxy-target-group or the API operation ModifyDBProxyTargetGroup.

For information on database connection limits, see Maximum number of database connections.

MaxIdleConnectionsPercent

You can control the number of idle database connections that RDS Proxy can keep in the connection pool. By default, RDS Proxy considers a database connection in its pool to be *idle* when there's been no activity on the connection for five minutes.

The `MaxIdleConnectionsPercent` value is expressed as a percentage of the `max_connections` setting for the RDS DB instance target group. The default value is 50 percent of `MaxConnectionsPercent`, and the upper limit is the value of `MaxConnectionsPercent`. For example, if `MaxConnectionsPercent`, is 80, then the default value of `MaxIdleConnectionsPercent` is 40. If the value of `MaxConnectionsPercent` isn't specified, then for RDS for SQL Server, `MaxIdleConnectionsPercent` is 5, and for all other engines, the default is 50.

With a high value, the proxy leaves a high percentage of idle database connections open. With a low value, the proxy closes a high percentage of idle database connections. If your workloads are unpredictable, consider setting a high value for `MaxIdleConnectionsPercent`. Doing so means that RDS Proxy can accommodate surges in activity without opening a lot of new database connections.

This setting is represented by the `MaxIdleConnectionsPercent` setting of `DBProxyTargetGroup` in the AWS CLI and the API. To learn how to change the value of the `MaxIdleConnectionsPercent` setting, see the CLI command modify-db-proxy-target-group or the API operation ModifyDBProxyTargetGroup.

For information on database connection limits, see Maximum number of database connections.

ConnectionBorrowTimeout

You can choose how long RDS Proxy waits for a database connection in the connection pool to become available for use before returning a timeout error. The default is 120 seconds. This setting applies when the number of connections is at the maximum, and so no connections are available in

the connection pool. It also applies when no appropriate database instance is available to handle the request, such as when a failover operation is in process. Using this setting, you can set the best wait period for your application without changing the query timeout in your application code.

This setting is represented by the **Connection borrow timeout** field in the RDS console or the `ConnectionBorrowTimeout` setting of `DBProxyTargetGroup` in the AWS CLI or API. To learn how to change the value of the **Connection borrow timeout** field in the RDS console, see AWS Management Console. To learn how to change the value of the `ConnectionBorrowTimeout` setting, see the CLI command modify-db-proxy-target-group or the API operation ModifyDBProxyTargetGroup.

Client and database connections

Connections from your application to RDS Proxy are known as client connections. Connections from a proxy to the database are database connections. When using RDS Proxy, client connections terminate at the proxy while database connections are managed within RDS Proxy.

Application-side connection pooling can provide the benefit of reducing recurring connection establishment between your application and RDS Proxy.

Consider the following configuration aspects before implementing an application-side connection pool:

- Client connection max life: RDS Proxy enforces a maximum life of client connections of 24 hours. This value is not configurable. Configure your pool with a maximum connection life less than 24 hours to avoid unexpected client connection drops.
- Client connection idle timeout: RDS Proxy enforces a maximum idle time for client connections. Configure your pool with an idle connection timeout of a value lower than your client connection idle timeout setting for RDS Proxy to avoid unexpected connection drops.

The maximum number of client connections configured in your application-side connection pool does not have to be limited to the **max_connections** setting for RDS Proxy.

Client connection pooling results in a longer client connection life. If your connections experience pinning, then pooling client connections might reduce multiplexing efficiency. Client connections that are pinned but idle in the application-side connection pool continue to hold on to a database connection and prevent the database connection to be reused by other client connections. Review your proxy logs to check whether your connections experience pinning.

> **Note**
>
> RDS Proxy closes database connections some time after 24 hours when they are no longer in use. The proxy performs this action regardless of the value of the maximum idle connections setting.

Avoiding pinning an RDS Proxy

Multiplexing is more efficient when database requests don't rely on state information from previous requests. In that case, RDS Proxy can reuse a connection at the conclusion of each transaction. Examples of such state information include most variables and configuration parameters that you can change through SET or SELECT statements. SQL transactions on a client connection can multiplex between underlying database connections by default.

Your connections to the proxy can enter a state known as *pinning*. When a connection is pinned, each later transaction uses the same underlying database connection until the session ends. Other client connections also can't reuse that database connection until the session ends. The session ends when the client connection is dropped.

RDS Proxy automatically pins a client connection to a specific DB connection when it detects a session state change that isn't appropriate for other sessions. Pinning reduces the effectiveness of connection reuse. If all or almost all of your connections experience pinning, consider modifying your application code or workload to reduce the conditions that cause the pinning.

For example, your application changes a session variable or configuration parameter. In this case, later statements can rely on the new variable or parameter to be in effect. Thus, when RDS Proxy processes requests to change session variables or configuration settings, it pins that session to the DB connection. That way, the session state remains in effect for all later transactions in the same session.

For some database engines, this rule doesn't apply to all parameters that you can set. RDS Proxy tracks certain statements and variables. Thus, RDS Proxy doesn't pin the session when you modify them. In this case, RDS Proxy only reuses the connection for other sessions that have the same values for those settings. For details about what RDS Proxy tracks for a database engine, see the following:

- What RDS Proxy tracks for RDS for SQL Server databases
- What RDS Proxy tracks for RDS for MariaDB and RDS for MySQL databases

What RDS Proxy tracks for RDS for SQL Server databases

RDS Proxy tracks the following SQL Server statements:

- USE
- SET ANSI_NULLS
- SET ANSI_PADDING
- SET ANSI_WARNINGS
- SET ARITHABORT
- SET CONCAT_NULL_YIELDS_NULL
- SET CURSOR_CLOSE_ON_COMMIT
- SET DATEFIRST
- SET DATEFORMAT
- SET LANGUAGE
- SET LOCK_TIMEOUT
- SET NUMERIC_ROUNDABORT
- SET QUOTED_IDENTIFIER
- SET TEXTSIZE
- SET TRANSACTION ISOLATION LEVEL

What RDS Proxy tracks for RDS for MariaDB and RDS for MySQL databases

RDS Proxy tracks the following MariaDB and MySQL statements:

- DROP DATABASE
- DROP SCHEMA
- USE

RDS Proxy tracks the following MySQL and MariaDB variables:

- AUTOCOMMIT
- AUTO_INCREMENT_INCREMENT
- CHARACTER SET (or CHAR SET)

- CHARACTER_SET_CLIENT
- CHARACTER_SET_DATABASE
- CHARACTER_SET_FILESYSTEM
- CHARACTER_SET_CONNECTION
- CHARACTER_SET_RESULTS
- CHARACTER_SET_SERVER
- COLLATION_CONNECTION
- COLLATION_DATABASE
- COLLATION_SERVER
- INTERACTIVE_TIMEOUT
- NAMES
- NET_WRITE_TIMEOUT
- QUERY_CACHE_TYPE
- SESSION_TRACK_SCHEMA
- SQL_MODE
- TIME_ZONE
- TRANSACTION_ISOLATION (or TX_ISOLATION)
- TRANSACTION_READ_ONLY (or TX_READ_ONLY)
- WAIT_TIMEOUT

Note

RDS Proxy tracks changes to the TRANSACTION_ISOLATION and TRANSACTION_READ_ONLY variables when you set them at the session scope. However, if you set them at the next transaction scope, RDS Proxy pins connections. This behavior applies whether you use a SET statement or a SET TRANSACTION statement to configure these values.

Minimizing pinning

Performance tuning for RDS Proxy involves trying to maximize transaction-level connection reuse (multiplexing) by minimizing pinning.

You can minimize pinning by doing the following:

- Avoid unnecessary database requests that might cause pinning.
- Set variables and configuration settings consistently across all connections. That way, later sessions are more likely to reuse connections that have those particular settings.

  However, for PostgreSQL setting a variable leads to session pinning.
- For a MySQL engine family database, apply a session pinning filter to the proxy. You can exempt certain kinds of operations from pinning the session if you know that doing so doesn't affect the correct operation of your application.
- See how frequently pinning occurs by monitoring the Amazon CloudWatch metric `DatabaseConnectionsCurrentlySessionPinned`. For information about this and other CloudWatch metrics, see Monitoring RDS Proxy metrics with Amazon CloudWatch.
- If you use SET statements to perform identical initialization for each client connection, you can do so while preserving transaction-level multiplexing. In this case, you move the statements that set up the initial session state into the initialization query used by a proxy. This property is a string containing one or more SQL statements, separated by semicolons.

  For example, you can define an initialization query for a proxy that sets certain configuration parameters. Then, RDS Proxy applies those settings whenever it sets up a new connection for that proxy. You can remove the corresponding SET statements from your application code, so that they don't interfere with transaction-level multiplexing.

  For metrics about how often pinning occurs for a proxy, see Monitoring RDS Proxy metrics with Amazon CloudWatch.

Conditions that cause pinning for all engine families

The proxy pins the session to the current connection in the following situations where multiplexing might cause unexpected behavior:

- Any statement with a text size greater than 16 KB causes the proxy to pin the session.

Conditions that cause pinning for RDS for Microsoft SQL Server

For RDS for SQL Server, the following interactions also cause pinning:

- Using multiple active result sets (MARS). For information about MARS, see the SQL Server documentation.
- Using distributed transaction coordinator (DTC) communication.
- Creating temporary tables, transactions, cursors, or prepared statements.
- Using the following SET statements:
  - SET ANSI_DEFAULTS
  - SET ANSI_NULL_DFLT
  - SET ARITHIGNORE
  - SET DEADLOCK_PRIORITY
  - SET FIPS_FLAGGER
  - SET FMTONLY
  - SET FORCEPLAN
  - SET IDENTITY_INSERT
  - SET NOCOUNT
  - SET NOEXEC
  - SET OFFSETS
  - SET PARSEONLY
  - SET QUERY_GOVERNOR_COST_LIMIT
  - SET REMOTE_PROC_TRANSACTIONS
  - SET ROWCOUNT
  - SET SHOWPLAN_ALL, SHOWPLAN_TEXT, and SHOWPLAN_XML
  - SET STATISTICS
  - SET XACT_ABORT

Conditions that cause pinning for RDS for MariaDB and RDS for MySQL

For MariaDB and MySQL, the following interactions also cause pinning:

- Explicit table lock statements LOCK TABLE, LOCK TABLES, or FLUSH TABLES WITH READ LOCK cause the proxy to pin the session.
- Creating named locks by using GET_LOCK causes the proxy to pin the session.

- Setting a user or system variable (with some exceptions) pins the session to the proxy. If this significantly limits connection reuse, you can configure SET operations to avoid pinning. To do this, adjust the session pinning filters property. For more information, see Creating a proxy for Amazon RDS and Modifying an RDS Proxy.
- Creating a temporary table causes the proxy to pin the session. That way, the contents of the temporary table are preserved throughout the session regardless of transaction boundaries.
- Calling the ROW_COUNT and FOUND_ROWS functions sometimes causes pinning.
- Prepared statements cause the proxy to pin the session. This rule applies whether the prepared statement uses SQL text or the binary protocol.
- RDS Proxy does not pin connections when you use SET LOCAL.
- Calling stored procedures and stored functions doesn't cause pinning. RDS Proxy doesn't detect any session state changes resulting from such calls. Make sure that your application doesn't change session state inside stored routines if you rely on that session state to persist across transactions. For example, RDS Proxy isn't currently compatible with a stored procedure that creates a temporary table that persists across all transactions.
- Queries with executable comments for MySQL (syntax /*! ... */) or MariaDB (syntax /*M! ... */) cause pinning. RDS Proxy cannot parse SQL embedded in these comments to track session state changes.

If you have expert knowledge about your application behavior, you can skip the pinning behavior for certain application statements. To do so, choose the **Session pinning filters** option when creating the proxy. Currently, you can opt out of session pinning for setting session variables and configuration settings.

Conditions that cause pinning for RDS for PostgreSQL

For PostgreSQL, the following interactions also cause pinning:

- Using SET commands.
- Using PREPARE, DISCARD, DEALLOCATE, or EXECUTE commands to manage prepared statements.
- Creating temporary sequences, tables, or views.
- Declaring cursors.
- Discarding the session state.
- Listening on a notification channel.

- Loading a library module such as `auto_explain`.
- Manipulating sequences using functions such as `nextval` and `setval`.
- Interacting with locks using functions such as `pg_advisory_lock` and `pg_try_advisory_lock`.

> **Note**
>
> RDS Proxy does not pin on transaction level advisory locks, specifically `pg_advisory_xact_lock`, `pg_advisory_xact_lock_shared`, `pg_try_advisory_xact_lock`, and `pg_try_advisory_xact_lock_shared`.

- Setting a parameter, or resetting a parameter to its default. Specifically, using `SET` and `set_config` commands to assign default values to session variables.
- Calling stored procedures and stored functions doesn't cause pinning. RDS Proxy doesn't detect any session state changes resulting from such calls. Make sure that your application doesn't change session state inside stored routines if you rely on that session state to persist across transactions. For example, RDS Proxy isn't currently compatible with a stored procedure that creates a temporary table that persists across all transactions.
- Discarding session state. If you use connection pooling libraries with `DISCARD ALL` query configured as a reset query, RDS Proxy pins your client connection on release. This reduces the proxy's multiplexing efficiency and might lead to unexpected results because the `DISCARD ALL` command can interfere with session state management.

Deleting an RDS Proxy

You can delete a proxy when you no longer need it. Or, you might delete a proxy if you take the DB instance or cluster associated with it out of service.

AWS Management Console

To delete a proxy

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Proxies**.
3. Choose the proxy to delete from the list.

4. Choose **Delete Proxy**.

AWS CLI

To delete a DB proxy, use the AWS CLI command delete-db-proxy. To remove related associations, also use the deregister-db-proxy-targets command.

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
    --db-proxy-name proxy_name
    [--target-group-name target_group_name]
    [--target-ids comma_separated_list]       # or
    [--db-instance-identifiers instance_id]       # or
    [--db-cluster-identifiers cluster_id]
```

RDS API

To delete a DB proxy, call the Amazon RDS API function DeleteDBProxy. To delete related items and associations, you also call the functions DeleteDBProxyTargetGroup and DeregisterDBProxyTargets.

Working with Amazon RDS Proxy endpoints

RDS Proxy endpoints provide flexible and efficient ways to manage database connections, which improves scalability, availability, and security. With proxy endpoints, you can:

- **Simplify monitoring and troubleshooting** – Use multiple endpoints to track and manage connections from different applications independently.

Topics

- Overview of proxy endpoints
- Limitations for proxy endpoints
- Proxy endpoints for Multi-AZ DB clusters
- Accessing RDS databases across VPCs
- Creating a proxy endpoint
- Viewing proxy endpoints

- Modifying a proxy endpoint
- Deleting a proxy endpoint

Overview of proxy endpoints

Working with RDS Proxy endpoints involves the same kinds of procedures as with RDS instance endpoints. If you aren't familiar with RDS endpoints, find more information in Connecting to a DB instance running the MySQL database engine and Connecting to a DB instance running the PostgreSQL database engine.

When you create a proxy endpoint, you can associate it with a different virtual private cloud (VPC) than the proxy's VPC. This allows you to connect to the proxy from another VPC, such as one used by a different application within your organization.

For information about limits associated with proxy endpoints, see Limitations for proxy endpoints.

RDS Proxy logs prefix each entry with the name of the associated proxy endpoint. This can be either the name that you specified for a user-defined endpoint, or the special name `default` for the proxy's default read/write endpoint.

Each proxy endpoint has its own set of CloudWatch metrics. Monitor metrics for all proxy endpoints, a specific endpoint, or all read/write or read-only endpoints of a proxy. For more information, see Monitoring RDS Proxy metrics with Amazon CloudWatch.

A proxy endpoint uses the same authentication mechanism as its associated proxy. RDS Proxy automatically sets up permissions and authorizations for the user-defined endpoint, consistent with the properties of the associated proxy.

Limitations for proxy endpoints

RDS Proxy endpoints have the following limitations:

- The RDS proxy default endpoint cannot be modified.
- The maximum number of user-defined endpoints for a proxy is 20. Thus, a proxy can have up to 21 endpoints: the default endpoint, plus 20 that you create.
- When you associate additional endpoints with a proxy, RDS Proxy automatically determines which DB instances in your cluster to use for each endpoint.
- For IPv6 or dual-stack endpoint network types, your VPC and subnets must be configured to support the selected network type.

When you create a proxy, RDS automatically creates a VPC endpoint for secure communication between applications and the database. The VPC endpoint is visible and can be accessed from the Amazon VPC Console.

Adding a new proxy endpoint provisions an AWS PrivateLink interface endpoint. If you add one or more endpoints to your proxy, you incure additional charges. For more information, see RDS Proxy Pricing.

Proxy endpoints for Multi-AZ DB clusters

By default, the endpoint that you connect to when you use RDS Proxy with a Multi-AZ DB cluster has read/write capability. As a result, this endpoint sends all requests to the writer instance of the cluster. All of those connections count against the `max_connections` value for the writer instance. If your proxy is associated with a Multi-AZ DB cluster, then you can create additional read/write or read-only endpoints for that proxy.

You can use a read-only endpoint with your proxy for read-only queries. You do this the same way that you use the reader endpoint for a Multi-AZ DB cluster. Doing so helps you to take advantage of the read scalability of a Multi-AZ DB cluster with one or more reader DB instances. You can run more simultaneous queries and make more simultaneous connections by using a read-only endpoint and adding more reader DB instances to your Multi-AZ DB cluster as needed. These reader endpoints help to improve the read scalability of your query-intensive applications. Reader endpoints also help to improve the availability of your connections if a reader DB instance in your cluster becomes unavailable.

Reader endpoints for Multi-AZ DB clusters

With RDS Proxy, you can create and use reader endpoints. However, these endpoints only work for proxies associated with Multi-AZ DB clusters. If you use the RDS CLI or API, you might see the `TargetRole` attribute with a value of `READ_ONLY`. You can take advantage of such proxies by changing the target of a proxy from an RDS DB instance to a Multi-AZ DB cluster.

You can create and connect to read-only endpoints called *reader endpoints* when you use RDS Proxy with Multi-AZ DB clusters.

How reader endpoints help application availability

In some cases, a reader instance in your cluster might become unavailable. If that occurs, connections that use a reader endpoint of a DB proxy can recover more quickly than ones that use the Multi-AZ DB cluster reader endpoint. RDS Proxy routes connections to only the available

reader instance in the cluster. There isn't a delay due to DNS caching when an instance becomes unavailable.

If the connection is multiplexed, RDS Proxy directs subsequent queries to a different reader instance without any interruption to your application. If a reader instance is in an unavailable state, all client connections to that instance endpoint are closed.

If the connection is pinned, the next query on the connection returns an error. However, your application can immediately reconnect to the same proxy endpoint. RDS Proxy routes the connection to a different reader DB instance that's in `available` state. When you manually reconnect, RDS Proxy doesn't check the replication lag between the old and new reader instance.

If your Multi-AZ DB cluster doesn't have any available reader instances, RDS Proxy attempts to connect to a reader endpoint when it becomes available. If no reader instance becomes available within the connection borrow timeout period, the connection attempt fails. If a reader instance does become available, the connection attempt succeeds.

How reader endpoints help query scalability

Reader endpoints for a proxy help with Multi-AZ DB cluster query scalability in the following ways:

- Where practical, RDS Proxy uses the same reader DB instance for all the queries issue using a particular reader endpoint connection. That way, a set of related queries on the same tables can take advantage of caching, plan optimization, and so on, on a particular DB instance.
- If a reader DB instance becomes unavailable, the effect on your application depends on whether the session is multiplexed or pinned. If the session is multiplexed, RDS Proxy routes any subsequent queries to a different reader DB instance without any action on your part. If the session is pinned, your application gets an error and must reconnect. You can reconnect to the reader endpoint immediately and RDS Proxy routes the connection to an available reader DB instance. For more information about multiplexing and pinning for proxy sessions, see Overview of RDS Proxy concepts.

Accessing RDS databases across VPCs

By default, the components of your RDS technology stack are all in the same Amazon VPC. For example, suppose that an application running on an Amazon EC2 instance connects to an Amazon RDS DB instance. In this case, the application server and database must both be within the same VPC.

With RDS Proxy, you can set up access to an Amazon RDS DB instance in one VPC from resources in another VPC, such as EC2 instances. For example, your organization might have multiple applications that access the same database resources. Each application might be in its own VPC.

To enable cross-VPC access, you create a new endpoint for the proxy. The proxy itself resides in the same VPC as the Amazon RDS DB instance. However, the cross-VPC endpoint resides in the other VPC, along with the other resources such as the EC2 instances. The cross-VPC endpoint is associated with subnets and security groups from the same VPC as the EC2 and other resources. These associations let you connect to the endpoint from the applications that otherwise can't access the database due to the VPC restrictions.

The following steps explain how to create and access a cross-VPC endpoint through RDS Proxy:

1. Create two VPCs, or choose two VPCs that you already use for RDS work. Each VPC should have its own associated network resources such as an internet gateway, route tables, subnets, and security groups. If you only have one VPC, you can consult Getting started with Amazon RDS for the steps to set up another VPC to use RDS successfully. You can also examine your existing VPC in the Amazon EC2 console to see the kinds of resources to connect together.
2. Create a DB proxy associated with the Amazon RDS DB instance that you want to connect to. Follow the procedure in Creating a proxy for Amazon RDS.
3. On the **Details** page for your proxy in the RDS console, under the **Proxy endpoints** section, choose **Create endpoint**. Follow the procedure in Creating a proxy endpoint.
4. Choose whether to make the cross-VPC endpoint read/write or read-only.
5. Instead of accepting the default of the same VPC as the Amazon RDS DB instance, choose a different VPC. This VPC must be in the same AWS Region as the VPC where the proxy resides.
6. Now instead of accepting the defaults for subnets and security groups from the same VPC as the Amazon RDS DB instance, make new selections. Make these based on the subnets and security groups from the VPC that you chose.
7. You don't need to change any of the settings for the Secrets Manager secrets. The same credentials work for all endpoints for your proxy, regardless of which VPC each endpoint is in. Similarly, when using IAM authentication, your IAM configuration and permissions work consistently across all proxy endpoints, even when endpoints are in different VPCs. No additional IAM configuration is required per endpoint.
8. Wait for the new endpoint to reach the **Available** state.

9. Make a note of the full endpoint name. This is the value ending in *`Region_name`*`.rds.amazonaws.com` that you supply as part of the connection string for your database application.
10. Access the new endpoint from a resource in the same VPC as the endpoint. A simple way to test this process is to create a new EC2 instance in this VPC. Then, log into the EC2 instance and run the `mysql` or `psql` commands to connect by using the endpoint value in your connection string.

Creating a proxy endpoint

To create a proxy endpoint, follow these instructions:

Console

To create a proxy endpoint

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Proxies**.
3. Click the name of the proxy that you want to create a new endpoint for.

   The details page for that proxy appears.
4. In the **Proxy endpoints** section, choose **Create proxy endpoint**.

   The **Create proxy endpoint** window appears.
5. For **Proxy endpoint name**, enter a descriptive name of your choice.
6. For **Target role**, choose whether to make the endpoint read/write or read-only.

   Connections that use read/write endpoints can perform any kind of operations, such as data definition language (DDL) statements, data manipulation language (DML) statements, and queries. These endpoints always connect to the primary instance of the RDS DB cluster. You can use read/write endpoints for general database operations when you only use a single endpoint in your application. You can also use read/write endpoints for administrative operations, online transaction processing (OLTP) applications, and extract-transform-load (ETL) jobs.

   Connections that use a read-only endpoint can only perform queries. RDS Proxy can use one of the reader instances for each connection to the endpoint. That way, a query-intensive

application can take advantage of a Multi-AZ DB cluster's clustering capability. These read-only connections don't impose any overhead on the primary instance of the cluster. That way, your reporting and analysis queries don't slow down the write operations of your OLTP applications.

7. For **Virtual Private Cloud (VPC)**, choose the default to access the endpoint from the same EC2 instances or other resources that normally use to access the proxy or its associated database. To set up cross-VPC access for this proxy, choose a VPC other than the default. For more information about cross-VPC access, see Accessing RDS databases across VPCs.
8. For **Endpoint network type**, choose the IP version for the proxy endpoint. The available options are:

   - **IPv4** – The proxy endpoint uses IPv4 addresses only (default).
   - **IPv6** – The proxy endpoint uses IPv6 addresses only.
   - **Dual-stack** – The proxy endpoint supports both IPv4 and IPv6 addresses.

   To use IPv6 or dual-stack, your VPC and subnets must be configured to support the selected network type.
9. For **Subnets**, RDS Proxy fills in the same subnets as the associated proxy by default. To restrict access to the endpoint to only a portion of the VPC's address range being able to connect to it, remove one or more subnets.
10. For **VPC security group**, you can choose an existing security group or create a new one. RDS Proxy fills in the same security group or groups as the associated proxy by default. If the inbound and outbound rules for the proxy are appropriate for this endpoint, then keep the default choice.

    If you choose to create a new security group, specify a name for the security group on this page. Then edit the security group settings from the EC2 console later.
11. Choose **Create proxy endpoint**.

AWS CLI

To create a proxy endpoint, use the AWS CLI create-db-proxy-endpoint command.

Include the following required parameters:

- `--db-proxy-name` *value*
- `--db-proxy-endpoint-name` *value*

- `--vpc-subnet-ids` *`list_of_ids`*. Separate the subnet IDs with spaces. You don't specify the ID of the VPC itself.

You can also include the following optional parameters:

- `--target-role { READ_WRITE | READ_ONLY }`. This parameter defaults to `READ_WRITE`. When the proxy is associated with a Multi-AZ DB cluster that only contains a writer DB instance, you can't specify `READ_ONLY`. For more information about the intended use of read-only endpoints with Multi-AZ DB clusters, see Reader endpoints for Multi-AZ DB clusters.
- `--vpc-security-group-ids` *`value`*. Separate the security group IDs with spaces. If you omit this parameter, RDS Proxy uses the default security group for the VPC. RDS Proxy determines the VPC based on the subnet IDs that you specify for the `--vpc-subnet-ids` parameter.
- `--endpoint-network-type { IPV4 | IPV6 | DUAL }`. This parameter specifies the IP version for the proxy endpoint. The default is `IPV4`. To use `IPV6` or `DUAL`, your VPC and subnets must be configured to support the selected network type.

Example

The following example creates a proxy endpoint named `my-endpoint`.

For Linux, macOS, or Unix:

```
aws rds create-db-proxy-endpoint \
  --db-proxy-name my-proxy \
  --db-proxy-endpoint-name my-endpoint \
  --vpc-subnet-ids subnet_id subnet_id subnet_id ... \
  --target-role READ_ONLY \
  --vpc-security-group-ids security_group_id \
  --endpoint-network-type DUAL
```

For Windows:

```
aws rds create-db-proxy-endpoint ^
  --db-proxy-name my-proxy ^
  --db-proxy-endpoint-name my-endpoint ^
  --vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
  --target-role READ_ONLY ^
  --vpc-security-group-ids security_group_id ^
  --endpoint-network-type DUAL
```

RDS API

To create a proxy endpoint, use the RDS API CreateDBProxyEndpoint action.

Viewing proxy endpoints

To view existing proxy endpoints, follow these instructions:

Console

To view the details for a proxy endpoint

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Proxies**.
3. In the list, choose the proxy whose endpoint you want to view. Click the proxy name to view its details page.
4. In the **Proxy endpoints** section, choose the endpoint that you want to view. Click its name to view the details page.
5. Examine the parameters whose values you're interested in. You can check properties such as the following:
   - Whether the endpoint is read/write or read-only.
   - The endpoint address that you use in a database connection string.
   - The VPC, subnets, and security groups associated with the endpoint.

AWS CLI

To view one or more proxy endpoints, use the AWS CLI describe-db-proxy-endpoints command.

You can include the following optional parameters:

- `--db-proxy-endpoint-name`
- `--db-proxy-name`

The following example describes the `my-endpoint` proxy endpoint.

Example

For Linux, macOS, or Unix:

```
aws rds describe-db-proxy-endpoints \
  --db-proxy-endpoint-name my-endpoint
```

For Windows:

```
aws rds describe-db-proxy-endpoints ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

To describe one or more proxy endpoints, use the RDS API DescribeDBProxyEndpoints operation.

Modifying a proxy endpoint

To modify your proxy endpoints, follow these instructions:

Console

To modify one or more proxy endpoints

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Proxies**.
3. In the list, choose the proxy whose endpoint you want to modify. Click the proxy name to view its details page.
4. In the **Proxy endpoints** section, choose the endpoint that you want to modify. You can select it in the list, or click its name to view the details page.
5. On the proxy details page, under the **Proxy endpoints** section, choose **Edit**. Or, on the proxy endpoint details page, for **Actions**, choose **Edit**.
6. Change the values of the parameters that you want to modify.
7. Choose **Save changes**.

AWS CLI

To modify a proxy endpoint, use the AWS CLI modify-db-proxy-endpoint command with the following required parameters:

- `--db-proxy-endpoint-name`

Specify changes to the endpoint properties by using one or more of the following parameters:

- `--new-db-proxy-endpoint-name`
- `--vpc-security-group-ids`. Separate the security group IDs with spaces.

The following example renames the `my-endpoint` proxy endpoint to `new-endpoint-name`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint \
  --new-db-proxy-endpoint-name new-endpoint-name
```

For Windows:

```
aws rds modify-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint ^
  --new-db-proxy-endpoint-name new-endpoint-name
```

RDS API

To modify a proxy endpoint, use the RDS API ModifyDBProxyEndpoint operation.

Deleting a proxy endpoint

To delete an endpoint for your proxy, follow these instructions:

> **Note**
>
> You can't delete the default proxy endpoint that RDS Proxy automatically creates for each proxy.

When you delete a proxy, RDS Proxy automatically deletes all the associated endpoints.

Console

To delete a proxy endpoint using the AWS Management Console

1. In the navigation pane, choose **Proxies**.
2. In the list, choose the proxy whose endpoint you want to endpoint. Click the proxy name to view its details page.
3. In the **Proxy endpoints** section, choose the endpoint that you want to delete. You can select one or more endpoints in the list, or click the name of a single endpoint to view the details page.
4. On the proxy details page, under the **Proxy endpoints** section, choose **Delete**. Or, on the proxy endpoint details page, for **Actions**, choose **Delete**.

AWS CLI

To delete a proxy endpoint, run the delete-db-proxy-endpoint command with the following required parameters:

- `--db-proxy-endpoint-name`

The following command deletes the proxy endpoint named `my-endpoint`.

For Linux, macOS, or Unix:

```
aws rds delete-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint
```

For Windows:

```
aws rds delete-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

To delete a proxy endpoint with the RDS API, run the DeleteDBProxyEndpoint operation. Specify the name of the proxy endpoint for the `DBProxyEndpointName` parameter.

Monitoring RDS Proxy metrics with Amazon CloudWatch

You can monitor RDS Proxy by using Amazon CloudWatch. CloudWatch collects and processes raw data from the proxies into readable, near-real-time metrics. To find these metrics in the CloudWatch console, choose **Metrics**, then choose **RDS**, and choose **Per-Proxy Metrics**. For more information, see Using Amazon CloudWatch metrics in the Amazon CloudWatch User Guide.

> **Note**
>
> RDS publishes these metrics for each underlying Amazon EC2 instance associated with a proxy. A single proxy might be served by more than one EC2 instance. Use CloudWatch statistics to aggregate the values for a proxy across all the associated instances.
> Some of these metrics might not be visible until after the first successful connection by a proxy.

In the RDS Proxy logs, each entry is prefixed with the name of the associated proxy endpoint. This name can be the name you specified for a user-defined endpoint, or the special name `default` for the default endpoint of a proxy that performs read/write requests.

All RDS Proxy metrics are in the group `proxy`.

Each proxy endpoint has its own CloudWatch metrics. You can monitor the usage of each proxy endpoint independently. For more information about proxy endpoints, see Working with Amazon RDS Proxy endpoints.

You can aggregate the values for each metric using one of the following dimension sets. For example, by using the `ProxyName` dimension set, you can analyze all the traffic for a particular proxy. By using the other dimension sets, you can split the metrics in different ways. You can split the metrics based on the different endpoints or target databases of each proxy, or the read/write and read-only traffic to each database.

- Dimension set 1: `ProxyName`
- Dimension set 2: `ProxyName`, `EndpointName`
- Dimension set 3: `ProxyName`, `TargetGroup`, `Target`
- Dimension set 4: `ProxyName`, `TargetGroup`, `TargetRole`

Metric	Description	Valid period	CloudWatch dimension set
AvailabilityPercentage	The percentage of time for which the target group was available in the role indicated by the dimension. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 4
ClientConnections	The current number of client connections. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 2
ClientConnectionsClosed	The number of client connections closed. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 2
ClientConnectionsInSetup	The current number of client connections open but have not completed setup. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 2

Metric	Description	Valid period	CloudWatch dimension set
ClientConnectionsNoTLS	The current number of client connections without Transport Layer Security (TLS). This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 2
ClientConnectionsReceived	The number of client connection requests received. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 2
ClientConnectionsSetupFailedAuth	The number of client connection attempts that failed due to misconfigured authentication or TLS. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 2
ClientConnectionsSetupSucceeded	The number of client connections successfully established with any authentication mechanism with or without TLS. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 2

Metric	Description	Valid period	CloudWatch dimension set
ClientConnectionsTLS	The current number of client connections with TLS. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 2
DatabaseConnectionRequests	The number of requests to create a database connection. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionRequestsWithTLS	The number of requests to create a database connection with TLS. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnections	The current number of database connections. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 3, Dimension set 4

Metric	Description	Valid period	CloudWatch dimension set
DatabaseConnectionsBorrowLatency	The time in microseconds that it takes for the proxy being monitored to get a database connection. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 2
DatabaseConnectionsCurrentlyBorrowed	The current number of database connections in the borrow state. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsCurrentlyInTransaction	The current number of database connections in a transaction. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 3, Dimension set 4

Metric	Description	Valid period	CloudWatch dimension set
DatabaseConnectionsCurrentlySessionPinned	The current number of database connections currently pinned because of operations in client requests that change session state. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsSetupFailed	The number of database connection requests that failed. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsSetupSucceeded	The number of database connections successfully established with or without TLS. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 3, Dimension set 4

Metric	Description	Valid period	CloudWatch dimension set
DatabaseConnectionsWithTLS	The current number of database connections with TLS. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
MaxDatabaseConnectionsAllowed	The maximum number of database connections allowed. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1, Dimension set 3, Dimension set 4
QueryDatabaseResponseLatency	The time in microseconds that the database took to respond to the query. The most useful statistic for this metric is Average.	1 minute and above	Dimension set 1, Dimension set 2, Dimension set 3, Dimension set 4
QueryRequests	The number of queries received. A query including multiple statements is counted as one query. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 2

Metric	Description	Valid period	CloudWatch dimension set
QueryRequestsNoTLS	The number of queries received from non-TLS connections. A query including multiple statements is counted as one query. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 2
QueryRequestsTLS	The number of queries received from TLS connections. A query including multiple statements is counted as one query. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1, Dimension set 2
QueryResponseLatency	The time in microseconds between getting a query request and the proxy responding to it. The most useful statistic for this metric is Average.	1 minute and above	Dimension set 1, Dimension set 2

You can find logs of RDS Proxy activity under CloudWatch in the AWS Management Console. Each proxy has an entry in the **Log groups** page.

Important

These logs are intended for human consumption for troubleshooting purposes and not for programmatic access. The format and content of the logs is subject to change.
In particular, older logs don't contain any prefixes indicating the endpoint for each request. In newer logs, each entry is prefixed with the name of the associated proxy endpoint. This name can be the name that you specified for a user-defined endpoint, or the special name `default` for requests using the default endpoint of a proxy.

Working with RDS Proxy events

An *event* indicates a change in an environment such as an AWS environment or a service or application from a software as a service (SaaS) partner. Or, it can be one of your own custom applications or services. For example, Amazon RDS generates an event when you create or modify an RDS Proxy. Amazon RDS delivers events to Amazon EventBridge in near-real time. Following, you can find a list of RDS Proxy events that you can subscribe to and an example of an RDS Proxy event.

For more information about working with events, see the following:

- For instructions on how to view events by using the AWS Management Console, AWS CLI, or RDS API, see Viewing Amazon RDS events.
- To learn how to configure Amazon RDS to send events to EventBridge, see Creating a rule that triggers on an Amazon RDS event.

RDS Proxy events

The following table shows the event category and a list of events when an RDS Proxy is the source type.

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0204	RDS modified DB proxy *name*.	None

Category	RDS event ID	Message	Notes
configuration change	RDS-EVENT-0207	RDS modified the end point of the DB proxy *name*.	None
configuration change	RDS-EVENT-0213	RDS detected the addition of the DB instance and automatically added it to the target group of the DB proxy *name*.	None
configuration change	RDS-EVENT-0214	RDS detected deletion of DB instance *name* and automatically removed it from target group *name* of DB proxy *name*.	None
configuration change	RDS-EVENT-0215	RDS detected deletion of DB cluster *name* and automatically removed it from target group *name* of DB proxy *name*.	None
creation	RDS-EVENT-0203	RDS created DB proxy *name*.	None
creation	RDS-EVENT-0206	RDS created endpoint *name* for DB proxy *name*.	None
deletion	RDS-EVENT-0205	RDS deleted DB proxy *name*.	None
deletion	RDS-EVENT-0208	RDS deleted endpoint *name* for DB proxy *name*.	None

Category	RDS event ID	Message	Notes
failure	RDS-EVENT-0243	RDS failed to provision capacity for proxy *name* because there aren't enough IP addresses available in your subnets: *name*. To fix the issue, make sure that your subnets have the minimum number of unused IP addresses as recommend ed in the RDS Proxy documentation.	To determine the recommended number for your instance class, see Planning for IP address capacity.
failure	RDS-EVENT-0275	RDS throttled some connections to DB proxy *name*. The number of simultaneous connection requests from the client to the proxy has exceeded the limit.	None

The following is an example of an RDS Proxy event in JSON format. The event shows that RDS modified the endpoint named `my-endpoint` of the RDS Proxy named `my-rds-proxy`. The event ID is RDS-EVENT-0207.

```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Proxy Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
     "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
  ],
```

```
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PROXY",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
    "SourceIdentifier": "my-endpoint",
    "EventID": "RDS-EVENT-0207"
  }
}
```

Troubleshooting for RDS Proxy

Following, you can find troubleshooting ideas for some common RDS Proxy issues and information on CloudWatch logs for RDS Proxy.

In the RDS Proxy logs, each entry is prefixed with the name of the associated proxy endpoint. This name can be the name that you specified for a user-defined endpoint. Or, it can be the special name `default` for the default endpoint of a proxy that performs read/write requests. For more information about proxy endpoints, see Working with Amazon RDS Proxy endpoints.

Topics

- Verifying connectivity for a proxy
- Common issues and solutions
- Troubleshooting RDS Proxy issues with RDS for MySQL
- Troubleshooting RDS Proxy issues with RDS for PostgreSQL

Verifying connectivity for a proxy

You can use the following commands to verify that all components such as the proxy, database, and compute instances in the connection can communicate with the each other.

Examine the proxy itself using the describe-db-proxies command. Also examine the associated target group using the describe-db-proxy-target-groups command. Check that the details of the targets match the RDS DB instance that you intend to associate with the proxy. Use commands such as the following.

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

To confirm that the proxy can connect to the underlying database, examine the targets specified in the target groups using the describe-db-proxy-targets command. Use a command such as the following.

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

The output of the describe-db-proxy-targets command includes a `TargetHealth` field. You can examine the fields `State`, `Reason`, and `Description` inside `TargetHealth` to check if the proxy can communicate with the underlying DB instance.

- A `State` value of `AVAILABLE` indicates that the proxy can connect to the DB instance.
- A `State` value of `UNAVAILABLE` indicates a temporary or permanent connection problem. In this case, examine the `Reason` and `Description` fields. For example, if `Reason` has a value of `PENDING_PROXY_CAPACITY`, try connecting again after the proxy finishes its scaling operation. If `Reason` has a value of `UNREACHABLE`, `CONNECTION_FAILED`, or `AUTH_FAILURE`, use the explanation from the `Description` field to help you diagnose the issue.
- The `State` field might have a value of `REGISTERING` for a brief time before changing to `AVAILABLE` or `UNAVAILABLE`.

If the following Netcat command (`nc`) is successful, you can access the proxy endpoint from the EC2 instance or other system where you're logged in. This command reports failure if you're not in the same VPC as the proxy and the associated database. You might be able to log directly in to the database without being in the same VPC. However, you can't log into the proxy unless you're in the same VPC.

```
nc -zx MySQL_proxy_endpoint 3306

nc -zx PostgreSQL_proxy_endpoint 5432
```

You can use the following commands to make sure that your EC2 instance has the required properties. In particular, the VPC for the EC2 instance must be the same as the VPC for the that the proxy connects to.

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

Examine the Secrets Manager secrets used for the proxy.

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

Make sure that the `SecretString` field displayed by `get-secret-value` is encoded as a JSON string that includes the `username` and `password` fields. The following example shows the format of the `SecretString` field.

```
{
  "ARN": "some_arn",
  "Name": "some_name",
  "VersionId": "some_version_id",
  "SecretString": '{"username":"some_username","password":"some_password"}',
  "VersionStages": [ "some_stage" ],
  "CreatedDate": some_timestamp
}
```

When troubleshooting IAM authentication issues, verify the following:

- The database has IAM authentication enabled.
- The proxy is configured with the correct authentication scheme.
- The IAM policies in the IAM role provided to the proxy grant the necessary `rds-db:connect` permission to the appropriate database and its username.
- For end-to-end IAM authentication, database users exist that match the IAM user or role names.
- SSL/TLS is enabled for the connection.

Common issues and solutions

This section describes some common issues and potential solutions when using RDS Proxy.

After running the `aws rds describe-db-proxy-targets` CLI command, if the `TargetHealth` description states `Proxy does not have any registered credentials`, verify the following:

- There are credentials registered for the user to access the proxy.
- The IAM role to access the Secrets Manager secret used by the proxy is valid.

You might encounter the following RDS events while creating or connecting to a DB proxy.

Category	RDS event ID	Description
failure	RDS-EVENT-0243	RDS couldn't provision capacity for the proxy because there aren't enough IP addresses available in your subnets. To fix the issue, make sure that your subnets have the minimum number of unused IP addresses. To determine the recommend ed number for your instance class, see Planning for IP address capacity.
failure	RDS-EVENT-0275	RDS throttled some connectio ns to DB proxy *name*. The number of simultaneous connection requests from the client to the proxy has exceeded the limit.

You might encounter the following issues while creating a new proxy or connecting to a proxy.

Error	Causes or workarounds
`403: The security token included in the request is invalid`	Select an existing IAM role instead of choosing to create a new one.

Troubleshooting RDS Proxy issues with RDS for MySQL

You might encounter the following issues while connecting to a MySQL proxy.

Error	Causes or workarounds
`ERROR 1040 (HY000): Connections rate limit exceeded (limit_value )`	The rate of connection requests from the client to the proxy has exceeded the limit.
`ERROR 1040 (HY000): IAM authentication rate limit exceeded`	The number of simultaneous requests with IAM authentication from the client to the proxy has exceeded the limit.
`ERROR 1040 (HY000): Number simultane ous connectio ns exceeded (limit_value )`	The number of simultaneous connection requests from the client to the proxy exceeded the limit.
`ERROR 1045 (28000): Access denied for user 'DB_USER'@'%' (usi password: YES)`	The Secrets Manager secret used by the proxy doesn't match the user name and password of an existing database user. Either update the credentials in the Secrets Manager secret, or make sure the database user exists and has the same password as in the secret.
`ERROR 1105 (HY000): Unknown error`	An unknown error occurred.

Error	Causes or workarounds
`ERROR 1231 (42000): Variable ''character_set_client'' can't be set to the value of value`	The value set for the `character_set_client` parameter is not valid. For example, the value `ucs2` is not valid because it can crash the MySQL server.
`ERROR 3159 (HY000): This RDS Proxy requires TLS connections.`	You enabled the setting **Require Transport Layer Security** in the proxy but your connection included the parameter `ssl-mode=DISABLED` in the MySQL client. Do either of the following: • Disable the setting **Require Transport Layer Security** for the proxy. • Connect to the database using the minimum setting of `ssl-mode=REQUIRED` in the MySQL client.
`ERROR 2026 (HY000): SSL connection error: Internal Server Error`	The TLS handshake to the proxy failed. Some possible reasons include the following: • SSL is required but the server doesn't support it. • An internal server error occurred. • A bad handshake occurred.
`ERROR 9501 (HY000): Timed-out waiting to acquire database connection`	The proxy timed-out waiting to acquire a database connection. Some possible reasons include the following: • The proxy is unable to establish a database connection because the maximum connections have been reached • The proxy is unable to establish a database connection because the database is unavailable.

Troubleshooting RDS Proxy issues with RDS for PostgreSQL

You might encounter the following issues while connecting to a PostgreSQL proxy.

Error	Cause	Solution
`ERROR 28000: IAM authentication is allowed only with SSL connections.`	The user tried to connect to the database using IAM authentication with the setting `sslmode=disable` in the PostgreSQL client.	The user needs to connect to the database using the minimum setting of `sslmode=require` in the PostgreSQL client. For more information, see the PostgreSQL SSL support documentation.
`ERROR 28000: This RDS proxy has no credentials for the role` *`role_name`* `. Check the credentials for this role and try again.`	There is no Secrets Manager secret for this role.	Add a Secrets Manager secret for this role. For more information, see Configuring IAM authentication for RDS Proxy.
`ERROR 28000: RDS supports only IAM, MD5, or SCRAM authentication.`	The database client being used to connect to the proxy is using an authentication mechanism not currently supported by the proxy.	If you're not using IAM authentication, use the MD5 or SCRAM password authentication.
`ERROR 28000: A user name is missing from the connection startup packet. Provide a user name for this connection.`	The database client being used to connect to the proxy isn't sending a user name when trying to establish a connection.	Make sure to define a user name when setting up a connection to the proxy using the PostgreSQL client of your choice.
`ERROR 28000: IAM is allowed only with SSL connections.`	A client tried to connect using IAM authentication, but SSL wasn't enabled.	Enable SSL in the PostgreSQL client.

Error	Cause	Solution
`ERROR 28000: This RDS Proxy requires TLS connections.`	The user enabled the option **Require Transport Layer Security** but tried to connect with `sslmode=disable` in the PostgreSQL client.	To fix this error, do one of the following: • Disable the proxy's **Require Transport Layer Security** option. • Connect to the database using the minimum setting of `sslmode=allow` in the PostgreSQL client.
`ERROR 28P01: IAM authentication failed for user` *`user_name`* `. Check the IAM token for this user and try again.`	This error might be due to the following reasons: • The client supplied the incorrect IAM user name. • The client supplied an incorrect IAM authorization token for the user. • The client is using an IAM policy that does not have the necessary permissions. • The client supplied an expired IAM authorization token for the user.	To fix this error, do the following: 1. Confirm that the provided IAM user exists. 2. Confirm that the IAM authorization token belongs to the provided IAM user. 3. Confirm that the IAM policy has adequate permissions for RDS. 4. Check the validity of the IAM authorization token used.
`ERROR 28P01: The password that was provided for the role` *`role_name`* `is wrong.`	The password for this role doesn't match the Secrets Manager secret.	Check the secret for this role in Secrets Manager to see if the password is the same as what's being used in your PostgreSQL client.

Error	Cause	Solution
`ERROR 28P01: The IAM authentication failed for the role` *`role_name`* `. Check the IAM token for this role and try again.`	There is a problem with the IAM token used for IAM authentication.	Generate a new authentication token and use it in a new connection.
`ERROR 0A000: Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.`	The PostgreSQL client used to connect to the proxy uses a protocol older than 3.0.	Use a newer PostgreSQL client that supports the 3.0 messaging protocol. If you're using the PostgreSQL `psql` CLI, use a version greater than or equal to 7.4.
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support streaming replication mode.`	The PostgreSQL client used to connect to the proxy is trying to use the streaming replication mode, which isn't currently supported by RDS Proxy.	Turn off the streaming replication mode in the PostgreSQL client being used to connect.
`ERROR 0A000: Feature not supported: RDS Proxy currently doesn't support the option` *`option_name`* `.`	Through the startup message, the PostgreSQL client used to connect to the proxy is requesting an option that isn't currently supported by RDS Proxy.	Turn off the option being shown as not supported from the message above in the PostgreSQL client being used to connect.
`ERROR 53300: The IAM authentication failed because of too many competing requests.`	The number of simultaneous requests with IAM authentication from the client to the proxy has exceeded the limit.	Reduce the rate in which connections using IAM authentication from a PostgreSQL client are established.

Error	Cause	Solution
`ERROR 53300: The maximum number of client connections to the proxy exceeded number_value .`	The number of simultaneous connection requests from the client to the proxy exceeded the limit.	Reduce the number of active connections from PostgreSQL clients to this RDS proxy.
`ERROR 53300: Rate of connection to proxy exceeded number_value .`	The rate of connection requests from the client to the proxy has exceeded the limit.	Reduce the rate in which connections from a PostgreSQL client are established.
`ERROR XX000: Unknown error.`	An unknown error occurred.	Reach out to AWS Support to investigate the issue.

Error	Cause	Solution
`ERROR 08000: Timed-out waiting to acquire database connection.`	The proxy timed-out waiting to acquire a database connection within the duration specified by the `ConnectionBorrowTimeout` setting. Some possible reasons include the following: • The proxy can't establish a database connection because the maximum connections have been reached. • The proxy can't establish a database connection because the database is unavailable or if connection establishment to the database takes longer than the configured `ConnectionBorrowTimeout` .	Possible solutions are the following: • Avoid pinning proxy connections. See Avoiding pinning an RDS Proxy. • Review `ConnectionBorrowTimeout` and `MaxConnectionsPercent` settings. See RDS Proxy connection considerations. • Review target availability. See `AvailabilityPercentage` in Monitoring RDS Proxy metrics with Amazon CloudWatch.

Error	Cause	Solution
`ERROR XX000: Request returned an error: database_error .`	The database connection established from the proxy returned an error.	The solution depends on the specific database error. One example is: `Request returned an error: database "your-database-name" does not exist`. This means that the specified database name doesn't exist on the database server. Or it means that the user name used as a database name (if a database name isn't specified) doesn't exist on the server.
`ERROR 53300: The IAM authentication failed because of too many competing requests.`	The number of simultaneous requests with IAM authentication from the client to the proxy has exceeded the limit.	Reduce the rate in which connections using IAM authentication from a PostgreSQL client are established.
`ERROR 28000: Enable IAM authentication for the client connection to the proxy and try again.`	RDS Proxy can't connect to the database because IAM authentication isn't enabled for the client connection to the proxy. This occurs when the proxy's `DefaultAuthScheme` parameter is set to `IAM_AUTH` with a registered user, but the client is using password authentication instead of IAM authentication.	Enable IAM authentication for the client connection to the proxy and try again.

Error	Cause	Solution
`ERROR 28000: Configure IAM authentication as the DefaultAuthScheme in your proxy and try again.`	RDS Proxy can't connect to the database because the `DefaultAuthScheme` isn't set to `IAM_AUTH`. The proxy's `DefaultAuthScheme` parameter is set to `NONE`, but the client is attempting to use IAM authentication.	Set `DefaultAuthScheme` to `IAM_AUTH` for your proxy and try again.

Troubleshooting deleted `postgres` database

If you mistakenly delete the `postgres` database from your instance, you need to restore the database to restore connectivity to your instance. Run the following commands within your DB instance:

```
CREATE DATABASE postgres;
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

Using RDS Proxy with AWS CloudFormation

You can use RDS Proxy with AWS CloudFormation. This helps you to create groups of related resources. Such a group can include a proxy that can connect to a newly created Amazon RDS DB instance. RDS Proxy support in CloudFormation involves two new registry types: `DBProxy` and `DBProxyTargetGroup`.

The following listing shows a sample CloudFormation template for RDS Proxy.

```
Resources:
 DBProxy:
   Type: AWS::RDS::DBProxy
   Properties:
     DBProxyName: CanaryProxy
     EngineFamily: MYSQL
     RoleArn:
      Fn::ImportValue: SecretReaderRoleArn
     Auth:
```

```
      - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
    VpcSubnetIds:
      Fn::Split: [",", "Fn::ImportValue": SubnetIds]

ProxyTargetGroup:
  Type: AWS::RDS::DBProxyTargetGroup
  Properties:
    DBProxyName: CanaryProxy
    TargetGroupName: default
    DBInstanceIdentifiers:
      - Fn::ImportValue: DBInstanceName
  DependsOn: DBProxy
```

For more information about the resources in this sample, see DBProxy and DBProxyTargetGroup.

For more information about resources that you can create using CloudFormation, see RDS resource type reference.

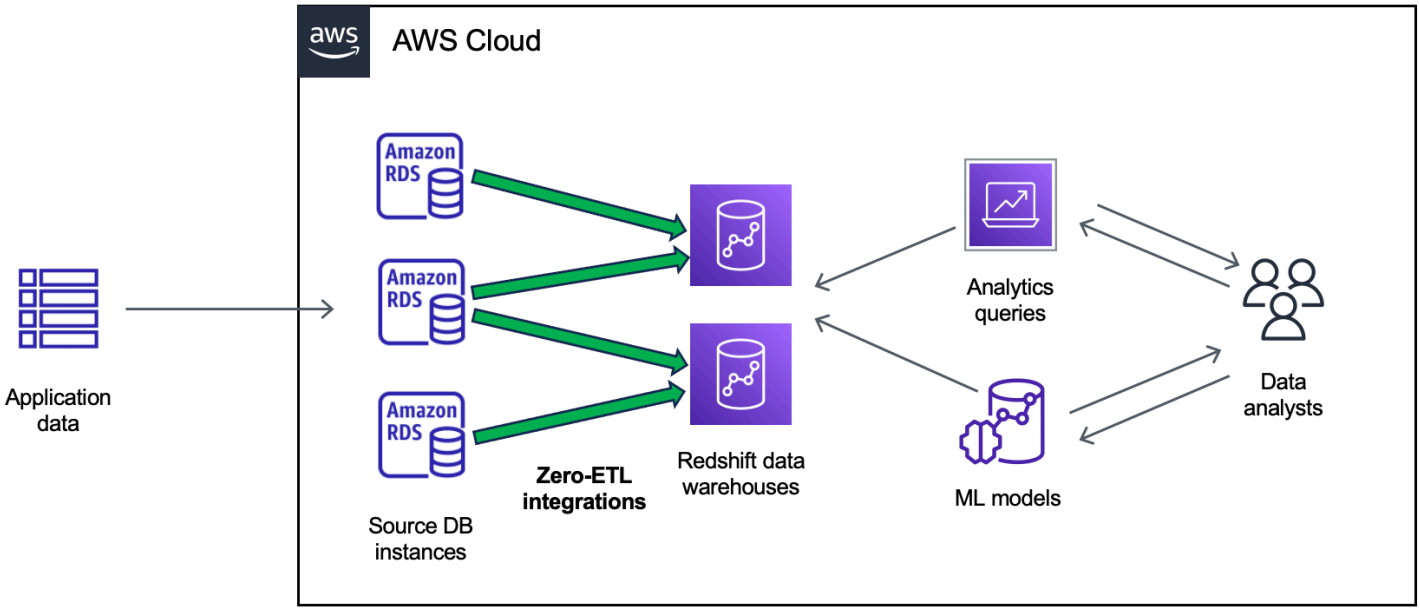
Amazon RDS zero-ETL integrations

An Amazon RDS zero-ETL integration with Amazon Redshift and Amazon SageMaker AI enables near real-time analytics and machine learning (ML) using data from RDS. It's a fully managed solution for making transactional data available in your analytics destination after it is written to an RDS database. *Extract, transform,* and *load* (ETL) is the process of combining data from multiple sources into a large, central data warehouse.

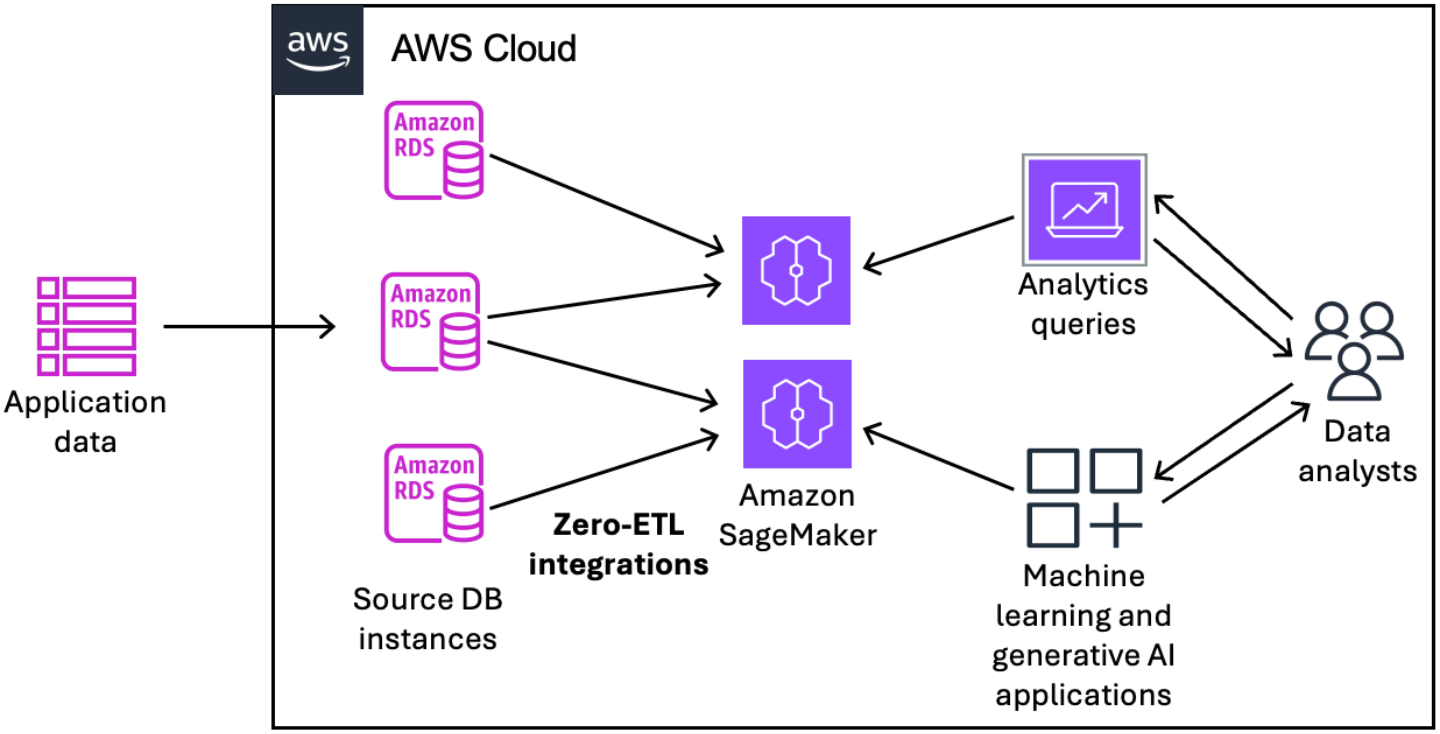
A zero-ETL integration makes the data in your RDS database available in Amazon Redshift or an Amazon SageMaker AI lakehouse in near real-time. Once that data is in the target data warehouse or data lake, you can power your analytics, ML, and AI workloads using the built-in capabilities, such as machine learning, materialized views, data sharing, federated access to multiple data stores and data lakes, and integrations with Amazon SageMaker AI, Quick, and other AWS services.

To create a zero-ETL integration, you specify an RDS database as the *source*, and a supported data warehouse or lakehouse as the *target*. The integration replicates data from the source database into the target data warehouse or lakehouse.

The following diagram illustrates this functionality for zero-ETL integration with Amazon Redshift:



The following diagram illustrates this functionality for zero-ETL integration with an Amazon SageMaker AI lakehouse:



The integration monitors the health of the data pipeline and recovers from issues when possible. You can create integrations from multiple RDS databases into a single target data warehouse or lakehouse enabling you to derive insights across multiple applications.

Topics

- Benefits
- Key concepts
- Limitations
- Quotas
- Supported Regions
- Getting started with Amazon RDS zero-ETL integrations
- Creating Amazon RDS zero-ETL integrations with Amazon Redshift
- Creating Amazon RDS zero-ETL integrations with an Amazon SageMaker lakehouse
- Data filtering for Amazon RDS zero-ETL integrations
- Adding data to a source RDS database and querying it
- Viewing and monitoring Amazon RDS zero-ETL integrations
- Modifying Amazon RDS zero-ETL integrations

- Deleting Amazon RDS zero-ETL integrations
- Troubleshooting Amazon RDS zero-ETL integrations

Benefits

RDS zero-ETL integrations have the following benefits:

- Help you derive holistic insights from multiple data sources.
- Eliminate the need to build and maintain complex data pipelines that perform extract, transform, and load (ETL) operations. Zero-ETL integrations remove the challenges that come with building and managing pipelines by provisioning and managing them for you.
- Reduce operational burden and cost, and let you focus on improving your applications.
- Let you leverage the target destination's analytics and ML capabilities to derive insights from transactional and other data, to respond effectively to critical, time-sensitive events.

Key concepts

As you get started with zero-ETL integrations, consider the following concepts:

Integration

A fully managed data pipeline that automatically replicates transactional data and schemas from an RDS database to a data warehouse or catalog.

Source database

The RDS database where data is replicated from. You can specify a Single-AZ or Multi-AZ DB instance, or a Multi-AZ DB cluster (for RDS for MySQL only).

Target

The data warehouse or lakehouse where the data is replicated to. There are two types of data warehouse: a provisioned cluster data warehouse and a serverless data warehouse. A provisioned cluster data warehouse is a collection of computing resources called nodes, which are organized into a group called a *cluster*. A serverless data warehouse is comprised of a workgroup that stores compute resources, and a namespace that houses the database objects and users. Both data warehouses run an analytics engine and contain one or more databases.

A target lakehouse consists of catalogs, databases, tables, and views. For more information about lakehouse architecture, see SageMaker Lakehouse components in the *Amazon SageMaker AI Unified Studio User Guide*.

Multiple source databases can write to the same target.

For more information, see Data warehouse system architecture in the *Amazon Redshift Developer Guide*.

Limitations

The following limitations apply to RDS zero-ETL integrations.

Topics

- General limitations
- RDS for MySQL limitations
- RDS for PostgreSQL limitations
- RDS for Oracle limitations
- Amazon Redshift limitations
- Amazon SageMaker AI lakehouse limitations

General limitations

- The source database must be in the same Region as the target.
- You can't rename a database if it has existing integrations.
- You can't create multiple integrations between the same source and target databases.
- You can't delete a database that has existing integrations. You must delete all associated integrations first.
- If you stop the source database, the last few transactions might not be replicated to the target until you resume the database.
- You can’t delete an integration if the source database is stopped.
- If your database is the source of a blue/green deployment, the blue and green environments can't have existing zero-ETL integrations during switchover. You must delete the integration first and switch over, then recreate it.

- You can't create an integration for a source database that has another integration being actively created.
- When you initially create an integration, or when a table is being resynchronized, data seeding from the source to the target can take 20-25 minutes or more depending on the size of the source database. This delay can lead to increased replica lag.
- Some data types aren't supported. For more information, see the section called “Data type differences”.
- System tables, temporary tables, and views aren't replicated to target warehouses.
- Performing DDL commands (for example `ALTER TABLE`) on a source table can trigger a table resynchronization, making the table unavailable for querying while it's resynchronizing. For more information, see the section called “One or more of my Amazon Redshift tables requires a resync”.

RDS for MySQL limitations

- Your source database must be running a supported version of RDS for MySQL. For a list of supported versions, see the section called “Zero-ETL integrations ”.
- Zero-ETL integrations are not supported on both the primary instance and a read replica instance in the same AWS Region.
- Zero-ETL integrations rely on MySQL binary logging (binlog) to capture ongoing data changes. Don't use binlog-based data filtering, as it can cause data inconsistencies between the source and target databases.
- Zero-ETL integrations are supported only for databases configured to use the InnoDB storage engine.
- Foreign key references with predefined table updates aren't supported. Specifically, `ON DELETE` and `ON UPDATE` rules aren't supported with `CASCADE`, `SET NULL`, and `SET DEFAULT` actions. Attempting to create or update a table with such references to another table will put the table into a failed state.
- You can't create an integration for a source database that uses magnetic storage.

RDS for PostgreSQL limitations

- The source database must be an RDS for PostgreSQL instance running version 15.7+, 16.3+, or 17.1+. Earlier versions are not supported.

- RDS for PostgreSQL zero-ETL integrations don't support Multi-AZ DB clusters as source databases.
- You can't create a zero-ETL integration from an RDS for PostgreSQL read replica instance.
- PostgreSQL unlogged tables and materialized views are not replicated to Amazon Redshift.
- Replication of certain PostgreSQL data types, such as geometry data types and data greater than 64KB, is not supported due to limitations in Amazon Redshift. For more information about data type differences between RDS for PostgreSQL and Amazon Redshift, see the section called "RDS for PostgreSQL" in the **Data type differences** section.
- You can't perform a major version upgrade on the source RDS for PostgreSQL instance while it has an active zero-ETL integration. To upgrade the source instance, you must first delete all existing zero-ETL integrations. After the major version upgrade is complete, you can recreate the zero-ETL integrations.
- If you perform declarative partitioning transactions on the source DB instance, all affected tables enter a failed state and are no longer accessible.

RDS for Oracle limitations

- The source database must be an RDS for Oracle instance running version 19c Enterprise Edition or Standard Edition 2, July 2019 Release Update or later. Earlier versions are not supported.
- You can't create a zero-ETL integration from an RDS for Oracle read replica instance.
- You can't rename a tenant database when there is a zero-ETL integration on that tenant database.
- A tenant database can have only one zero-ETL integration.
- RDS for Oracle and Amazon Redshift have some data type differences. For more information, see the section called "RDS for Oracle" in the **Data type differences** section.

Amazon Redshift limitations

For a list of Amazon Redshift limitations related to zero-ETL integrations, see Considerations when using zero-ETL integrations with Amazon Redshift in the *Amazon Redshift Management Guide*.

Amazon SageMaker AI lakehouse limitations

Following is a limitation for Amazon SageMaker AI lakehouse zero-ETL integrations.

- Catalog names are limited to 19 characters in length.

Quotas

Your account has the following quotas related to RDS zero-ETL integrations. Each quota is per-Region unless otherwise specified.

Name	Default	Description
Integrations	100	The total number of integrations within an AWS account.
Integrations per target	50	The number of integrations sending data to a single target data warehouse or lakehouse.
Integrations per source instance	5	The number of integrations sending data from a single source DB instance.

In addition, the target warehouse places certain limits on the number of tables allowed in each DB instance or cluster node. For more information about Amazon Redshift quotas and limits, see Quotas and limits in Amazon Redshift in the *Amazon Redshift Management Guide*.

Supported Regions

RDS zero-ETL integrations are available in a subset of AWS Regions. For a list of supported Regions, see the section called “Zero-ETL integrations ”.

Getting started with Amazon RDS zero-ETL integrations

Before you create a zero-ETL integration, configure your RDS database and your data warehouse with the required parameters and permissions. During setup, you'll complete the following steps:

1. Create a custom DB parameter group.
2. Create a source database.
3. Create a target data warehouse for Amazon Redshift or Create a target Amazon SageMaker Lakehouse.

After you complete these tasks, continue to the section called "Creating zero-ETL integrations with Amazon Redshift" or the section called "Creating zero-ETL integrations with an Amazon SageMaker lakehouse".

> **Tip**
>
> You can have RDS complete these setup steps for you while you're creating the integration, rather than performing them manually. To immediately start creating an integration, see the section called "Creating zero-ETL integrations with Amazon Redshift".

For Step 3, you can choose to create either a target data warehouse (Step 3a) or a target lakehouse (Step 3b) depending on your needs:

- Choose a data warehouse if you need traditional data warehousing capabilities with SQL-based analytics.
- Choose an Amazon SageMaker Lakehouse if you need machine learning capabilities and want to use lakehouse features for data science and ML workflows.

Step 1: Create a custom DB parameter group

Amazon RDS zero-ETL integrations require specific values for the DB parameters that control data replication. The specific parameters depend on your source DB engine. To configure these parameters, you must first create a custom DB parameter group, and then associate it with the source database. Configure the following parameter values depending on your source DB engine. For instructions to create a parameter group, see the section called "DB parameter groups". We recommend that you configure all parameter values within the same request to avoid dependency issues.

RDS for MySQL:

- `binlog_format=ROW`
- `binlog_row_image=full`

In addition, make sure that the `binlog_row_value_options` parameter is *not* set to `PARTIAL_JSON`. If the source database is a Multi-AZ DB cluster, make sure that the `binlog_transaction_compression` parameter is *not* set to `ON`.

Some of these parameters (such as `binlog_format`) are dynamic, meaning you can apply changes to the parameter without triggering a reboot. This means that some existing sessions might continue using the old value of the parameter. To prevent this from causing problems when creating a zero-ETL integration, enable Performance Schema. Performance Schema ensures that zero-ETL pre-checks run, which help detect missing parameters early in the process.

RDS for PostgreSQL:

- `rds.logical_replication = 1`
- `rds.replica_identity_full = 1`
- `session_replication_role = origin`
- `wal_sender_timeout ≥ 20000 or = 0`
- `max_wal_senders ≥ 20`
- `max_replication_slots ≥ 20`

For multiple PostgreSQL integrations, one logical replication slot will be used per integration. Review the `max_replication_slots` and `max_wal_senders` parameters based on your usage.

For efficient data synchronization in zero-ETL integrations, set `rds.replica_identity_full` in your source DB instance. This instructs the database to log complete row data in the write-ahead log (WAL) during `UPDATE` and `DELETE` operations, rather than just primary key information. Zero-ETL requires complete row data even when all replicated tables are required to have primary keys. To determine which data is visible during queries, Amazon Redshift uses a specialized anti-join strategy to compare your data against an internal delete tracking table. Logging full-row images helps Amazon Redshift perform these anti-joins efficiently. Without full row data, Amazon Redshift would need to perform additional lookups, which could slow performance during high-throughput operations in the columnar engine that Amazon Redshift uses.

> **Important**
>
> Setting replica identity to log full rows increases your WAL volume, which can lead to higher write amplification and I/O usage, especially for wide tables or frequent updates. To prepare for these impacts, plan your storage capacity and I/O requirements, monitor your WAL growth, and track replication lag in write-heavy workloads.

RDS for Oracle:

No parameter changes are required for RDS for Oracle.

Step 2: Select or create a source database

After you create a custom DB parameter group, choose or create an RDS DB instance . This database will be the source of data replication to the target data warehouse. For instructions to create a Single-AZ or Multi-AZ DB instance, see the section called “Creating a DB instance”. For instructions to create a Multi-AZ DB cluster (RDS for MySQL only), see the section called “Creating a Multi-AZ DB cluster”.

The database must be running a supported DB engine version. For a list of supported versions, see the section called “Zero-ETL integrations ”.

When you create the database, under **Additional configuration**, change the default **DB parameter group** to the custom parameter group that you created in the previous step.

> **Note**
>
> If you associate the parameter group with the database *after* the database is already created, you must reboot the database to apply the changes before you can create a zero-ETL integration. For instructions, see the section called “Rebooting a DB instance” or the section called “Rebooting a Multi-AZ DB cluster”.

In addition, make sure that automated backups are enabled on the database. For more information, see the section called “Enabling automated backups”.

Step 3a: Create a target data warehouse

After you create your source database, you must create and configure a target data warehouse. The data warehouse must meet the following requirements:

- Using an RA3 node type with at least two nodes, or Redshift Serverless.
- Encrypted (if using a provisioned cluster). For more information, see Amazon Redshift database encryption.

For instructions to create a data warehouse, see Creating a cluster for provisioned clusters, or Creating a workgroup with a namespace for Redshift Serverless.

Enable case sensitivity on the data warehouse

For the integration to be successful, the case sensitivity parameter (`enable_case_sensitive_identifier`) must be enabled for the data warehouse. By default, case sensitivity is disabled on all provisioned clusters and Redshift Serverless workgroups.

To enable case sensitivity, perform the following steps depending on your data warehouse type:

- **Provisioned cluster** – To enable case sensitivity on a provisioned cluster, create a custom parameter group with the `enable_case_sensitive_identifier` parameter enabled. Then, associate the parameter group with the cluster. For instructions, see Managing parameter groups using the console or Configuring parameter values using the AWS CLI.

  > **Note**
  >
  > Remember to reboot the cluster after you associate the custom parameter group with it.

- **Serverless workgroup** – To enable case sensitivity on a Redshift Serverless workgroup, you must use the AWS CLI. The Amazon Redshift console doesn't currently support modifying Redshift Serverless parameter values. Send the following update-workgroup request:

  ```
  aws redshift-serverless update-workgroup \
    --workgroup-name target-workgroup \
    --config-parameters
   parameterKey=enable_case_sensitive_identifier,parameterValue=true
  ```

  You don't need to reboot a workgroup after you modify its parameter values.

Configure authorization for the data warehouse

After you create a data warehouse, you must configure the source RDS database as an authorized integration source. For instructions, see Configure authorization for your Amazon Redshift data warehouse.

Set up an integration using the AWS SDKs

Rather than setting up each resource manually, you can run the following Python script to automatically set up the required resources for you. The code example uses the AWS SDK for Python (Boto3) to create a source RDS for MySQL DB instance and target data warehouse, each

with the required parameter values. It then waits for the databases to be available before creating a zero-ETL integration between them. You can comment out different functions depending on which resources you need to set up.

To install the required dependencies, run the following commands:

```
pip install boto3
pip install time
```

Within the script, optionally modify the names of the source, target, and parameter groups. The final function creates an integration named `my-integration` after the resources are set up.

Python code example

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_db_name = 'my-source-db' # A name for the source database
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_db(*args):
    """Creates a source RDS for MySQL DB instance"""

    response = rds.create_db_parameter_group(
        DBParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='mysql8.0',
        Description='RDS for MySQL zero-ETL integrations'
    )
    print('Created source parameter group: ' + response['DBParameterGroup']
['DBParameterGroupName'])
```

```
    response = rds.modify_db_parameter_group(
        DBParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'binlog_format',
                'ParameterValue': 'ROW',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_image',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' + response['DBParameterGroupName'])

    response = rds.create_db_instance(
        DBInstanceIdentifier=source_db_name,
        DBParameterGroupName=source_param_group_name,
        Engine='mysql',
        EngineVersion='8.0.32',
        DBName='mydb',
        DBInstanceClass='db.m5.large',
        AllocatedStorage=15,
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source database: ' + response['DBInstance']
['DBInstanceIdentifier'])
    source_arn = (response['DBInstance']['DBInstanceArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='RDS for MySQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])
```

```
response = redshift.modify_cluster_parameter_group(
    ParameterGroupName=target_param_group_name,
    Parameters=[
        {
            'ParameterName': 'enable_case_sensitive_identifier',
            'ParameterValue': 'true'
        }
    ]
)
print('Modified target parameter group: ' + response['ParameterGroupName'])

response = redshift.create_cluster(
    ClusterIdentifier=target_cluster_name,
    NodeType='ra3.4xlarge',
    NumberOfNodes=2,
    Encrypted=True,
    MasterUsername='username',
    MasterUserPassword='Password01**',
    ClusterParameterGroupName=target_param_group_name
)
print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

# Retrieve the target cluster ARN
response = redshift.describe_clusters(
    ClusterIdentifier=target_cluster_name
)
target_arn = response['Clusters'][0]['ClusterNamespaceArn']

# Retrieve the current user's account ID
response = sts.get_caller_identity()
account_id = response['Account']

# Create a resource policy granting access to source database and account ID
response = redshift.put_resource_policy(
    ResourceArn=target_arn,
    Policy='''
    {
        \"Version\":\"2012-10-17\",
        \"Statement\":[
            {\"Effect\":\"Allow\",
            \"Principal\":{
                \"Service\":\"redshift.amazonaws.com\"
            },
```

```
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_db_availability(*args):
    """Waits for both databases to be available"""

    print('Waiting for source and target to be available...')

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_db_name
    )
    source_status = response['DBInstances'][0]['DBInstanceStatus']
    source_arn = response['DBInstances'][0]['DBInstanceArn']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the databases are available
    if source_status != 'available' or target_status != 'available':
        time.sleep(60)
        response = wait_for_db_availability(
            source_db_name, target_cluster_name)
    else:
        print('Databases available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
```

```
    """Creates a zero-ETL integration using the source and target databases"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_db(source_db_name, source_param_group_name)
    wait_for_db_availability(source_db_name, target_cluster_name)

if __name__ == "__main__":
    main()
```

Step 3b: Create an AWS Glue catalog for Amazon SageMaker Lakehouse zero-ETL integration

When creating a zero-ETL integration with an Amazon SageMaker Lakehouse, you must create an AWS Glue managed catalog in AWS Lake Formation. The target catalog must be an Amazon Redshift managed catalog. To create an Amazon Redshift managed catalog, first create the `AWSServiceRoleForRedshift` service-linked role. In the Lake Formation console, add the `AWSServiceRoleForRedshift` as a read-only administrator.

For more information about the previous tasks, see the following topics.

- For information about creating an Amazon Redshift managed catalog, see Creating an Amazon Redshift managed catalog in the AWS Glue Data Catalog in the *AWS Lake Formation Developer Guide*.
- For information about the service-linked role for Amazon Redshift, see Using service-linked roles for Amazon Redshift in the *Amazon Redshift Management Guide*.
- For information about read-only administrator permissions for Lake Formation, see Lake Formation personas and IAM permissions reference in the *AWS Lake Formation Developer Guide*.

Configure permissions for the target AWS Glue catalog

Before creating a target catalog for zero-ETL integration, you must create the Lake Formation target creation role and the AWS Glue data transfer role. Use the Lake Formation target creation role to create the target catalog. When creating the target catalog, enter the Glue data transfer role in the **IAM role** field in the **Access from engines section**.

Lake Formation target creation role

The target creation role must be a Lake Formation administrator and requires the following permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "lakeformation:RegisterResource",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "s3:PutEncryptionConfiguration",
                "iam:PassRole",
                "glue:CreateCatalog",
                "glue:GetCatalog",
                "s3:PutBucketTagging",
                "s3:PutLifecycleConfiguration",
                "s3:PutBucketPolicy",
                "s3:CreateBucket",
                "redshift-serverless:CreateNamespace",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "redshift-serverless:CreateWorkgroup"
            ],
            "Resource": [
                "arn:aws:glue:*:111122223333:catalog",
```

```
                "arn:aws:glue:*:111122223333:catalog/*",
                "arn:aws:s3:::*",
                "arn:aws:redshift-serverless:*:111122223333:workgroup/*",
                "arn:aws:redshift-serverless:*:111122223333:namespace/*",
                "arn:aws:iam::111122223333:role/GlueDataCatalogDataTransferRole"
            ]
        }
    ]
}
```

The target creation role must have the following trust relationship.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
          "Effect": "Allow",
          "Principal": {
            "AWS": "arn:aws:iam::111122223333:user/Username"
          },
          "Action": "sts:AssumeRole"
        }
    ]
}
```

Glue data transfer role

The Glue data transfer role is required for MySQL catalog operations and must have the following permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DataTransferRolePolicy",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "glue:GetCatalog",
                "glue:GetDatabase"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

The Glue data transfer role must have the following trust relationship.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "glue.amazonaws.com",
                    "redshift.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Next steps

With a source RDS database and either an Amazon Redshift target data warehouse or Amazon SageMaker Lakehouse, you can create a zero-ETL integration and replicate data. For instructions, see the section called "Creating zero-ETL integrations with Amazon Redshift".

Creating Amazon RDS zero-ETL integrations with Amazon Redshift

When you create an Amazon RDS zero-ETL integration, you specify the source RDS database and the target Amazon Redshift data warehouse. You can also customize encryption settings and add tags. Amazon RDS creates an integration between the source database and its target. Once the integration is active, any data that you insert into the source database will be replicated into the configured Amazon Redshift target.

Prerequisites

Before you create a zero-ETL integration, you must create a source database and a target Amazon Redshift data warehouse. You also must allow replication into the data warehouse by adding the database as an authorized integration source.

For instructions to complete each of these steps, see the section called "Getting started with zero-ETL integrations".

Required permissions

Certain IAM permissions are required to create a zero-ETL integration. At minimum, you need permissions to perform the following actions:

- Create zero-ETL integrations for the source RDS database.
- View and delete all zero-ETL integrations.
- Create inbound integrations into the target data warehouse.

The following sample policies demonstrate the least privilege permissions required to create and manage integrations. You might not need these exact permissions if your user or role has broader permissions, such as an `AdministratorAccess` managed policy.

Note

Redshift Amazon Resource Names (ARNs) have the following format. Note the use of a forward slash (`/`) rather than a colon (`:`) before the serverless namespace UUID.

- Provisioned cluster – `arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid`
- Serverless – `arn:aws:redshift-serverless:{region}:{account-id}:namespace/namespace-uuid`

Sample policy for Redshift target

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CreateIntegration",
      "Effect": "Allow",
      "Action": [
        "rds:CreateIntegration"
      ],
      "Resource": [
        "arn:aws:rds:us-east-1:123456789012:db:source-db",
        "arn:aws:rds:us-east-1:123456789012:integration:*"
      ]
    },
    {
      "Sid": "DescribeIntegrationDetails",
      "Effect": "Allow",
      "Action": [
        "rds:DescribeIntegrations"
      ],
      "Resource": [
      "arn:aws:rds:us-east-1:123456789012:integration:*"
  ]
    },
    {
```

```
      "Sid": "ChangeIntegrationDetails",
      "Effect": "Allow",
      "Action": [
        "rds:DeleteIntegration",
        "rds:ModifyIntegration"
      ],
      "Resource": [
        "arn:aws:rds:us-east-1:123456789012:integration:*"
      ]
    },
    {
      "Sid": "AllowRedShiftIntegration",
      "Effect": "Allow",
      "Action": [
        "redshift:CreateInboundIntegration"
      ],
      "Resource": [
        "arn:aws:redshift:us-east-1:123456789012:namespace:namespace-uuid"
      ]
    }
  ]
}
```

Choosing a target data warehouse in a different account

If you plan to specify a target Amazon Redshift data warehouse that's in another AWS account, you must create a role that allows users in the current account to access resources in the target account. For more information, see Providing access to an IAM user in another AWS account that you own.

The role must have the following permissions, which allow the user to view available Amazon Redshift provisioned clusters and Redshift Serverless namespaces in the target account.

Required permissions and trust policy

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
```

```
         "Effect":"Allow",
         "Action":[
            "redshift:DescribeClusters",
            "redshift-serverless:ListNamespaces"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

The role must have the following trust policy, which specifies the target account ID.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

For instructions to create the role, see Creating a role using custom trust policies.

Creating zero-ETL integrations

You can create a zero-ETL integration using the AWS Management Console, the AWS CLI, or the RDS API.

> **Important**
>
> zero-ETL integrations do not support refresh or resync operations. If you encounter issues with an integration after creation, you must delete the integration and create a new one.

By default, RDS for MySQL immediately purges binary log files. Because zero-ETL integrations rely on binary logs to replicate data from the source to the target, the retention period for the source database must be at least one hour. As soon as you create an integration, Amazon RDS checks the binary log file retention period for the selected source database. If the current value is 0 hours, Amazon RDS automatically changes it to 1 hour. Otherwise, the value remains the same.

RDS console

To create a zero-ETL integration

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Zero-ETL integrations**.
3. Choose **Create zero-ETL integration**.
4. For **Integration identifier**, enter a name for the integration. The name can have up to 63 alphanumeric characters and can include hyphens.

   > **Important**
   >
   > Catalog names are limited to 19 characters in length. Ensure your integration identifier meets this requirement if it will be used as a catalog name.

5. Choose **Next**.
6. For **Source**, select the RDS database where the data will originate from.

   > **Note**
   >
   > RDS notifies you if the DB parameters aren't configured correctly. If you receive this message, you can either choose **Fix it for me**, or configure them manually. For instructions to fix them manually, see the section called “Step 1: Create a custom DB parameter group”.

Modifying DB parameters requires a reboot. Before you can create the integration, the reboot must be complete and the new parameter values must be successfully applied to the database.

7. Once your source database is successfully configured, choose **Next**.
8. For **Target**, do the following:

   1. (Optional) To use a different AWS account for the Amazon Redshift target, choose **Specify a different account**. Then, enter the ARN of an IAM role with permissions to display your data warehouses. For instructions to create the IAM role, see the section called "Choosing a target data warehouse in a different account".
   2. For **Amazon Redshift data warehouse**, select the target for replicated data from the source database. You can choose a provisioned Amazon Redshift *cluster* or a Redshift Serverless *namespace* as the target.

   Note

   RDS notifies you if the resource policy or case sensitivity settings for the specified data warehouse aren't configured correctly. If you receive this message, you can either choose **Fix it for me**, or configure them manually. For instructions to fix them manually, see Turn on case sensitivity for your data warehouse and Configure authorization for your data warehouse in the *Amazon Redshift Management Guide*.
   Modifying case sensitivity for a *provisioned* Redshift cluster requires a reboot. Before you can create the integration, the reboot must be complete and the new parameter value must be successfully applied to the cluster.
   If your selected source and target are in different AWS accounts, then Amazon RDS cannot fix these settings for you. You must navigate to the other account and fix them manually in Amazon Redshift.

9. Once your target data warehouse is configured correctly, choose **Next**.
10. (Optional) For **Tags**, add one or more tags to the integration. For more information, see the section called "Tagging RDS resources".
11. For **Encryption**, specify how you want your integration to be encrypted. By default, RDS encrypts all integrations with an AWS owned key. To choose a customer managed key instead, enable **Customize encryption settings** and choose a KMS key to use for encryption. For more information, see the section called "Encrypting Amazon RDS resources".

Optionally, add an encryption context. For more information, see Encryption context in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> Amazon RDS adds the following encryption context pairs in addition to any that you add:
>
> - `aws:redshift:integration:arn` - `IntegrationArn`
> - `aws:servicename:id` - `Redshift`
>
> This reduces the overall number of pairs that you can add from 8 to 6, and contributes to the overall character limit of the grant constraint. For more information, see Using grant constraints in the *AWS Key Management Service Developer Guide*.

12. Choose **Next**.
13. Review your integration settings and choose **Create zero-ETL integration**.

    If creation fails, see the section called "I can't create a zero-ETL integration" for troubleshooting steps.

The integration has a status of `Creating` while it's being created, and the target Amazon Redshift data warehouse has a status of `Modifying`. During this time, you can't query the data warehouse or make any configuration changes on it.

When the integration is successfully created, the status of the integration and the target Amazon Redshift data warehouse both change to `Active`.

AWS CLI

To create a zero-ETL integration using the AWS CLI, use the create-integration command with the following options:

> **Note**
>
> Remember that catalog names are limited to 19 characters. Choose your integration name accordingly if it will be used as a catalog name.

- `--integration-name` – Specify a name for the integration.
- `--source-arn` – Specify the ARN of the RDS database that will be the source for the integration.
- `--target-arn` – Specify the ARN of the Amazon Redshift data warehouse that will be the target for the integration.

Example

For Linux, macOS, or Unix:

```
aws rds create-integration \
    --integration-name my-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-db \
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

For Windows:

```
aws rds create-integration ^
    --integration-name my-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-db ^
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

RDS API

To create a zero-ETL integration by using the Amazon RDS API, use the `CreateIntegration` operation with the following parameters:

> **Note**
>
> Catalog names are limited to 19 characters. Ensure your IntegrationName parameter meets this requirement if it will be used as a catalog name.

- `IntegrationName` – Specify a name for the integration.
- `SourceArn` – Specify the ARN of the RDS database that will be the source for the integration.
- `TargetArn` – Specify the ARN of the Amazon Redshift data warehouse that will be the target for the integration.

Encrypting integrations with a customer managed key

If you specify a custom KMS key rather than an AWS owned key when you create an integration, the key policy must provide the Amazon Redshift service principal access to the `CreateGrant` action. In addition, it must allow the current user to perform to the `DescribeKey` and `CreateGrant` actions.

The following sample policy demonstrates how to provide the required permissions in the key policy. It includes context keys to further reduce the scope of permissions.

Sample key policy

JSON

```
{
    "Version":"2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "Enables IAM user permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allows the Redshift service principal to add a grant to a KMS
 key",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}": "{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
```

```
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current user or role to add a grant to a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}": "{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "Allows the current uer or role to retrieve information about
 a KMS key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

For more information, see Creating a key policy in the *AWS Key Management Service Developer Guide*.

Next steps

After you successfully create a zero-ETL integration, you must create a destination database within your target Amazon Redshift cluster or workgroup. Then, you can start adding data to the source RDS database and querying it in Amazon Redshift. For instructions, see Creating destination databases in Amazon Redshift.

Creating Amazon RDS zero-ETL integrations with an Amazon SageMaker lakehouse

When you create an Amazon RDS zero-ETL integration with an Amazon SageMaker lakehouse, you specify the source RDS database and the target AWS Glue managed catalog. You can also customize encryption settings and add tags. Amazon RDS creates an integration between the source database and its target. Once the integration is active, any data that you insert into the source database will be replicated into the configured target.

Prerequisites

Before you create a zero-ETL integration with an Amazon SageMaker lakehouse, you must create a source database and a target AWS Glue managed catalog. You also must allow replication into the catalog by adding the database as an authorized integration source.

For instructions to complete each of these steps, see the section called “Getting started with zero-ETL integrations”.

Required permissions

Certain IAM permissions are required to create a zero-ETL integration with an Amazon SageMaker lakehouse. At minimum, you need permissions to perform the following actions:

- Create zero-ETL integrations for the source RDS database.
- View and delete all zero-ETL integrations.
- Create inbound integrations into the target AWS Glue managed catalog.
- Access Amazon S3 buckets used by the AWS Glue managed catalog.

- Use AWS KMS keys for encryption if custom encryption is configured.
- Register resources with Lake Formation.
- Put resource policy on the AWS Glue managed catalog to authorize inbound integrations.

The following sample policy demonstrates the least privilege permissions required to create and manage integrations with an Amazon SageMaker lakehouse. You might not need these exact permissions if your user or role has broader permissions, such as an `AdministratorAccess` managed policy.

Additionally, you must configure a resource policy on the target AWS Glue managed catalog to authorize inbound integrations. Use the following AWS CLI command to apply the resource policy.

Sample AWS CLI command to authorize inbound integrations on the target catalog

```
aws glue put-resource-policy \
      --policy-in-json  '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "glue.amazonaws.com"
        },
        "Action": [
            "glue:AuthorizeInboundIntegration"
        ],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"],
        "Condition": {
            "StringEquals": {
                "aws:SourceArn": "arn:aws:rds:region:account_id:db:source_name"
            }
        }
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "account_id"
        },
        "Action": ["glue:CreateInboundIntegration"],
        "Resource": ["arn:aws:glue:region:account_id:catalog/catalog_name"]
    }
    ]
```

```
}' \
      --region region
```

> **Note**
>
> Glue catalog Amazon Resource Names (ARNs) have the following format:
>
> - Glue catalog – `arn:aws:glue:{region}:{account-id}:catalog/catalog-name`

Choosing a target AWS Glue managed catalog in a different account

If you plan to specify a target AWS Glue managed catalog that's in another AWS account, you must create a role that allows users in the current account to access resources in the target account. For more information, see Providing access to an IAM user in another AWS account that you own.

The role must have the following permissions, which allow the user to view available AWS Glue catalogs in the target account.

Required permissions and trust policy

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "glue:GetCatalog"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

The role must have the following trust policy, which specifies the target account ID.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::111122223333:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

For instructions to create the role, see Creating a role using custom trust policies.

Creating zero-ETL integrations with an Amazon SageMaker lakehouse

You can create a zero-ETL integration with an Amazon SageMaker lakehouse using the AWS Management Console, the AWS CLI, or the RDS API.

> **Important**
>
> Zero-ETL integrations with an Amazon SageMaker lakehouse do not support refresh or resync operations. If you encounter issues with an integration after creation, you must delete the integration and create a new one.

By default, RDS for MySQL immediately purges binary log files. Because zero-ETL integrations rely on binary logs to replicate data from the source to the target, the retention period for the source database must be at least one hour. As soon as you create an integration, Amazon RDS checks the binary log file retention period for the selected source database. If the current value is 0 hours, Amazon RDS automatically changes it to 1 hour. Otherwise, the value remains the same.

RDS console

To create a zero-ETL integration with an Amazon SageMaker lakehouse

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the left navigation pane, choose **Zero-ETL integrations**.
3. Choose **Create zero-ETL integration**.
4. For **Integration identifier**, enter a name for the integration. The name can have up to 63 alphanumeric characters and can include hyphens.
5. Choose **Next**.
6. For **Source**, select the RDS database where the data will originate from.

> **Note**
>
> RDS notifies you if the DB parameters aren't configured correctly. If you receive this message, you can either choose **Fix it for me**, or configure them manually. For instructions to fix them manually, see the section called "Step 1: Create a custom DB parameter group".
> Modifying DB parameters requires a reboot. Before you can create the integration, the reboot must be complete and the new parameter values must be successfully applied to the database.

7. Once your source database is successfully configured, choose **Next**.
8. For **Target**, do the following:

    1. (Optional) To use a different AWS account for the Amazon SageMaker lakehouse target, choose **Specify a different account**. Then, enter the ARN of an IAM role with permissions to display your AWS Glue catalogs. For instructions to create the IAM role, see the section called "Choosing a target AWS Glue managed catalog in a different account".
    2. For **AWS Glue catalog**, select the target for replicated data from the source database. You can choose an existing AWS Glue managed catalog as the target.
    3. The target IAM role needs describe permissions on the target catalog and must have the following permissions:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "glue:GetCatalog",
            "Resource": [
                "arn:aws:glue:us-east-1:111122223333:catalog/*",
                "arn:aws:glue:us-east-1:111122223333:catalog"
            ]
        }
    ]
}
```

The target IAM role must have the following trust relationship:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

4. You must grant the target IAM role describe permissions for the target AWS Glue managed catalog with the Lake Formation administrator role created in Step 3b: Create an AWS Glue catalog for Amazon SageMaker Lakehouse zero-ETL integration.

> **Note**
>
> RDS notifies you if the resource policy or configuration settings for the specified AWS Glue managed catalog aren't configured correctly. If you receive this message, you can either choose **Fix it for me**, or configure them manually.
> If your selected source and target are in different AWS accounts, then Amazon RDS cannot fix these settings for you. You must navigate to the other account and fix them manually in SageMaker Unified Studio.

9. Once your target AWS Glue managed catalog is configured correctly, choose **Next**.
10. (Optional) For **Tags**, add one or more tags to the integration. For more information, see the section called "Tagging RDS resources".
11. For **Encryption**, specify how you want your integration to be encrypted. By default, RDS encrypts all integrations with an AWS owned key. To choose a customer managed key instead, enable **Customize encryption settings** and choose a KMS key to use for encryption. For more information, see the section called "Encrypting Amazon RDS resources".

    Optionally, add an encryption context. For more information, see Encryption context in the *AWS Key Management Service Developer Guide*.

    > **Note**
    >
    > Amazon RDS adds the following encryption context pairs in addition to any that you add:
    >
    > - `aws:glue:integration:arn` - `IntegrationArn`
    > - `aws:servicename:id` - `glue`
    >
    > This reduces the overall number of pairs that you can add from 8 to 6, and contributes to the overall character limit of the grant constraint. For more information, see Using grant constraints in the *AWS Key Management Service Developer Guide*.

12. Choose **Next**.
13. Review your integration settings and choose **Create zero-ETL integration**.

If creation fails, see the section called "Troubleshooting zero-ETL integrations" for troubleshooting steps.

The integration has a status of `Creating` while it's being created, and the target Amazon SageMaker lakehouse has a status of `Modifying`. During this time, you can't query the catalog or make any configuration changes on it.

When the integration is successfully created, the status of the integration and the target Amazon SageMaker lakehouse both change to `Active`.

AWS CLI

To prepare a target AWS Glue managed catalog for zero-ETL integration using the AWS CLI, you must first use the create-integration-resource-property command with the following options:

- `--resource-arn` – Specify the ARN of the AWS Glue managed catalog that will be the target for the integration.
- `--target-processing-properties` – Specify the ARN of the IAM role to access the target AWS Glue managed catalog

```
aws glue create-integration-resource-property --region us-east-1
 --resource-arn arn:aws:glue:region:account_id:catalog/catalog_name \
 --target-processing-properties '{"RoleArn" : "arn:aws:iam::account_id:role/
TargetIamRole"}'
```

To create a zero-ETL integration with an Amazon SageMaker lakehouse using the AWS CLI, use the create-integration command with the following options:

- `--integration-name` – Specify a name for the integration.
- `--source-arn` – Specify the ARN of the RDS database that will be the source for the integration.
- `--target-arn` – Specify the ARN of the AWS Glue managed catalog that will be the target for the integration.

Example

For Linux, macOS, or Unix:

```
aws rds create-integration \
    --integration-name my-sagemaker-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:cluster:my-db \
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

For Windows:

```
aws rds create-integration ^
    --integration-name my-sagemaker-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:cluster:my-db ^
    --target-arn arn:aws:glue:{region}:{account-id}:catalog/catalog-name
```

RDS API

To create a zero-ETL integration with Amazon SageMaker by using the Amazon RDS API, use the `CreateIntegration` operation with the following parameters:

> **Note**
>
> Catalog names are limited to 19 characters. Ensure your IntegrationName parameter meets this requirement if it will be used as a catalog name.

- `IntegrationName` – Specify a name for the integration.
- `SourceArn` – Specify the ARN of the RDS database that will be the source for the integration.
- `TargetArn` – Specify the ARN of the AWS Glue managed catalog that will be the target for the integration.

Encrypting integrations with a customer managed key

If you specify a custom KMS key rather than an AWS owned key when you create an integration with Amazon SageMaker, the key policy must provide the SageMaker Unified Studio service principal access to the `CreateGrant` action. In addition, it must allow the current user to perform to the `DescribeKey` and `CreateGrant` actions.

The following sample policy demonstrates how to provide the required permissions in the key policy. It includes context keys to further reduce the scope of permissions.

Sample key policy

JSON

```
{
    "Version":"2012-10-17",
    "Id": "Key policy",
    "Statement": [
        {
            "Sid": "EnablesIAMUserPermissions",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "GlueServicePrincipalAddGrant",
            "Effect": "Allow",
            "Principal": {
                "Service": "glue.amazonaws.com"
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "AllowsCurrentUserRoleAddGrantKMSKey",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:role/{role-name}"
```

```
            },
            "Action": "kms:CreateGrant",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:{context-key}":"{context-value}",
                    "kms:ViaService": "rds.us-east-1.amazonaws.com"
                },
                "ForAllValues:StringEquals": {
                    "kms:GrantOperations": [
                        "Decrypt",
                        "GenerateDataKey",
                        "CreateGrant"
                    ]
                }
            }
        },
        {
            "Sid": "AllowsCurrentUserRoleRetrieveKMSKeyInformation",
            "Effect": "Allow",
            "Principal": {
            "AWS": "arn:aws:iam::111122223333:role/{role-name}"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*"
        }
    ]
}
```

For more information, see Creating a key policy in the *AWS Key Management Service Developer Guide*.

Next steps

After you successfully create a zero-ETL integration with Amazon SageMaker, you can start adding data to the source RDS database and querying it in your Amazon SageMaker lakehouse. The data will be automatically replicated and made available for analytics and machine learning workloads.

Data filtering for Amazon RDS zero-ETL integrations

Amazon RDS zero-ETL integrations support data filtering, which lets you control which data is replicated from your source Amazon RDS database to your target data warehouse. Instead of replicating the entire database, you can apply one or more filters to selectively include or exclude specific tables. This helps you optimize storage and query performance by ensuring that only relevant data is transferred. Currently, filtering is limited to the database and table levels. Column- and row-level filtering are not supported.

Data filtering can be useful when you want to:

- Join certain tables from two or more different source databases, and you don't need complete data from either database.
- Save costs by performing analytics using only a subset of tables rather than an entire fleet of databases.
- Filter out sensitive information—such as phone numbers, addresses, or credit card details—from certain tables.

You can add data filters to a zero-ETL integration using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the Amazon RDS API.

If the integration has a provisioned cluster as its target, the cluster must be on patch 180 or higher to use data filtering.

Topics

- Format of a data filter
- Filter logic
- Filter precedence
- RDS for MySQL examples
- RDS for PostgreSQL examples
- RDS for Oracle examples
- Adding data filters to an integration
- Removing data filters from an integration

Format of a data filter

You can define multiple filters for a single integration. Each filter either includes or excludes any existing and future database tables that match one of the patterns in the filter expression. Amazon RDS zero-ETL integrations use Maxwell filter syntax for data filtering.

Each filter has the following elements:

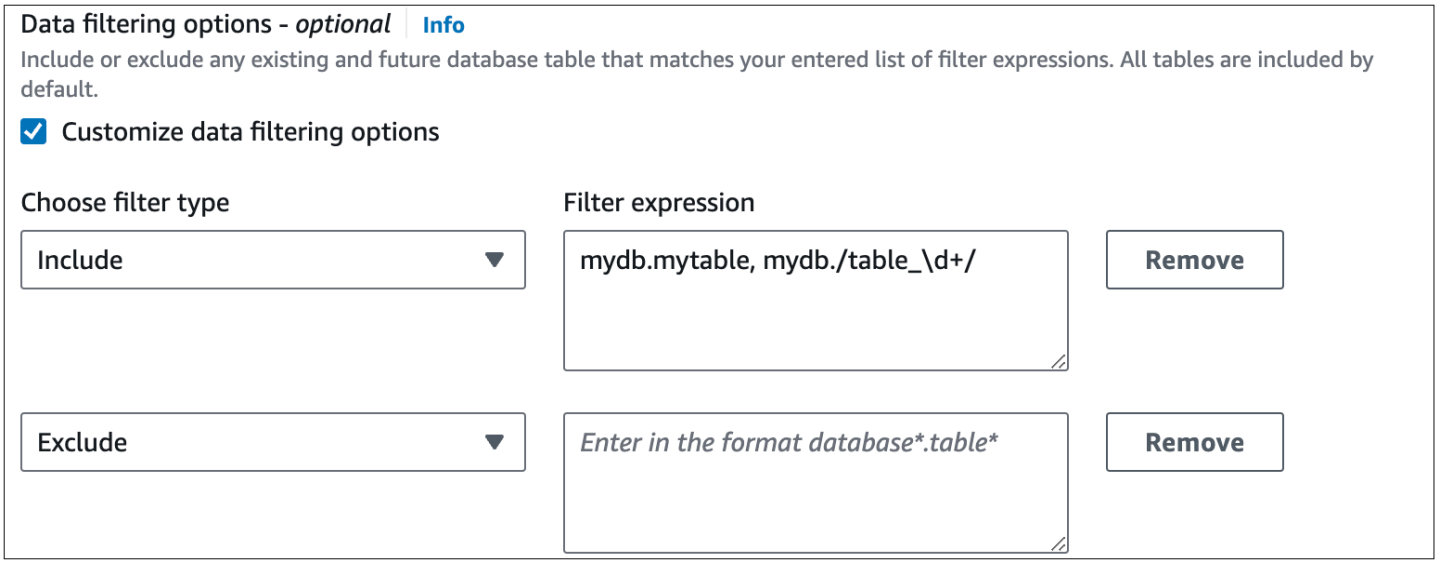
Element	Description
Filter type	An `Include` filter type *includes* all tables that match one of the patterns in the filter expression. An `Exclude` filter type *excludes* all tables that match one of the patterns.
Filter expression	A comma-separated list of patterns. Expressions must use Maxwell filter syntax.
Pattern	A filter pattern in the format *`database`*`.`*`table`* for RDS for MySQL, or *`database`*`.`*`schema`*`.`*`table`* for RDS for PostgreSQL. You can specify literal names, or define regular expressions. **Note** For RDS for MySQL, regular expressions are supported in both the database and table name. For RDS for PostgreSQL, regular expressions are supported only in the schema and table name, not in the database name. You can't include column-level filters or denylists.

Element	Description
	A single integration can have a maximum of 99 total patterns. In the console, you can enter patterns within a single filter expressio n, or spread them out among multiple expressions. A single pattern can't exceed 256 characters in length.

Important

If you select an RDS for PostgreSQL source database, you must specify at least one data filter pattern. At minimum, the pattern must include a single database (*`database-name`*`.*.*`) for replication to the target data warehouse.

The following image shows the structure of RDS for MySQL data filters in the console:



Important

Do not include personally identifying, confidential, or sensitive information in your filter patterns.

Data filters in the AWS CLI

When using the AWS CLI to add a data filter, the syntax differs slightly from the console. You must assign a filter type (`Include` or `Exclude`) to each pattern individually, so you can't group multiple patterns under one filter type.

For example, in the console you can group the following comma-separated patterns under a single `Include` statement:

RDS for MySQL

```
mydb.mytable, mydb./table_\d+/
```

RDS for PostgreSQL

```
mydb.myschema.mytable, mydb.myschema./table_\d+/
```

However, when using the AWS CLI, the same data filter must be in the following format:

RDS for MySQL

```
'include: mydb.mytable, include: mydb./table_\d+/'
```

RDS for PostgreSQL

```
'include: mydb.myschema.mytable, include: mydb.myschema./table_\d+/'
```

Filter logic

If you don't specify any data filters in your integration, Amazon RDS assumes a default filter of `include:*.*`, which replicates all tables to the target data warehouse. However, if you add at least one filter, the default logic switches to `exclude:*.*`, which excludes all tables by default. This lets you explicitly define which databases and tables to include in replication.

For example, if you define the following filter:

```
'include: db.table1, include: db.table2'
```

Amazon RDS evaluates the filter as follows:

```
'exclude:*.*, include: db.table1, include: db.table2'
```

Therefore, Amazon RDS only replicates `table1` and `table2` from the database named db to the target data warehouse.

Filter precedence

Amazon RDS evaluates data filters in the order you specify. In the AWS Management Console, it processes filter expressions from left to right and top to bottom. A second filter or an individual pattern that follows the first can override it.

For example, if the first filter is `Include books.stephenking`, it includes only the `stephenking` table from the `books` database. However, if you add a second filter, `Exclude books.*`, it overrides the first filter. This prevents any tables from the `books` index from being replicated to the target data warehouse.

When you specify at least one filter, the logic starts by assuming `exclude:*.*` by default, which automatically *excludes* all tables from replication. As a best practice, define filters from broadest to most specific. Start with one or more `Include` statements to specify the data to replicate, then add `Exclude` filters to selectively remove certain tables.

The same principle applies to filters that you define using the AWS CLI. Amazon RDS evaluates these filter patterns in the order that you specify them, so a pattern might override one that you specify before it.

RDS for MySQL examples

The following examples demonstrate how data filtering works for RDS for MySQL examples zero-ETL integrations:

- Include all databases and all tables:

  ```
  'include: *.*'
  ```

- Include all tables within the `books` database:

  ```
  'include: books.*'
  ```

- Exclude any tables named `mystery`:

```
'include: *.*, exclude: *.mystery'
```

- Include two specific tables within the `books` database:

```
'include: books.stephen_king, include: books.carolyn_keene'
```

- Include all tables in the `books` database, except for those containing the substring `mystery`:

```
'include: books.*, exclude: books./.*mystery.*/'
```

- Include all tables in the `books` database, except those starting with `mystery`:

```
'include: books.*, exclude: books./mystery.*/'
```

- Include all tables in the `books` database, except those ending with `mystery`:

```
'include: books.*, exclude: books./.*mystery/'
```

- Include all tables in the `books` database that start with `table_`, except for the one named `table_stephen_king`. For example, `table_movies` or `table_books` would be replicated, but not `table_stephen_king`.

```
'include: books./table_.*/, exclude: books.table_stephen_king'
```

RDS for PostgreSQL examples

The following examples demonstrate how data filtering works for RDS for PostgreSQL zero-ETL integrations:

- Include all tables within the `books` database:

```
'include: books.*.*'
```

- Exclude any tables named `mystery` in the `books` database:

```
'include: books.*.*, exclude: books.*.mystery'
```

- Include one table within the `books` database in the `mystery` schema, and one table within `employee` database in the `finance` schema:

```
'include: books.mystery.stephen_king, include: employee.finance.benefits'
```

- Include all tables in the books database and `science_fiction` schema, except for those containing the substring `king`:

```
'include: books.science_fiction.*, exclude: books.*./.*king.*/
```

- Include all tables in the books database, except those with a schema name starting with `sci`:

```
'include: books.*.*, exclude: books./sci.*/.*'
```

- Include all tables in the books database, except those in the `mystery` schema ending with `king`:

```
'include: books.*.*, exclude: books.mystery./.*king/'
```

- Include all tables in the books database that start with `table_`, except for the one named `table_stephen_king`. For example, `table_movies` in the `fiction` schema and `table_books` in the `mystery` schema are replicated, but not `table_stephen_king` in either schema:

```
'include: books.*./table_.*/, exclude: books.*.table_stephen_king'
```

RDS for Oracle examples

The following examples demonstrate how data filtering works for RDS for Oracle zero-ETL integrations:

- Include all tables within the books database:

```
'include: books.*.*'
```

- Exclude any tables named mystery in the books database:

```
'include: books.*.*, exclude: books.*.mystery'
```

- Include one table within the books database in the mystery schema, and one table within employee database in the finance schema:

```
'include: books.mystery.stephen_king, include: employee.finance.benefits'
```

- Include all tables in the mystery schema within the books database:

```
'include: books.mystery.*'
```

Case sensitivity considerations

Oracle Database and Amazon Redshift handle object name casing differently, which affects both data filter configuration and target queries. Note the following:

- Oracle Database stores database, schema, and object names in uppercase unless explicitly quoted in the `CREATE` statement. For example, if you create `mytable` (no quotes), the Oracle data dictionary stores the table name as `MYTABLE`. If you quote the object name, the data dictionary preserves the case.
- Zero-ETL data filters are case sensitive and must match the exact case of object names as they appear in the Oracle data dictionary.
- Amazon Redshift queries default to lowercase object names unless explicitly quoted. For example, a query of `MYTABLE` (no quotes) searches for `mytable`.

Be mindful of the case differences when you create the Amazon Redshift filter and query the data.

Creating an uppercase integration

When you create a table without specifying the name in double quotes, the Oracle database stores the name in uppercase in the data dictionary. For example, you can create `MYTABLE` using any of the following SQL statements.

```
CREATE TABLE REINVENT.MYTABLE (id NUMBER PRIMARY KEY, description VARCHAR2(100));
CREATE TABLE reinvent.mytable (id NUMBER PRIMARY KEY, description VARCHAR2(100));
CREATE TABLE REinvent.MyTable (id NUMBER PRIMARY KEY, description VARCHAR2(100));
CREATE TABLE reINVENT.MYtabLE (id NUMBER PRIMARY KEY, description VARCHAR2(100));
```

Because you didn't quote the table name in the preceding statements, the Oracle database stores the object name in uppercase as `MYTABLE`.

To replicate this table to Amazon Redshift, you must specify the uppercase name in your data filter of your `create-integration` command. The Zero-ETL filter name and Oracle data dictionary name must match.

```
aws rds create-integration \
  --integration-name upperIntegration \
  --data-filter "include: ORCL.REINVENT.MYTABLE" \
...
```

By default, Amazon Redshift stores data in lowercase. To query `MYTABLE` in the replicated database in Amazon Redshift, you must quote the uppercase name `MYTABLE` so that it matches the case in the Oracle data dictionary.

```
SELECT * FROM targetdb1."REINVENT"."MYTABLE";
```

The following queries don't use the quoting mechanism. They all return an error because they search for an Amazon Redshift table named `mytable`, which uses the default lowercase name, but the table is named `MYTABLE` in the Oracle data dictionary.

```
SELECT * FROM targetdb1."REINVENT".MYTABLE;
SELECT * FROM targetdb1."REINVENT".MyTable;
SELECT * FROM targetdb1."REINVENT".mytable;
```

The following queries uses the quoting mechanism to specify a mixed case name. The queries all return an error because they search for an Amazon Redshift table that isn't named `MYTABLE`.

```
SELECT * FROM targetdb1."REINVENT"."MYtablE";
SELECT * FROM targetdb1."REINVENT"."MyTable";
SELECT * FROM targetdb1."REINVENT"."mytable";
```

Creating a lowercase integration

In the following alternative example, you use double quotes to store the table name in lowercase in the Oracle data dictionary. You create `mytable` as follows.

```
CREATE TABLE REINVENT."mytable" (id NUMBER PRIMARY KEY, description VARCHAR2(100));
```

The Oracle database stores the table name as `mytable` in lowercase. To replicate this table to Amazon Redshift, you must specify the lowercase name `mytable` in your Zero-ETL data filter.

```
aws rds create-integration \
  --integration-name lowerIntegration \
  --data-filter "include: ORCL.REINVENT.mytable" \
...
```

When you query this table in the replicated database in Amazon Redshift, you can specify the lowercase name `mytable`. The query succeeds because it searches for a table named `mytable`, which is the table name in the Oracle data dictionary.

```
SELECT * FROM targetdb1."REINVENT".mytable;
```

Because Amazon Redshift defaults to lowercase object names, the following queries also succeed in finding `mytable`.

```
SELECT * FROM targetdb1."REINVENT".MYtablE;
SELECT * FROM targetdb1."REINVENT".MYTABLE;
SELECT * FROM targetdb1."REINVENT".MyTable;
```

The following queries use the quoting mechanism for the object name. They all return an error because they search for an Amazon Redshift table whose name is different from `mytable`.

```
SELECT * FROM targetdb1."REINVENT"."MYTABLE";
SELECT * FROM targetdb1."REINVENT"."MyTable";
SELECT * FROM targetdb1."REINVENT"."MYtablE";
```

Create a table with a mixed-case integration

In the following example, you use double quotes to store the table name in lowercase in the Oracle data dictionary. You create `MyTable` as follows.

```
CREATE TABLE REINVENT."MyTable" (id NUMBER PRIMARY KEY, description VARCHAR2(100));
```

The Oracle database stores this table name as `MyTable` with mixed case. To replicate this table to Amazon Redshift, you must specify the mixed case name in the data filter.

```
aws rds create-integration \
  --integration-name mixedIntegration \
  --data-filter "include: ORCL.REINVENT.MyTable" \
...
```

When you query this table in the replicated database in Amazon Redshift, you must specify the mixed case name `MyTable` by quoting the object name.

```
SELECT * FROM targetdb1."REINVENT"."MyTable";
```

Because Amazon Redshift defaults to lowercase object names, the following queries don't find the object because they are searching for the lowercase name `mytable`.

```
SELECT * FROM targetdb1."REINVENT".MYtablE;
SELECT * FROM targetdb1."REINVENT".MYTABLE;
SELECT * FROM targetdb1."REINVENT".mytable;
```

> **Note**
>
> You can't use regular expressions in the filter value for database name, schema, or table name in RDS for Oracle integrations.

Adding data filters to an integration

You can configure data filtering using the AWS Management Console, the AWS CLI, or the Amazon RDS API.

> **Important**
>
> If you add a filter after you create an integration, Amazon RDS treats it as if it always existed. It removes any data in the target data warehouse that doesn't match the new filtering criteria and resynchronizes all affected tables.

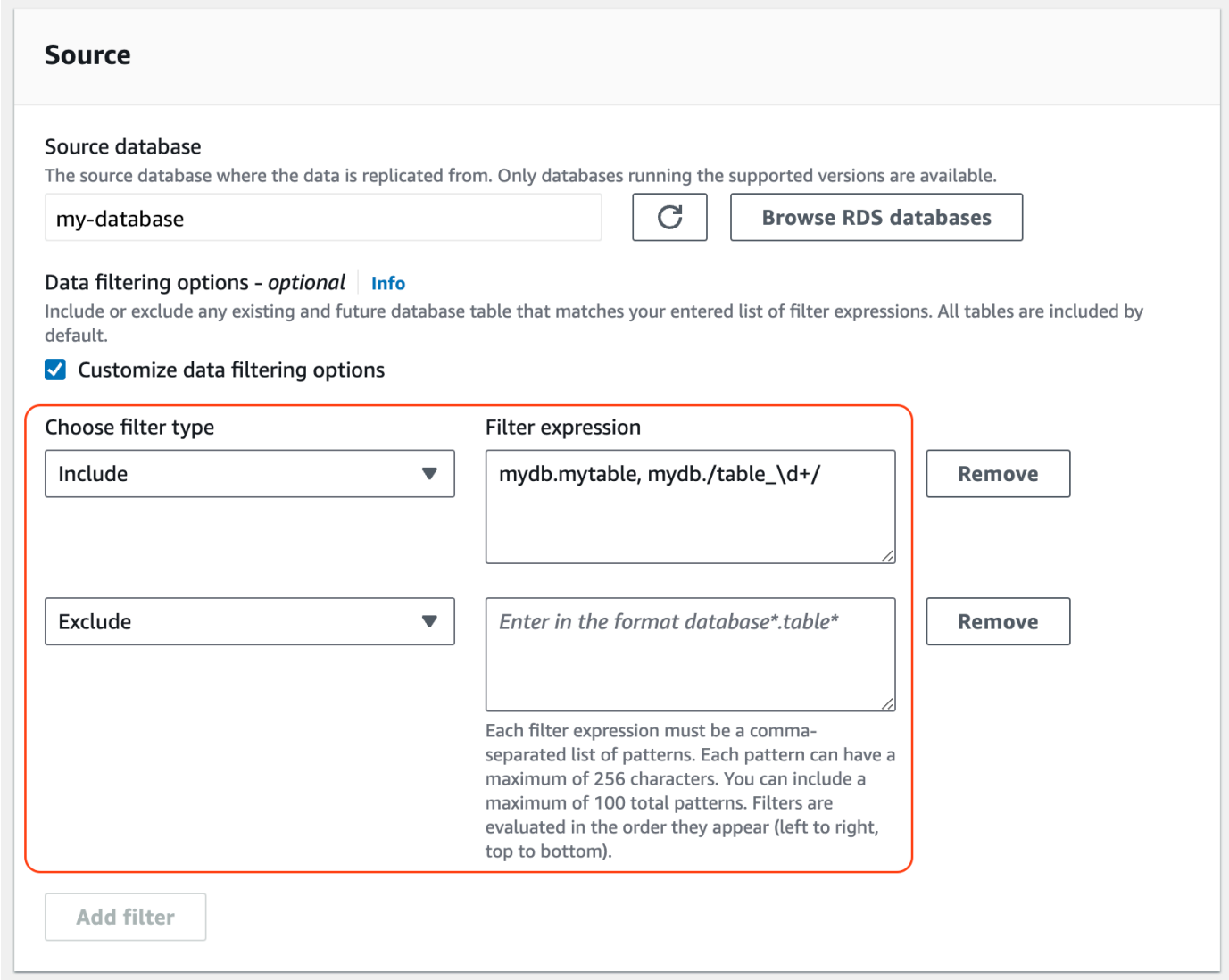
RDS console

To add data filters to a zero-ETL integration

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Zero-ETL integrations**. Select the integration that you want to add data filters to, and then choose **Modify**.

3. Under **Source**, add one or more `Include` and `Exclude` statements.

   The following image shows an example of data filters for a MySQL integration:



4. When you're satisfied with the changes, choose **Continue** and **Save changes**.

AWS CLI

To add data filters to a zero-ETL integration using the AWS CLI, call the [modify-integration] command. In addition to the integration identifier, specify the `--data-filter` parameter with a comma-separated list of `Include` and `Exclude` Maxwell filters.

Example

The following example adds filter patterns to `my-integration`.

For Linux, macOS, or Unix:

```
aws rds modify-integration \
    --integration-identifier my-integration \
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

For Windows:

```
aws rds modify-integration ^
    --integration-identifier my-integration ^
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

RDS API

To modify a zero-ETL integration using the RDS API, call the ModifyIntegration operation. Specify the integration identifier and provide a comma-separated list of filter patterns.

Removing data filters from an integration

When you remove a data filter from an integration, Amazon RDS reevaluates the remaining filters as if the removed filter never existed. It then replicates any previously excluded data that now meets the criteria into the target data warehouse. This triggers a resynchronization of all affected tables.

Adding data to a source RDS database and querying it

To finish creating a zero-ETL integration that replicates data from Amazon RDS into Amazon Redshift, you must create a database in the target destination.

For connections with Amazon Redshift, connect to your Amazon Redshift cluster or workgroup and create a database with a reference to your integration identifier. Then, you can add data to your source RDS database and see it replicated in Amazon Redshift or Amazon SageMaker.

Topics

- Creating a target database
- Adding data to the source database
- Querying your Amazon RDS data in Amazon Redshift

- Data type differences between RDS and Amazon Redshift databases
- DDL operations for RDS for PostgreSQL

Creating a target database

Before you can start replicating data into Amazon Redshift, after you create an integration, you must create a database in your target data warehouse. This database must include a reference to the integration identifier. You can use the Amazon Redshift console or the Query editor v2 to create the database.

For instructions to create a destination database, see Create a destination database in Amazon Redshift.

Adding data to the source database

After you configure your integration, you can populate the source RDS database with data that you want to replicate into your data warehouse.

> **Note**
>
> There are differences between data types in Amazon RDS and the target analytics warehouse. For a table of data type mappings, see the section called "Data type differences".

First, connect to the source database using the MySQL client of your choice. For instructions, see the section called "Connecting to a DB instance running MySQL".

Then, create a table and insert a row of sample data.

> **Important**
>
> Make sure that the table has a primary key. Otherwise, it can't be replicated to the target data warehouse.

RDS for MySQL

The following example uses the MySQL Workbench utility.

```
CREATE DATABASE my_db;

USE my_db;

CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977, 'Supernatural
 fiction');
```

RDS for PostgreSQL

The following example uses the psql PostgreSQL interactive terminal. When connecting to the database, include the database name that you want to replicate.

```
psql -h mydatabase.123456789012.us-east-2.rds.amazonaws.com -p 5432 -U username -
d named_db;

named_db=> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL,
 Author VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

named_db=> INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977,
 'Supernatural fiction');
```

RDS for Oracle

The following example uses SQL*Plus to connect to your RDS for Oracle database.

```
sqlplus 'user_name@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=dns_name)(PORT=port))
(CONNECT_DATA=(SID=database_name)))'

SQL> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

SQL> INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977,
 'Supernatural fiction');
```

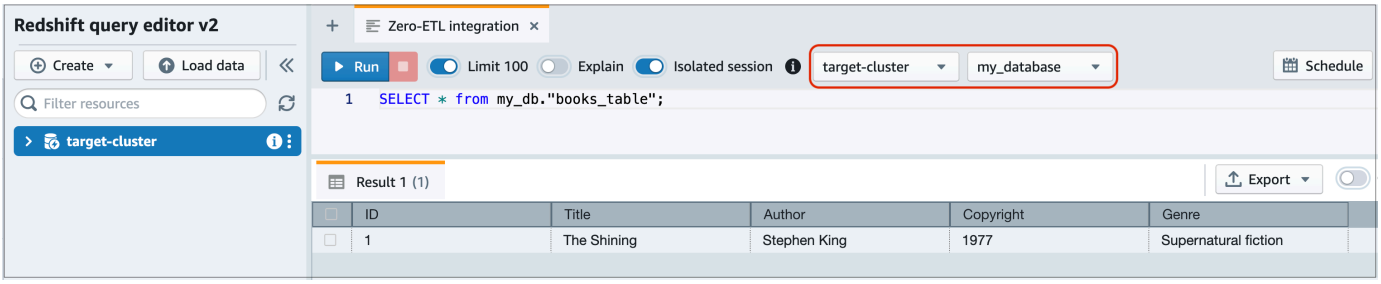
Querying your Amazon RDS data in Amazon Redshift

After you add data to the RDS database, it's replicated into the destination database and is ready to be queried.

To query the replicated data

1. Navigate to the Amazon Redshift console and choose **Query editor v2** from the left navigation pane.
2. Connect to your cluster or workgroup and choose your destination database (which you created from the integration) from the dropdown menu (**destination_database** in this example). For instructions to create a destination database, see Create a destination database in Amazon Redshift.
3. Use a SELECT statement to query your data. In this example, you can run the following command to select all data from the table that you created in the source RDS database:

```
SELECT * from my_db."books_table";
```



- *my_db* is the RDS database schema name.
- *books_table* is the RDS table name.

You can also query the data using the a command line client. For example:

```
destination_database=# select * from my_db."books_table";

 ID |       Title |        Author |   Copyright |                 Genre |  txn_seq |
 txn_id
----+-------------+---------------+-------------+-----------------------+----------
+--------+
```

```
 1 | The Shining |  Stephen King |        1977 |   Supernatural fiction |        2 |
12192
```

> **Note**
>
> For case-sensitivity, use double quotes (" ") for schema, table, and column names. For more information, see enable_case_sensitive_identifier.

Data type differences between RDS and Amazon Redshift databases

The following tables show the mappings of RDS for MySQL, RDS for PostgreSQL, and RDS for Oracle data types to corresponding destination data types. *Amazon RDS currently supports only these data types for zero-ETL integrations.*

If a table in your source database includes an unsupported data type, the table goes out of sync and isn't consumable by the destination target. Streaming from the source to the target continues, but the table with the unsupported data type isn't available. To fix the table and make it available in the target destination, you must manually revert the breaking change and then refresh the integration by running `ALTER DATABASE...INTEGRATION REFRESH`.

> **Note**
>
> You can't refresh zero-ETL integrations with an Amazon SageMaker lakehouse. Instead, delete and try to create the integration again.

Topics

- RDS for MySQL
- RDS for PostgreSQL
- RDS for Oracle

RDS for MySQL

RDS for MySQL data type	Target data type	Description	Limitations
INT	INTEGER	Signed four-byte integer	None
SMALLINT	SMALLINT	Signed two-byte integer	None
TINYINT	SMALLINT	Signed two-byte integer	None
MEDIUMINT	INTEGER	Signed four-byte integer	None
BIGINT	BIGINT	Signed eight-byte integer	None
INT UNSIGNED	BIGINT	Signed eight-byte integer	None
TINYINT UNSIGNED	SMALLINT	Signed two-byte integer	None
MEDIUMINT UNSIGNED	INTEGER	Signed four-byte integer	None
BIGINT UNSIGNED	DECIMAL(20,0)	Exact numeric of selectable precision	None
DECIMAL(p,s) = NUMERIC(p,s)	DECIMAL(p,s)	Exact numeric of selectable precision	Precision greater than 38 and scale greater than 37 not supported

RDS for MySQL data type	Target data type	Description	Limitations
DECIMAL(p,s) UNSIGNED = NUMERIC(p,s) UNSIGNED	DECIMAL(p,s)	Exact numeric of selectable precision	Precision greater than 38 and scale greater than 37 not supported
FLOAT4/REAL	REAL	Single precision floating-point number	None
FLOAT4/REAL UNSIGNED	REAL	Single precision floating-point number	None
DOUBLE/REAL/FLOAT8	DOUBLE PRECISION	Double precision floating-point number	None
DOUBLE/REAL/FLOAT8 UNSIGNED	DOUBLE PRECISION	Double precision floating-point number	None
BIT(n)	VARBYTE(8)	Variable-length binary value	None
BINARY(n)	VARBYTE(n)	Variable-length binary value	None
VARBINARY(n)	VARBYTE(n)	Variable-length binary value	None
CHAR(n)	VARCHAR(n)	Variable-length string value	None
VARCHAR(n)	VARCHAR(n)	Variable-length string value	None

RDS for MySQL data type	Target data type	Description	Limitations
TEXT	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
TINYTEXT	VARCHAR(255)	Variable-length string value up to 255 characters	None
MEDIUMTEXT	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
LONGTEXT	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
ENUM	VARCHAR(1020)	Variable-length string value up to 1,020 characters	None
SET	VARCHAR(1020)	Variable-length string value up to 1,020 characters	None
DATE	DATE	Calendar date (year, month, day)	None

RDS for MySQL data type	Target data type	Description	Limitations
DATETIME	TIMESTAMP	Date and time (without time zone)	None
TIMESTAMP(p)	TIMESTAMP	Date and time (without time zone)	None
TIME	VARCHAR(18)	Variable-length string value up to 18 characters	None
YEAR	VARCHAR(4)	Variable-length string value up to 4 characters	None
JSON	SUPER	Semistructured data or documents as values	None

RDS for PostgreSQL

Zero-ETL integrations for RDS for PostgreSQL don't support custom data types or data types created by extensions.

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
array	SUPER	Semistructured data or documents as values	None

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
bigint	BIGINT	Signed eight-byte integer	None
bigserial	BIGINT	Signed eight-byte integer	None
bit varying(n)	VARBYTE(n)	Variable-length binary value up to 16,777,216 bytes	None
bit(n)	VARBYTE(n)	Variable-length binary value up to 16,777,216 bytes	None
bit, bit varying	VARBYTE(16777216)	Variable-length binary value up to 16,777,216 bytes	None
boolean	BOOLEAN	Logical boolean (true/false)	None
bytea	VARBYTE(16777216)	Variable-length binary value up to 16,777,216 bytes	None
char(n)	CHAR(n)	Fixed-length character string value up to 65,535 bytes	None

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
char varying(n)	VARCHAR(65535)	Variable-length character string value up to 65,535 characters	None
cid	BIGINT	Signed eight-byte integer	None
cidr	VARCHAR(19)	Variable-length string value up to 19 characters	None
date	DATE	Calendar date (year, month, day)	Values greater than 294,276 A.D. not supported
double precision	DOUBLE PRECISION	Double precision floating-point numbers	Subnormal values not fully supported
gtsvector	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
inet	VARCHAR(19)	Variable-length string value up to 19 characters	None
integer	INTEGER	Signed four-byte integer	None

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
int2vector	SUPER	Semistructured data or documents as values.	None
interval	INTERVAL	Duration of time	Only INTERVAL types that specify either a year to month or a day to second qualifier are supported.
json	SUPER	Semistructured data or documents as values	None
jsonb	SUPER	Semistructured data or documents as values	None
jsonpath	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
macaddr	VARCHAR(17)	Variable-length string value up to 17 characters	None
macaddr8	VARCHAR(23)	Variable-length string value up to 23 characters	None

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
money	DECIMAL(20,3)	Currency amount	None
name	VARCHAR(64)	Variable-length string value up to 64 characters	None
numeric(p,s)	DECIMAL(p,s)	User-defined fixed precision value	• NaN values not supported • Precision and scale must be explicitly defined and not greater than 38 (precision) and 37 (scale) • Negative scale not supported
oid	BIGINT	Signed eight-byte integer	None
oidvector	SUPER	Semistructured data or documents as values.	None
pg_brin_bloom_summary	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
pg_dependencies	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
pg_lsn	VARCHAR(17)	Variable-length string value up to 17 characters	None
pg_mcv_list	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
pg_ndistinct	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
pg_node_tree	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
pg_snapshot	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
real	REAL	Single precision floating-point number	Subnormal values not fully supported

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
refcursor	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
smallint	SMALLINT	Signed two-byte integer	None
smallserial	SMALLINT	Signed two-byte integer	None
serial	INTEGER	Signed four-byte integer	None
text	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
tid	VARCHAR(23)	Variable-length string value up to 23 characters	None
time [(p)] without time zone	VARCHAR(19)	Variable-length string value up to 19 characters	`Infinity` and `-Infinity` values not supported
time [(p)] with time zone	VARCHAR(22)	Variable-length string value up to 22 characters	`Infinity` and `-Infinity` values not supported

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
timestamp [(p)] without time zone	TIMESTAMP	Date and time (without time zone)	• `Infinity` and `-Infinity` values not supported • Values greater than `9999-12-31` not supported • B.C. values not supported
timestamp [(p)] with time zone	TIMESTAMPTZ	Date and time (with time zone)	• `Infinity` and `-Infinity` values not supported • Values greater than `9999-12-31` not supported • B.C. values not supported
tsquery	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
tsvector	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None

RDS for PostgreSQL data type	Amazon Redshift data type	Description	Limitations
txid_snapshot	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None
uuid	VARCHAR(36)	Variable-length 36 character string	None
xid	BIGINT	Signed eight-byte integer	None
xid8	DECIMAL(20, 0)	Fixed precision decimal	None
xml	VARCHAR(65535)	Variable-length string value up to 65,535 characters	None

RDS for Oracle

Unsupported data types

The following RDS for Oracle data types are not supported by Amazon Redshift:

- `ANYDATA`
- `BFILE`
- `REF`
- `ROWID`
- `UROWID`
- `VARRAY`
- `SDO_GEOMETRY`
- User-defined data types

Data type differences

The following table shows the data type differences that affect a zero-ETL integration when RDS for Oracle is the source and Amazon Redshift is the target.

RDS for Oracle data type	Amazon Redshift data type
BINARY_FLOAT	FLOAT4
BINARY_DOUBLE	FLOAT8
BINARY	VARCHAR (Length)
FLOAT (P)	If precision is =< 24, then FLOAT4. If precision is > 24, then FLOAT8.
NUMBER (P,S)	If scale is => 0 and =< 37, then NUMERIC (p,s). If scale is => 38 and =< 127, then VARCHAR (Length). If scale is 0: • And precision is less than or equal to 2, then INT2. • And precision is greater than 2 and less than or equal to 4, then INT2. • And precision is greater than 4 and less than or equal to 9, then INT4. • And precision is greater than 9, then NUMERIC (p,s). • And precision is greater than or equal to scale, then NUMERIC (p,s). If scale is less than 0, then INT8.

RDS for Oracle data type	Amazon Redshift data type
DATE	If the scale is => 0 and =< 6, depending on the Redshift target column type, then one of the following: • TIMESTAMP (s) • TIMESTAMPTZ (s) — If source timestamp contains a zone offset (such as in SQL Server or Oracle) it converts to UTC on insert/up date. If it does not contain an offset, then time is considered in UTC already. If the scale is => 7 and =< 9, then VARCHAR (37).
INTERVAL_YEAR TO MONTH	If the length is 1–65,535, then VARCHAR (length in bytes). If the length is 65,536–2,147,483,647, then VARCHAR (65535).
INTERVAL_DAY TO SECOND	If the length is 1–65,535, then VARCHAR (length in bytes). If the length is 65,536–2,147,483,647, then VARCHAR (65535).

RDS for Oracle data type	Amazon Redshift data type
TIMESTAMP	If the scale is => 0 and =< 6, depending on the Redshift target column type, then one of the following: • TIMESTAMP (s) • TIMESTAMPTZ (s) — If source timestamp contains a zone offset (such as in SQL Server or Oracle) it converts to UTC on insert/up date. If it does not contain an offset, then time is considered in UTC already. If the scale is => 7 and =< 9, then VARCHAR (37).
TIMESTAMP WITH TIME ZONE	If the length is 1–65,535, then VARCHAR (length in bytes). If the length is 65,536–2,147,483,647, then VARCHAR (65535).
TIMESTAMP WITH LOCAL TIME ZONE	If the length is 1–65,535, then VARCHAR (length in bytes). If the length is 65,536–2,147,483,647, then VARCHAR (65535).
CHAR	If the length is 1–65,535, then VARCHAR (length in bytes). If the length is 65,536–2,147,483,647, then VARCHAR (65535).

RDS for Oracle data type	Amazon Redshift data type
VARCHAR2	When the length is greater than 4,000 bytes, then VARCHAR (maximum LOB size). The maximum LOB size cannot exceed 63 KB. Amazon Redshift does not support VARCHARs larger than 64 KB. When the length is 4,000 bytes or less, then VARCHAR (length in bytes).
NCHAR	If the length is 1–65,535, then NVARCHAR (length in bytes). If the length is 65,536–2,147,483,647, then NVARCHAR (65535).
NVARCHAR2	When the length is greater than 4,000 bytes, then NVARCHAR (maximum LOB size). The maximum LOB size cannot exceed 63 KB. Amazon Redshift does not support VARCHARs larger than 64 KB. When the length is 4,000 bytes or less, then NVARCHAR (length in bytes).
RAW	VARCHAR (Length)
REAL	FLOAT8
BLOB	VARCHAR (maximum LOB size *2) The maximum LOB size cannot exceed 31 KB. Amazon Redshift does not support VARCHARs larger than 64 KB.

RDS for Oracle data type	Amazon Redshift data type
CLOB	VARCHAR (maximum LOB size) The maximum LOB size cannot exceed 63 KB. Amazon Redshift does not support VARCHARs larger than 64 KB.
NCLOB	NVARCHAR (maximum LOB size) The maximum LOB size cannot exceed 63 KB. Amazon Redshift does not support VARCHARs larger than 64 KB.
LONG	VARCHAR (maximum LOB size) The maximum LOB size cannot exceed 63 KB. Amazon Redshift does not support VARCHARs larger than 64 KB.
LONG RAW	VARCHAR (maximum LOB size *2) The maximum LOB size cannot exceed 31 KB. Amazon Redshift does not support VARCHARs larger than 64 KB.
XMLTYPE	VARCHAR (maximum LOB size) The maximum LOB size cannot exceed 63 KB. Amazon Redshift does not support VARCHARs larger than 64 KB.

DDL operations for RDS for PostgreSQL

Amazon Redshift is derived from PostgreSQL, so it shares several features with RDS for PostgreSQL due to their common PostgreSQL architecture. Zero-ETL integrations leverage these similarities to streamline data replication from RDS for PostgreSQL to Amazon Redshift, mapping databases by name and utilizing the shared database, schema, and table structure.

Consider the following points when managing RDS for PostgreSQL zero-ETL integrations:

- Isolation is managed at the database level.
- Replication occurs at the database level.
- RDS for PostgreSQL databases are mapped to Amazon Redshift databases by name, with data flowing to the corresponding renamed Redshift database if the original is renamed.

Despite their similarities, Amazon Redshift and RDS for PostgreSQL have important differences. The following sections outline Amazon Redshift system responses for common DDL operations.

Topics

- Database operations
- Schema operations
- Table operations

Database operations

The following table shows the system responses for database DDL operations.

DDL operation	Redshift system response
`CREATE DATABASE`	No operation
`DROP DATABASE`	Amazon Redshift drops all the data in the target Redshift database.
`RENAME DATABASE`	Amazon Redshift drops all the data in the original target database and resynchro nize the data in the new target database. If the new database doesn't exist, you must manually create it. For instructions, see Create a destination database in Amazon Redshift.

Schema operations

The following table shows the system responses for schema DDL operations.

DDL operation	Redshift system response
CREATE SCHEMA	No operation
DROP SCHEMA	Amazon Redshift drops the original schema.
RENAME SCHEMA	Amazon Redshift drops the original schema then resynchronizes the data in the new schema.

Table operations

The following table shows the system responses for table DDL operations.

DDL operation	Redshift system response
CREATE TABLE	Amazon Redshift creates the table. Some operations cause table creation to fail, such as creating a table without a primary key or performing declarative partitioning. For more information, see the section called "Limitations" and the section called "Troubles hooting zero-ETL integrations".
DROP TABLE	Amazon Redshift drops the table.
TRUNCATE TABLE	Amazon Redshift truncates the table.
ALTER TABLE (RENAME...)	Amazon Redshift renames the table or column.
ALTER TABLE (SET SCHEMA)	Amazon Redshift drops the table in the original schema and resynchronizes the table in the new schema.
ALTER TABLE (ADD PRIMARY KEY)	Amazon Redshift adds a primary key and resynchronizes the table.

DDL operation	Redshift system response
ALTER TABLE (ADD COLUMN)	Amazon Redshift adds a column to the table.
ALTER TABLE (DROP COLUMN)	Amazon Redshift drops the column if it's not a primary key column. Otherwise, it resynchronizes the table.
ALTER TABLE (SET LOGGED/UNLOGGED)	If you change the table to logged, Amazon Redshift resynchronizes the table. If you change the table to unlogged, Amazon Redshift drops the table.

Viewing and monitoring Amazon RDS zero-ETL integrations

You can view the details of an Amazon RDS zero-ETL integration to see its configuration information and current status. You can also monitor the status of your integration by querying specific system views in Amazon Redshift. In addition, Amazon Redshift publishes certain integration-related metrics to Amazon CloudWatch, which you can view within the Amazon Redshift console.

Topics

- Viewing integrations
- Monitoring integrations using system tables for Amazon Redshift
- Monitoring integrations with Amazon EventBridge for Amazon Redshift

Viewing integrations

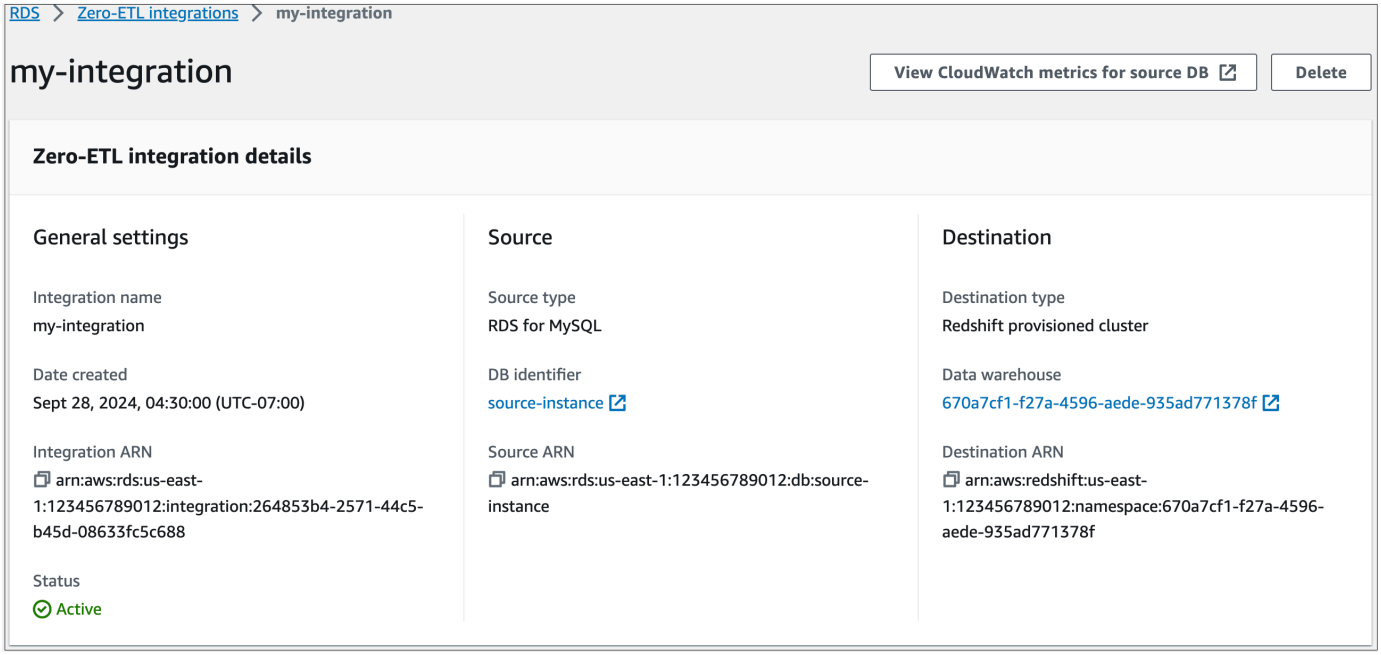
You can view Amazon RDS zero-ETL integrations using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To view the details of a zero-ETL integration

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. From the left navigation pane, choose **Zero-ETL integrations**.
3. Select an integration to view more details about it, such as its source database and target data warehouse.



An integration can have the following statuses:

- `Creating` – The integration is being created.
- `Active` – The integration is sending transactional data to the target data warehouse.
- `Syncing` – The integration has encountered a recoverable error and is reseeding data. Affected tables aren't available for querying until they finish resyncing.
- `Needs attention` – The integration encountered an event or error that requires manual intervention to resolve it. To fix the issue, follow the instructions in the error message on the integration details page.
- `Failed` – The integration encountered an unrecoverable event or error that can't be fixed. You must delete and recreate the integration.
- `Deleting` – The integration is being deleted.

AWS CLI

To view all zero-ETL integrations in the current account using the AWS CLI, use the describe-integrations command and specify the `--integration-identifier` option.

Example

For Linux, macOS, or Unix:

```
aws rds describe-integrations \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

For Windows:

```
aws rds describe-integrations ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

To view zero-ETL integration using the Amazon RDS API, use the `DescribeIntegrations` operation with the `IntegrationIdentifier` parameter.

Monitoring integrations using system tables for Amazon Redshift

Amazon Redshift has system tables and views that contain information about how the system is functioning. You can query these system tables and views the same way that you would query any other database table. For more information about system tables and views in Amazon Redshift, see System tables and views reference in the *Amazon Redshift Database Developer Guide*.

You can query the following system views and tables to get information about your zero-ETL integrations:

- SVV_INTEGRATION – Provides configuration details for your integrations.
- SVV_INTEGRATION_TABLE_STATE – Describes the state of each table within an integration.
- SYS_INTEGRATION_TABLE_STATE_CHANGE – Displays table state change logs for an integration.
- SYS_INTEGRATION_ACTIVITY – Provides information about completed integration runs.

All integration-related Amazon CloudWatch metrics originate from Amazon Redshift. For more information, see Metrics for zero-ETL integrations in the *Amazon Redshift Management Guide*. Currently, Amazon RDS doesn't publish any integration metrics to CloudWatch.

Monitoring integrations with Amazon EventBridge for Amazon Redshift

Amazon Redshift send integration-related events to Amazon EventBridge. For a list of events and their corresponding event IDs, see Zero-ETL integration event notifications with Amazon EventBridge in the *Amazon Redshift Management Guide*.

Modifying Amazon RDS zero-ETL integrations

You can modify only the name, description, and data filtering options for a zero-ETL integration in a supported data warehouse. You can't modify the AWS KMS key used to encrypt the integration, or the source or target databases.

If you add a data filter to an existing integration, Amazon RDS reevaluates the filter as if it always existed. It removes any data that is currently in the target data warehouse that doesn't match the new filtering criteria. If you *remove* a data filter from an integration, it replicates any data that previously didn't match the filtering criteria (but now does) into the target data warehouse. For more information, see the section called “Data filtering for zero-ETL integrations”.

You can modify a zero-ETL integration using the AWS Management Console, the AWS CLI, or the Amazon RDS API.

RDS console

To modify a zero-ETL integration

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Zero-ETL integrations**, and then choose the integration that you want to modify.
3. Choose **Modify** and make modifications to any available settings.
4. When all the changes are as you want them, choose **Modify**.

AWS CLI

To modify a zero-ETL integration using the AWS CLI, call the modify-integration command. Along with the `--integration-identifier`, specify any of the following options:

- `--integration-name` – Specify a new name for the integration.

- `--description` – Specify a new description for the integration.
- `--data-filter` – Specify data filtering options for the integration. For more information, see the section called "Data filtering for zero-ETL integrations".

Example

The following request modifies an existing integration.

For Linux, macOS, or Unix:

```
aws rds modify-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 \
    --integration-name my-renamed-integration
```

For Windows:

```
aws rds modify-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 ^
    --integration-name my-renamed-integration
```

RDS API

To modify a zero-ETL integration using the RDS API, call the ModifyIntegration operation. Specify the integration identifier, and the parameters that you want to modify.

Deleting Amazon RDS zero-ETL integrations

When you delete a zero-ETL integration, Amazon RDS removes it from the source database. Your transactional data isn't deleted from Amazon RDS or the analytics destination, but Amazon RDS doesn't send new data to Amazon Redshift or Amazon SageMaker.

You can only delete an integration when it has a status of `Active`, `Failed`, `Syncing`, or `Needs attention`.

You can delete zero-ETL integrations using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To delete a zero-ETL integration

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. From the left navigation pane, choose **Zero-ETL integrations**.
3. Select the zero-ETL integration that you want to delete.
4. Choose **Actions**, **Delete**, and confirm deletion.

AWS CLI

To delete a zero-ETL integration, use the delete-integration command and specify the `--integration-identifier` option.

Example

For Linux, macOS, or Unix:

```
aws rds delete-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

For Windows:

```
aws rds delete-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

To delete a zero-ETL integration using the Amazon RDS API, use the `DeleteIntegration` operation with the `IntegrationIdentifier` parameter.

Troubleshooting Amazon RDS zero-ETL integrations

You can check the state of a zero-ETL integration by querying the SVV_INTEGRATION system table in the analytics destination. If the `state` column has a value of `ErrorState`, it means something's wrong. For more information, see the section called “Monitoring using system tables”.

Use the following information to troubleshoot common issues with Amazon RDS zero-ETL integrations.

> **Important**
>
> Resync and refresh operations are not available for zero-ETL integrations with an Amazon SageMaker AI lakehouse. If there are issues with an integration, you must delete the integration and create a new integration. You can't refresh or resync an existing integration.

Topics

- I can't create a zero-ETL integration
- My integration is stuck in a state of Syncing
- My tables aren't replicating to Amazon Redshift
- One or more of my Amazon Redshift tables requires a resync
- Integration failed issues for Amazon SageMaker AI lakehouse zero-ETL integrations

I can't create a zero-ETL integration

If you can't create a zero-ETL integration, make sure that the following are correct for your source database:

- Your source database must be running a supported DB engine version. For a list of supported versions, see the section called “Zero-ETL integrations ”.
- You correctly configured DB parameters. If the required parameters are set incorrectly or not associated with the database, creation fails. See the section called “Step 1: Create a custom DB parameter group”.

In addition, make sure the following are correct for your target data warehouse:

- Case sensitivity is enabled. See Turn on case sensitivity for your data warehouse.
- You added the correct authorized principal and integration source. See Configure authorization for your Amazon Redshift data warehouse.
- The data warehouse is encrypted (if it's a provisioned cluster). See Amazon Redshift database encryption.

My integration is stuck in a state of `Syncing`

Your integration might consistently show a status of `Syncing` if you change the value of one of the required DB parameters.

To fix this issue, check the values of the parameters in the parameter group associated with the source database, and make sure that they match the required values. For more information, see the section called “Step 1: Create a custom DB parameter group”.

If you modify any parameters, make sure to reboot the database to apply the changes.

My tables aren't replicating to Amazon Redshift

If you don't see one or more tables reflected in Amazon Redshift, you can run the following command to resynchronize them:

```
ALTER DATABASE dbname INTEGRATION REFRESH TABLES table1, table2;
```

For more information, see ALTER DATABASE in the Amazon Redshift SQL reference.

Your data might not be replicating because one or more of your source tables doesn't have a primary key. The monitoring dashboard in Amazon Redshift displays the status of these tables as `Failed`, and the status of the overall zero-ETL integration changes to `Needs attention`. To resolve this issue, you can identify an existing key in your table that can become a primary key, or you can add a synthetic primary key. For detailed solutions, see Handle tables without primary keys while creating Amazon Aurora MySQL or Amazon RDS for MySQL zero-ETL integrations with Amazon Redshift.

One or more of my Amazon Redshift tables requires a resync

Running certain commands on your source database might require your tables to be resynchronized. In these cases, the SVV_INTEGRATION_TABLE_STATE system view shows a `table_state` of `ResyncRequired`, which means that the integration must completely reload data for that specific table from MySQL to Amazon Redshift.

When the table starts to resynchronize, it enters a state of `Syncing`. You don't need to take any manual action to resynchronize a table. While table data is resynchronizing, you can't access it in Amazon Redshift.

The following are some example operations that can put a table into a `ResyncRequired` state, and possible alternatives to consider.

Operation	Example	Alternative
Adding a column into a specific position	`ALTER TABLE table_name ADD COLUMN column_name INTEGER NOT NULL first;`	Amazon Redshift doesn't support adding columns into specific positions using `first` or `after` keywords. If the order of columns in the target table isn't critical, add the column to the end of the table using a simpler command: `ALTER TABLE table_name ADD COLUMN column_name column_type ;`
Adding a timestamp	`ALTER TABLE table_name`	The CURRENT_T

Operation	Example	Alternative
column with the default CURRENT_TIMESTAMP	`ADD COLUMN column_name TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP;`	IMESTAMP value for existing table rows is calculated by RDS for MySQL and can't be simulated in Amazon Redshift without full table data resynchronization. If possible, switch the default value to a literal constant like 2023-01-01 00:00:15 to avoid latency in table availability.

Operation	Example	Alternative
Performing multiple column operations within a single command	`ALTER TABLE table_name ADD COLUMN column_1, RENAME COLUMN column_2 TO column_3;`	Consider splitting the command into two separate operations, ADD and RENAME, which won't require resynchronization.

Integration failed issues for Amazon SageMaker AI lakehouse zero-ETL integrations

If you encounter issues with an existing zero-ETL integration with an Amazon SageMaker AI lakehouse, the only resolution is to delete the integration and create a new one. Unlike other AWS services, zero-ETL integrations do not support refresh or resync operations.

To resolve integration issues:

1. Delete the problematic zero-ETL integration using the console, CLI, or API.
2. Verify that the source database and target data warehouse configurations are correct.
3. Create a new zero-ETL integration with the same or updated configuration.

This process will result in a complete re-initialization of the data pipeline, which may take time depending on the size of your source database.

Amazon RDS for Db2

Amazon RDS supports DB instances that run the following editions of IBM Db2:

- Db2 Standard Edition
- Db2 Advanced Edition

Amazon RDS supports DB instances that run the following versions of Db2:

- Db2 11.5

For more information about minor version support, see Db2 on Amazon RDS versions.

Before creating a DB instance, complete the steps in the Setting up your Amazon RDS environment section of this user guide. When you create a DB instance using your master user, the user gets `DBADM` authority, with some limitations. Use this user for administrative tasks such as creating additional database accounts. You can't use `SYSADM`, `SYSCTRL`, `SYSMAINT` instance-level authority, or `SECADM` database-level authority.

You can create the following:

- DB instances
- DB snapshots
- Point-in-time restores
- Automated storage backups
- Manual storage backups

You can use DB instances running Db2 inside a virtual private cloud (VPC). You can also add features to your Amazon RDS for Db2 DB instance by enabling various options. Amazon RDS supports Multi-AZ deployments for RDS for Db2 as a high availability, failover solution.

Important

To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. It also restricts access to certain system procedures and tables that need elevated

privileges. You can access your database using standard SQL clients such as IBM Db2 CLP. However, you can't access the host directly by using Telnet or Secure Shell (SSH).

Topics

- Overview of Db2 on Amazon RDS
- Prerequisites for creating an Amazon RDS for Db2 DB instance
- Multiple databases on an Amazon RDS for Db2 DB instance
- Connecting to your Db2 DB instance
- Securing Amazon RDS for Db2 DB instance connections
- Administering your Amazon RDS for Db2 DB instance
- Integrating an Amazon RDS for Db2 DB instance with Amazon S3
- Migrating data to Amazon RDS for Db2
- Amazon RDS for Db2 federation
- Working with replicas for Amazon RDS for Db2
- Options for Amazon RDS for Db2 DB instances
- External stored procedures for Amazon RDS for Db2
- Known issues and limitations for Amazon RDS for Db2
- Amazon RDS for Db2 stored procedure reference
- Amazon RDS for Db2 user-defined function reference
- Troubleshooting for Amazon RDS for Db2

Overview of Db2 on Amazon RDS

You can read the following sections to get an overview of Db2 on Amazon RDS.

Topics

- Amazon RDS for Db2 features
- Db2 on Amazon RDS versions
- Amazon RDS for Db2 licensing options
- Amazon RDS for Db2 instance classes

- Amazon RDS for Db2 default roles
- Amazon RDS for Db2 parameters
- EBCDIC collation for Db2 databases on Amazon RDS
- Local time zone for Amazon RDS for Db2 DB instances

Amazon RDS for Db2 features

Amazon RDS for Db2 supports most of the features and capabilities of the IBM Db2 database. Some features might have limited support or restricted privileges. For more information about the Db2 database features for specific Db2 versions, see the IBM Db2 documentation.

You can filter new Amazon RDS features on the What's New with Database? page. For **Products**, choose **Amazon RDS**. Then, you can search by using keywords such as **`Db2 2023`**.

> **Note**
>
> The following lists aren't exhaustive.

Topics

- Supported features in RDS for Db2
- Unsupported features in RDS for Db2

Supported features in RDS for Db2

RDS for Db2 supports features that include features that are native to IBM Db2 and features that are core to Amazon RDS.

Features native to IBM Db2

RDS for Db2 supports the following Db2 database features:

- Creation of a standard database that uses a customer-defined code set, collation, page size, and territory. Use the Amazon RDS rdsadmin.create_database stored procedure.
- Addition, deletion, or modification of local users and groups. Use the Amazon RDS stored procedures for Stored procedures for granting and revoking privileges for RDS for Db2.

- Creation of roles with the Amazon RDS rdsadmin.create_role stored procedure.
- Support for standard row-organized tables.
- Support for analytic workload for column-organized tables.
- Ability to define Db2-compatibility features such as Oracle and MySQL.
- Support for Java-based external stored procedures.
- Support for data encryption in transit by using SSL/TLS.
- Monitoring the status of a database (`ALIVE`, `DOWN`, `STORAGE_FULL`, `UNKNOWN`, and `STANDBY_CONNECTABLE`).
- Restoration of a customer-provided offline or online Linux (LE) database. Use Amazon RDS stored procedures for Stored procedures for databases for RDS for Db2.
- Application of customer-provided Db2 archive logs to keep the database synchronized with self-managed Db2 databases. Use Amazon RDS stored procedures for Stored procedures for databases for RDS for Db2.
- Support for Db2 instance-level and database-level auditing.
- Support for homogeneous federation.
- Ability to load a table from data files in Amazon Simple Storage Service (Amazon S3).
- Authorizations granted to users, groups or roles, such as `CONNECT`, `SYSMON`, `ACCESSCTRL`, `DATAACCESS`, `SQLADM`, `WLMADM`, `EXPLAIN`, `LOAD`, or `IMPLICIT_SCHEMA`.
- Creation of multiple databases.

> **Note**
>
> An RDS for Db2 DB instance can contain up to 50 databases. For more information, see the section called “Multiple Db2 databases”.

Features core to Amazon RDS

RDS for Db2 supports the following core Amazon RDS features:

- Custom parameter groups to assign to DB instances
- Creation, modification, and deletion of DB instances
- Restoration of a self-managed Db2 offline or online Linux (LE) database backup

> **Note**
>
> To be able to restore your backup, don't provide a name for your database when you create a DB instance. For more information, see Creating an Amazon RDS DB instance.

- Support of gp3, io2, and io1 storage types
- Use of AWS Managed Microsoft AD for Kerberos authentication, and LDAP group authorization for RDS for Db2
- Modification of security groups, ports, instance types, storage, backup retention periods, and other settings for existing Db2 instances
- Deletion protection for DB instances
- Cross-Region point-in-time recovery (PITR), including for encrypted backups
- Use of AWS Key Management Service (AWS KMS) for storage encryption and encryption at rest
- Multi-AZ DB instances with one standby for high availability
- Reboots of DB instances
- Updates to master passwords
- Restoration of DB instances to a specific time
- Backup and restoration of DB instances by using storage-level backups
- Start and stop of DB instances
- Maintenance of DB instances
- Same-Region and cross-Region standby and read replicas

Unsupported features in RDS for Db2

RDS for Db2 doesn't support the following Db2 database features:

- `SYSADM`, `SECADM`, and `SYSMAINT` access for the master user.
- External stored procedures written in C, C++, or Cobol.
- Multiple Db2 DB instances on a single host.
- External GSS-API plugins for authentication.
- External third-party plugins to back up or restore Db2 databases.

- Multi-node massively parallel processing (MPP), such as IBM Db2 Warehouse.
- IBM Db2 pureScale.
- Manual setup of High Availability Disaster Recovery (HADR) for RDS for Db2.

> **Note**
>
> Amazon RDS supports and manages HADR for RDS for Db2 through replicas. For more information, see Working with replicas for Amazon RDS for Db2.
> RDS for Db2 supports Multi-AZ deployments, cross-Region automated backups, and replication. For more information, see Multi-AZ DB instance deployments for Amazon RDS and Replicating automated backups to another AWS Region.

- Native database encryption.
- Heterogeneous federation to Informix, Sybase, and Teradata. For more information, see the section called "Federation".
- Creation of non-fenced routines and migration of existing non-fenced routines by backing up and restoring data. For more information, see Non-fenced routines.
- Creation of new non-automatic storage tablespaces. For more information, see Non-automatic storage tablespaces during migration.
- External tables.

Db2 on Amazon RDS versions

For Db2, version numbers take the form of *major.minor.build.revision*, for example, 11.5.9.0.sb00000000.r1. Our version implementation matches that of Db2.

major

The major version number is both the integer and the first fractional part of the version number, for example, 11.5. A version change is considered major if the major version number changes—for example, going from version 11.5 to 12.1.

minor

The minor version number is both the third and fourth parts of the version number, for example, 9.0 in 11.5.9.0. The third part indicates the Db2 modpack, for example, 9 in 9.0. The fourth part indicates the Db2 fixpack, for example, 0 in 9.0. A version change is considered

minor if either the Db2 modpack or the Db2 fixpack changes—for example, going from version 11.5.9.0 to 11.5.9.1, or from 11.5.9.0 to 11.5.10.0, with exceptions to provide catalog table updates. (Amazon RDS takes care of these exceptions.)

build

The build number is the fifth part of the version number, for example, sb00000000 in 11.5.9.0.sb00000000. A build number where the number portion is all zeroes indicates a standard build. A build number where the number portion isn't all zeroes indicates a special build. A build number changes if there is a security fix or special build of an existing Db2 version. A build number change also indicates that Amazon RDS automatically applied a new minor version.

revision

The revision number is the sixth part of the version number, for example, r1 in 11.5.9.0.sb00000000.r1. A revision is an Amazon RDS revision to an existing Db2 release. A revision number change indicates that Amazon RDS automatically applied a new minor version.

Topics

- Upgrade management for Amazon RDS Db2 instances
- Supported Db2 major versions on Amazon RDS

Upgrade management for Amazon RDS Db2 instances

To see the current list of supported Db2 minor versions on RDS, use one of the following commands:

```
aws rds describe-db-engine-versions --engine db2-se
aws rds describe-db-engine-versions --engine db2-ae
```

Amazon RDS also supports upgrade rollout policy to manage automatic minor version upgrades across multiple database resources and AWS accounts. For more information, see Using AWS Organizations upgrade rollout policy for automatic minor version upgrades.

You can specify any currently supported Db2 version when creating a new DB instance. You can specify the major version (such as Db2 11.5) and any supported minor version for the specified major version. If no version is specified, Amazon RDS defaults to a supported version, typically

the most recent version. If a major version is specified but a minor version is not, Amazon RDS defaults to a recent release of the major version that you have specified. To see a list of supported versions, as well as defaults for newly created DB instances, use the describe-db-engine-versions AWS Command Line Interface (AWS CLI) command.

For example, to list the supported engine versions for Amazon RDS for Db2, run the following AWS CLI command. Replace *region* with your AWS Region.

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
    --filters Name=engine,Values=db2-ae,db2-se \
    --query "DBEngineVersions[].{Engine:Engine, EngineVersion:EngineVersion,
 DBParameterGroupFamily:DBParameterGroupFamily}" \
    --region region
```

For Windows:

```
aws rds describe-db-engine-versions ^
    --filters Name=engine,Values=db2-ae,db2-se ^
    --query "DBEngineVersions[].{Engine:Engine, EngineVersion:EngineVersion,
 DBParameterGroupFamily:DBParameterGroupFamily}" ^
    --region region
```

This command produces output similar to the following example:

```
[
    {
    "Engine": "db2-ae",
    "EngineVersion": "11.5.9.0.sb00000000.r1",
    "DBParameterGroupFamily": "db2-ae-11.5"
    },
    {
    "Engine": "db2-se",
    "EngineVersion": "11.5.9.0.sb00000000.r1",
    "DBParameterGroupFamily": "db2-se-11.5"
    }
]
```

The default Db2 version might vary by AWS Region. To create a DB instance with a specific minor version, specify the minor version during DB instance creation. You can determine the default

version for an AWS Region for db2-ae and db2-se database engines by running the `describe-db-engine-versions` command. The following example returns the default version for db2-ae in US East (N. Virginia).

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
    --default-only --engine db2-ae \
    --query "DBEngineVersions[].{Engine:Engine, EngineVersion:EngineVersion,
 DBParameterGroupFamily:DBParameterGroupFamily}" \
    --region us-east-1
```

For Windows:

```
aws rds describe-db-engine-versions ^
    --default-only --engine db2-ae ^
    --query "DBEngineVersions[].{Engine:Engine, EngineVersion:EngineVersion,
 DBParameterGroupFamily:DBParameterGroupFamily}" ^
    --region us-east-1
```

This command produces output similar to the following example:

```
[
    {
    "Engine": "db2-ae",
    "EngineVersion": "11.5.9.0.sb00000000.r1",
    "DBParameterGroupFamily": "db2-ae-11.5"
    }
]
```

With Amazon RDS, you control when to upgrade your Db2 instance to a new major version supported by Amazon RDS. You can maintain compatibility with specific Db2 versions, test new versions with your application before deploying in production, and perform major version upgrades at times that best fit your schedule.

When automatic minor version upgrade is enabled, Amazon RDS automatically upgrades your DB instances to new Db2 minor versions as they are supported by Amazon RDS. This patching occurs during your scheduled maintenance window. You can modify a DB instance to enable or disable automatic minor version upgrades.

Except for Db2 versions 11.5.9.1 and 11.5.10.0, automatic upgrades to new Db2 minor version includes automatic upgrades to new builds and revisions. For 11.5.9.1 and 11.5.10.0, manually upgrade minor versions.

If you opt out of automatically scheduled upgrades, you can manually upgrade to a supported minor version release by following the same procedure as you would for a major version update. For information, see Upgrading a DB instance engine version.

Supported Db2 major versions on Amazon RDS

RDS for Db2 major versions are available under standard support at least until IBM end of support (base) for the corresponding IBM version. The following table shows the dates that you can use to plan your testing and upgrade cycles. If Amazon extends support for an RDS for Db2 version for longer than originally stated, we plan to update this table to reflect the later date.

You can use the following dates to plan your testing and upgrade cycles.

> **Note**
>
> Dates with only a month and a year are approximate and are updated with an exact date when it's known.
> You can view the major versions of your Db2 databases by running the describe-db-major-engine-versions AWS CLI command or by using the DescribeDBMajorEngineVersions RDS API operation.

Db2 major version	IBM release date	RDS release date	IBM end of support (Standard and Advanced Edition)	IBM end of support (extended)	
Db2 11.5	27 June 2019	27 November 2023	30 April 2027	30 April 2031	

Amazon RDS for Db2 licensing options

Amazon RDS for Db2 has two licensing options: bring your own license (BYOL) and Db2 license through AWS Marketplace.

Topics

- Bring your own license (BYOL) for Db2
- Db2 license through AWS Marketplace
- Switching between Db2 licenses

Bring your own license (BYOL) for Db2

In the BYOL model, you use your existing Db2 database licenses to deploy databases on Amazon RDS. Verify that you have the appropriate Db2 database license for the DB instance class and Db2 database edition that you want to run. You must also follow IBM policies for licensing IBM database software in the cloud computing environment.

> **Note**
>
> Multi-AZ DB instances are cold standbys because the Db2 database is installed but not running. Standbys aren't readable, running, or serving requests. For more information, see IBM Db2 licensing information on the IBM website.

In this model, you continue to use your active IBM support account, and you contact IBM directly for Db2 database service requests. If you have an Support account with case support, you can contact Support for Amazon RDS issues. Amazon Web Services and IBM have a multi-vendor support process for cases that require assistance from both organizations.

Amazon RDS supports the BYOL model for Db2 Standard Edition and Db2 Advanced Edition.

Topics

- IBM IDs for bring your own license (BYOL) for Db2
- Adding IBM IDs to a parameter group for RDS for Db2 DB instances
- Integrating with AWS License Manager

IBM IDs for bring your own license (BYOL) for Db2

In the BYOL model, you need your IBM Customer ID and your IBM Site ID to create, modify, or restore RDS for Db2 DB instances. You must create a custom parameter group with your IBM Customer ID and your IBM Site ID *before* you create an RDS for Db2 DB instance. For more information, see Adding IBM IDs to a parameter group for RDS for Db2 DB instances. You can run multiple RDS for Db2 DB instances with different IBM Customer IDs and IBM Site IDs in the same AWS account or AWS Region.

> **Important**
>
> If we can't verify your license by your IBM Customer ID and your IBM Site ID, we might terminate any DB instances running with these unverified licenses.

If you're a new IBM Db2 customer, you must first purchase a Db2 software license from IBM. After you purchase a Db2 software license, you will receive a Proof of Entitlement from IBM, which lists your IBM Customer ID and your IBM Site ID.

If you're an existing IBM Db2 customer, you can find your IBM Customer ID and your IBM Site ID on your Proof of Entitlement certificate from IBM.

You can also find your IBM Customer ID and your IBM Site ID in your IBM Passport Advantage Online account. After your log in, you can view both IDs on either the main page or the Software downloads page.

Adding IBM IDs to a parameter group for RDS for Db2 DB instances

Because you can't modify default parameter groups, you must create a custom parameter group and then modify it to include the values for your IBM Customer ID and your IBM Site ID. For information about parameter groups, see DB parameter groups for Amazon RDS DB instances.

> **Important**
>
> You must create a custom parameter group with your IBM Customer ID and your IBM Site ID *before* you create an RDS for Db2 DB instance.

Use the parameter settings in the following table.

Parameter	Value
rds.ibm_customer_id	<your IBM Customer ID>
rds.ibm_site_id	<your IBM Site ID>
ApplyMethod	immediate ,pending-reboot

These parameters are dynamic, which means that any changes to them take effect immediately and that you don't need to reboot the DB instance. If you don't want the changes to take effect immediately, you can set `ApplyMethod` to `pending-reboot` and schedule these changes to be made during a maintenance window.

You can create and modify a custom parameter group by using the AWS Management Console, the AWS CLI, or the Amazon RDS API.

Console

To add your IBM Customer ID and your IBM Site ID to a parameter group

1. Create a new DB parameter group. For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.
2. Modify the parameter group that you created. For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.

AWS CLI

To add your IBM Customer ID and your IBM Site ID to a parameter group

1. Create a custom parameter group by running the create-db-parameter-group command.

   Include the following required options:

   - `--db-parameter-group-name` – A name for the parameter group that you are creating.
   - `--db-parameter-group-family` – The Db2 engine edition and major version. Valid values: `db2-se-11.5`, `db2-ae-11.5`.
   - `--description` – A description for this parameter group.

For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.

2. Modify the parameters in the custom parameter group that you created by running the modify-db-parameter-group command.

   Include the following required options:

   - `--db-parameter-group-name` – The name of the parameter group that you created.
   - `--parameters` – An array of parameter names, values, and the application methods for the parameter update.

   For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.

RDS API

To add your IBM Customer ID and your IBM Site ID to a parameter group

1. Create a custom DB parameter group by using the Amazon RDS API CreateDBParameterGroup operation.

   Include the following required parameters:

   - `DBParameterGroupName`
   - `DBParameterGroupFamily`
   - `Description`

   For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.

2. Modify the parameters in the custom parameter group that you created by using the RDS API ModifyDBParameterGroup operation.

   Include the following required parameters:

   - `DBParameterGroupName`
   - `Parameters`

For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.

Now you are ready to create a DB instance and attach the custom parameter group to the DB instance. For more information, see Creating an Amazon RDS DB instance and Associating a DB parameter group with a DB instance in Amazon RDS.

Integrating with AWS License Manager

To aid in monitoring RDS for Db2 license usage in the BYOL model, AWS License Manager integrates with RDS for Db2. License Manager supports tracking of RDS for Db2 engine editions based on virtual CPUs (vCPUs). You can also use License Manager with AWS Organizations to manage all of your organizational accounts centrally.

To track license usage of your RDS for Db2 DB instances, you must create self-managed licenses. You can create self-managed licenses by using the AWS Management Console, the AWS License Manager CLI, and the AWS License Manager API. Or, you can automate the creation of self-managed licenses with CloudFormation and Terraform templates.

RDS for Db2 resources that match the product information filter are automatically associated with the self-managed license. Discovery of RDS for Db2 DB instances can take up to 24 hours.

The following table shows available values for the Engine Edition product information filter for RDS for Db2.

Value	Description
`db2-se`	Db2 Standard Edition
`db2-ae`	Db2 Advanced Edition

Topics

- Terminology
- Creating a self-managed license in AWS License Manager
- Automating the creation of self-managed licenses in AWS License Manager with templates
- Settings for creating self-managed licenses

Terminology

This page uses the following terminology when discussing the Amazon RDS integration with AWS License Manager.

Self-managed license

Self-managed license is a term used in AWS License Manager. The Amazon RDS console refers to the license as an AWS License Manager configuration. A self-managed license contains licensing rules based on the terms of your enterprise agreements. The rules that you create determine how AWS processes commands that consume licenses. While creating a self-managed license, work closely with your organization's compliance team to review your enterprise agreements. For more information, see Self-managed licenses in License Manager.

Creating a self-managed license in AWS License Manager

You can create a self-managed license by using the AWS Management Console, the AWS License Manager CLI, and the AWS License Manager API.

> **Note**
>
> If you create an RDS for Db2 DB instance by using the AWS Management Console, you will create a self-managed license by entering a name for the license. Then Amazon RDS associates the DB instance with this license. (In the Amazon RDS console, this license is referred to as an AWS License Manager configuration.) If you want to create an RDS for Db2 DB instance by using the AWS License Manager CLI or AWS License Manager API, you must first create a self-managed license with the following steps. The same situation applies to restoring an RDS for Db2 DB instance to a point in time or from a snapshot.

Console

To create a self-managed license to track the license usage of your RDS for Db2 DB instances

1. Go to https://console.aws.amazon.com/license-manager/.
2. Create a self-managed license.

   For instructions, see Create a self-managed license in the *AWS License Manager User Guide*.

   Add a rule for an **RDS Product Information Filter** in the **Product Information** panel.

For more information, see ProductInformation in the *AWS License Manager API Reference*.

AWS License Manager CLI

> **Note**
>
> This procedure uses an AWS License Manager CLI command.

To create a self-managed license by using the AWS CLI, run the AWS License Manager create-license-configuration command. Use the `--cli-input-json` or `--cli-input-yaml` options to pass the options to the command.

For more information, see the section called "Settings for creating self-managed licenses".

The following command creates a self-managed license for Db2 Standard Edition.

```
aws license-manager create-license-configuration --cli-input-json file://rds-db2-
se.json
```

The following JSON is the content of the `rds-db2-se.json` file used in the previous command.

```
{
    "Name": "rds-db2-se",
    "Description": "RDS Db2 Standard Edition",
    "LicenseCountingType": "vCPU",
    "LicenseCountHardLimit": false,
    "ProductInformationList": [
        {
            "ResourceType": "RDS",
            "ProductInformationFilterList": [
                {
                    "ProductInformationFilterName": "Engine Edition",
                    "ProductInformationFilterValue": ["db2-se"],
                    "ProductInformationFilterComparator": "EQUALS"
                }
            ]
        }
    ]
}
```

For more information about product information, see Automated discovery of resource inventory in the *AWS License Manager User Guide*.

For more information about the `--cli-input` parameter, see Generating AWS CLI skeleton and input parameters from a JSON or YAML input file in the *AWS CLI User Guide*.

AWS License Manager API

> **Note**
>
> This procedure uses an AWS License Manager API command.

To create a self-managed license, use the `CreateLicenseConfiguration` AWS License Manager API operation with the following required parameters:

- `Name`
- `LicenseCountingType`
- `ProductInformationList`
- `ResourceType`
- `ProductInformationFilterList`
- `ProductInformationFilterName`
- `ProductInformationFilterValue`
- `ProductInformationFilterComparator`

For more information about the parameters, see the section called "Settings for creating self-managed licenses".

Automating the creation of self-managed licenses in AWS License Manager with templates

You can automate the creation of self-managed licenses by using CloudFormation and Terraform templates.

The following example CloudFormation template creates self-managed licenses for Db2 Standard Edition on RDS for Db2. For a template for Db2 Advanced Edition, update the values for `Name`, `Description`, and `ProductInformationFilter`.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: CloudFormation template to create a License Configuration for Db2 Standard
 Edition on RDS for Db2.

Resources:
  Db2LicenseConfiguration:
    Type: "AWS::LicenseManager::LicenseConfiguration"
    Properties:
      Name: "rds-db2-se"
      Description: "Db2 Standard Edition on RDS for Db2"
      LicenseCountingType: "vCPU"
      LicenseCountHardLimit: false
      ProductInformationList:
        - ResourceType: "RDS"
          ProductInformationFilterList:
            - ProductInformationFilterName: "Engine Edition"
              ProductInformationFilterValue:
                - "db2-se"
              ProductInformationFilterComparator: "EQUALS"
```

For more information about using CloudFormation with Amazon RDS, see Creating Amazon RDS resources with AWS CloudFormation.

The following example Terraform template creates self-managed licenses for Db2 Standard Edition on RDS for Db2. Replace *us-east-1* with your AWS Region. For a template for Db2 Advanced Edition, update the values for `name`, `description`, and `product_information_filter`.

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_licensemanager_license_configuration" "rds_db2_license_config" {
  name                     = "rds-db2-se"
  description              = "Db2 Standard Edition on RDS for Db2
  license_counting_type    = "vCPU"
  license_count_hard_limit = false

  product_information_list {
    resource_type = "RDS"

    product_information_filter {
      name        = "Engine Edition"
```

```
      comparator = "EQUALS"
      value      = ["db2-se"]
    }
  }
}
```

For more information about using Terraform and Amazon RDS, see Using Terraform as an IaC tool for the AWS Cloud and Best practices for using the Terraform AWS Provider in *AWS Prescriptive Guidance*.

Settings for creating self-managed licenses

In the following table, you can find details about the settings for creating self-managed licenses by using the AWS License Manager CLI, the AWS License Manager API, an CloudFormation template, and a Terraform template. The parameter name in the following table appears in the format of the name used in the AWS License Manager API and the CloudFormation template.

Parameter name	Data type	Required	Description
Name	string	Yes	The name of the license configuration.
Description	string	No	The description of the license configuration.
LicenseCountingType	string	Yes	The dimension used to track the license inventory. Valid value: vCPU.
LicenseCountHardLimit	boolean	No	Indicates whether hard or soft license enforcement is used. Exceeding a hard limit blocks the launch of new instances.

Parameter name	Data type	Required	Description
ProductInformationList	array of objects	Yes	A list of product information for a license configuration.
ResourceType	string	Yes	The resource type. Valid value: RDS.
ProductInformationFilterList	array of objects	Yes	A list of product information filters for a license configuration.
ProductInformationFilterName	string	Yes	The name of the type of filter being declared. Valid value: `Engine Edition`.
ProductInformationFilterValue	array of strings	Yes	The value to filter on. You must only specify one value. Valid values: `db2-se` or `db2-ae`.
ProductInformationFilterComparator	string	Yes	The logical operator for `ProductInformation FilterName` . Valid value: EQUALS.

Db2 license through AWS Marketplace

In the Db2 license through AWS Marketplace model, you pay an hourly rate to subscribe to Db2 licenses. This model helps you get started quickly with RDS for Db2 without needing to purchase licenses.

To use Db2 license through AWS Marketplace, you need an active AWS Marketplace subscription for the particular IBM Db2 edition that you want to use. If you don't already have one, subscribe to AWS Marketplace for that IBM Db2 edition.

Amazon RDS supports Db2 license through AWS Marketplace for IBM Db2 Standard Edition and IBM Db2 Advanced Edition.

Topics

- Terminology
- Payments and billing
- Subscribing to Db2 Marketplace listings and registering with IBM
- Obtaining a private offer

Terminology

This page uses the following terminology when discussing the Amazon RDS integration with AWS Marketplace.

SaaS subscription

: In AWS Marketplace, software-as-a-service (SaaS) products such as the pay-as-you-go license model adopt a usage-based subscription model. IBM, the software seller for Db2, tracks your usage and you pay only for what you use.

Public offer

: Public offers allow you to purchase AWS Marketplace products directly from the AWS Management Console.

Private offer

: Private offers are a purchasing program that allow sellers and buyers to negotiate custom prices and end user licensing agreement (EULA) terms for purchases in AWS Marketplace.

Db2 Marketplace fees

: Fees charged for the Db2 software license usage by IBM. These service fees are metered through AWS Marketplace and appear on your AWS bill under the AWS Marketplace section.

Amazon RDS fees

Fees that AWS charges for the RDS for Db2 services, which excludes licenses when using AWS Marketplace for Db2 licenses. Fees are metered through the Amazon RDS service being used and appear on your AWS bill.

Payments and billing

RDS for Db2 integrates with AWS Marketplace to offer hourly, pay-as-you-go licenses for Db2. The Db2 Marketplace fees cover the license costs of the Db2 software, and the Amazon RDS fees cover the costs of your RDS for Db2 DB instance usage. For information about pricing, see Amazon RDS for Db2 pricing.

To stop these fees, you must delete any RDS for Db2 DB instances. In addition, you can remove your subscriptions to AWS Marketplace for Db2 licenses. If you remove your subscriptions without deleting your DB instances, Amazon RDS will continue to bill you for the use of the DB instances. For more information, see the section called “Deleting a DB instance”.

You can view bills and manage payments for your RDS for Db2 DB instances that use Db2 license through AWS Marketplace in the AWS Billing console. Your bills includes two charges: one for your usage of Db2 license through AWS Marketplace and one for your usage of Amazon RDS. For more information about billing, see Viewing your bill in the *AWS Billing and Cost Management User Guide*.

Subscribing to Db2 Marketplace listings and registering with IBM

To use Db2 license through AWS Marketplace, you must use the AWS Management Console to complete the following two tasks. You can't complete these tasks through the AWS CLI or the RDS API.

> **Note**
>
> If you want to create your DB instances by using the AWS CLI or the RDS API, you must complete these two tasks first.

Topics

- Task 1: Subscribe to Db2 in AWS Marketplace

- Task 2: Register your subscription with IBM

Task 1: Subscribe to Db2 in AWS Marketplace

To use Db2 license with AWS Marketplace, you need to have an active AWS Marketplace subscription for Db2. Because subscriptions are associated with a specific IBM Db2 edition, you need to subscribe to Db2 in AWS Marketplace for each edition of Db2 that you want to use: IBM Db2 Advanced Edition, IBM Db2 Standard Edition. For information about AWS Marketplace subscriptions, see Saas usage-based subscriptions in the *AWS Marketplace Buyer Guide*.

We recommend that you subscribe to Db2 in AWS Marketplace *before* you start to create a DB instance.

Task 2: Register your subscription with IBM

After you subscribe to Db2 in AWS Marketplace, complete the registration of your IBM order from the AWS Marketplace page for the type of Db2 subscription that you chose. On the AWS Marketplace page, choose **View purchase options**, and then choose **Set up your account**. You can register either with your existing IBM account or by creating a free IBM account.

Obtaining a private offer

You can request an AWS Marketplace private offer for Db2 from IBM. For more information, see Private offers in the *AWS Marketplace Buyer Guide*.

> **Note**
>
> If you are an AWS Organizations user and received a private offer that was issued to your payer and member accounts, follow the procedure below to subscribe to Db2 directly on each account in your organization.

To obtain a Db2 private offer

1. After a private offer has been issued, sign in to the AWS Marketplace Console.
2. Open the email with a Db2 private offer link.
3. Follow the link to directly access the private offer.

> **Note**
>
> Following this link before logging in to the correct account will result in a **Page note found** (404) error.

4. Review the terms and conditions.
5. Choose **Accept terms**.

> **Note**
>
> If an AWS Marketplace private offer is not accepted, the Db2 service fees from AWS Marketplace will continue to be billed at the public hourly rate.

6. To verify the offer details, select **Show details** in the product listing.

After you've completed the procedure, you can create your DB instance by following the steps in the section called "Creating a DB instance". In the AWS Management Console, under **License**, make sure that you choose **Through AWS Marketplace**.

Switching between Db2 licenses

You can switch between Db2 licenses in RDS for Db2. For example, you can start with bring your own license (BYOL), and then switch to Db2 license through AWS Marketplace.

> **Important**
>
> If you want to switch to Db2 license through AWS Marketplace, make sure that you have an active AWS Marketplace subscription for the IBM Db2 edition that you want to use. If you don't, first subscribe to Db2 in AWS Marketplace for that Db2 edition, and then complete the restore procedure.

Console

To switch between Db2 licenses

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Automated backups**.

   The automated backups are displayed on the **Current Region** tab.
3. Choose the DB instance that you want to restore.
4. For **Actions**, choose **Restore to point in time**.

   The **Restore to point in time** window appears.
5. Choose **Latest restorable time** to restore to the latest possible time, or choose **Custom** to choose a time.

   If you chose **Custom**, enter the date and time you want to restore the instance to.

   > **Note**
   >
   > Times are shown in your local time zone, which is indicated by an offset from Coordinated Universal Time (UTC). For example, UTC-5 is Eastern Standard Time/ Central Daylight Time.
6. For **DB engine**, choose the Db2 license you want to use.
7. For **DB instance identifier**, enter the name of the target restored DB instance. The name must be unique.
8. Choose other options as needed, such as DB instance class, storage, and whether you want to use storage autoscaling.

   For information about each setting, see Settings for DB instances.
9. Choose **Restore to point in time**.

For more information, see Restoring a DB instance to a specified time for Amazon RDS.

AWS CLI

To switch between Db2 licenses, run the restore-db-instance-to-point-in-time command. The following example restores the latest point-in-time version, sets the DB engine to IBM Db2 Advanced Edition, and sets the license model to Db2 license through AWS Marketplace.

You can specify other settings. For information about each setting, see Settings for DB instances.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-to-point-in-time \
    --source-db-instance-identifier my_source_db_instance \
    --target-db-instance-identifier my_target_db_instance \
    --use-latest-restorable-time \
    --engine db2-ae \
    --license-model marketplace-license
```

For Windows:

```
aws rds restore-db-instance-to-point-in-time ^
    --source-db-instance-identifier my_source_db_instance ^
    --target-db-instance-identifier my_target_db_instance ^
    --use-latest-restorable-time ^
    --engine db2-ae ^
    --license-model marketplace-license
```

For more information, see Restoring a DB instance to a specified time for Amazon RDS.

RDS API

To switch between Db2 licenses, call the Amazon RDS API `RestoreDBInstanceToPointInTime` operation with the following parameters:

- `SourceDBInstanceIdentifier`
- `TargetDBInstanceIdentifier`
- `RestoreTime`
- `Engine`
- `LicenseModel`

For more information, see Restoring a DB instance to a specified time for Amazon RDS.

Amazon RDS for Db2 instance classes

The computation and memory capacity of a DB instance is determined by its instance class. The DB instance class you need depends on your processing power and memory requirements.

Supported RDS for Db2 instance classes

The supported Amazon RDS for Db2 instance classes are a subset of the Amazon RDS DB instance classes. For the complete list of Amazon RDS instance classes, see DB instance classes.

Topics

- Supported RDS for Db2 instance classes for Db2 Standard Edition
- Supported RDS for Db2 instance classes for Db2 Advanced Edition

Supported RDS for Db2 instance classes for Db2 Standard Edition

The following table lists all instance classes supported for the Db2 Standard Edition of Db2 database version 11.5.9.0. These instance classes are available for both bring your own license (BYOL) and Db2 license through AWS Marketplace.

Instance class type	Instance class
General purpose instance classes with 3rd generation Intel Xeon Scalable processors, SSD storage, and network optimization	db.m6idn.large–db.m6idn.8xlarge
General purpose instance classes powered by 3rd generation Intel Xeon Scalable processors	db.m6in.large–db.m6in.8xlarge
General purpose instance classes	db.m7i.large–db.m7i.8xlarge db.m6i.large–db.m6i.8xlarge
Memory optimized instance classes with local NVMe-based SSDs, powered by 3rd generation Intel Xeon Scalable processors	db.x2iedn.xlarge
Memory optimized instance classes powered by 3rd generation Intel Xeon Scalable processors	db.r6idn.large–db.r6idn.4xlarge db.r6in.large–db.r6in.4xlarge
Memory optimized instance classes	db.r7i.large–db.r7i.8xlarge

Instance class type	Instance class
	db.r6i.large–db.r6i.4xlarge
Burstable performance instance classes	db.t3.small–db.t3.2xlarge

Supported RDS for Db2 instance classes for Db2 Advanced Edition

The following table lists all instance classes supported for the Db2 Advanced Edition of Db2 database version 11.5.9.0. These instance classes are available for both bring your own license (BYOL) and Db2 license through AWS Marketplace.

Instance class type	Instance class
General purpose instance classes with 3rd generation Intel Xeon Scalable processors, SSD storage, and network optimization	db.m6idn.large–db.m6idn.32xlarge
General purpose instance classes powered by 3rd generation Intel Xeon Scalable processors	db.m6in.large–db.m6in.32xlarge
General purpose instance classes	db.m6i.large–db.m7i.48xlarge db.m7i.large–db.m7i.48xlarge
Memory optimized instance classes with local NVMe-based SSDs, powered by 3rd generation Intel Xeon Scalable processors	db.x2iedn.xlarge–db.x2iedn.32xlarge
Memory optimized instance classes powered by 3rd generation Intel Xeon Scalable processors	db.r6idn.large–db.r6idn.32xlarge db.r6in.8xlarge–db.r6in.32xlarge
Memory optimized instance classes	db.r6i.large–db.r7i.48xlarge db.r7i.large–db.r7i.48xlarge

Instance class type	Instance class
Burstable performance instance classes	db.t3.small–db.t3.2xlarge

Amazon RDS for Db2 default roles

RDS for Db2 adds the following six roles and grants them to the `master_user_role` with the `ADMIN` option. When the database is provisioned, RDS for Db2 grants `master_user_role` to the master user. The master user can in turn grant these roles to other users, groups, or roles with native `GRANT` statements by connecting to the database.

- **DBA** – RDS for Db2 creates this empty role with `DATAACCESS` authorization. The master user can add more authorizations or privileges to this role, and then grant the role to other users, groups, or roles.
- **DBA_RESTRICTED** – RDS for Db2 creates this empty role. The master user can add privileges to this role, and then grant the role to other users, groups, or roles.
- **DEVELOPER** – RDS for Db2 creates this empty role with `DATAACCESS` authorization. The master user can add more authorizations or privileges to this role, and then grant the role to other users, groups, or roles.
- **ROLE_NULLID_PACKAGES** – RDS for Db2 grants `EXECUTE` privileges to this role on `ALL NULLID` packages that were bound by Db2 when `CREATE DATABASE` was run.
- **ROLE_PROCEDURES** – RDS for Db2 grants `EXECUTE` privileges to this role on all `SYSIBM` procedures.
- **ROLE_TABLESPACES** – RDS for Db2 grants `USAGE` privileges on tablespaces created by the `CREATE DATABASE` command.

Amazon RDS for Db2 parameters

Amazon RDS for Db2 uses three types of parameters: database manager configuration parameters, registry variables, and database configuration parameters. You can manage the first two types through parameter groups and the last type through the rdsadmin.update_db_param stored procedure.

By default, an RDS for Db2 DB instance uses a DB parameter group that is specific to a Db2 database and DB instance. This parameter group contains parameters for the IBM Db2 database

engine, specifically the database manager configuration parameters and registry variables. For information about working with parameter groups, see Parameter groups for Amazon RDS.

RDS for Db2 database configuration parameters are set to the default values of the storage engine that you have selected. For more information about Db2 parameters, see the Db2 database configuration parameters in the IBM Db2 documentation.

Topics

- Viewing the parameters in parameter groups
- Viewing all parameters with Db2 commands
- Modifying the parameters in parameter groups
- Modifying the database configuration parameters with Db2 commands

Viewing the parameters in parameter groups

The database manager configuration parameters and the registry variables are set in parameter groups. You can view the database manager configuration parameters and the registry variables for a specific Db2 version by using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To view the parameter values for a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.

   The DB parameter groups appear in a list.
3. Choose the name of the parameter group to see its list of parameters.

AWS CLI

You can view the database manager configuration parameters and the registry variables for a Db2 version by running the describe-engine-default-parameters command. Specify one of the following values for the `--db-parameter-group-family` option:

- `db2-ae-11.5`
- `db2-se-11.5`

For example, to view the parameters for Db2 Standard Edition 11.5, run the following command:

```
aws rds describe-engine-default-parameters --db-parameter-group-family db2-se-11.5
```

This command produces output similar to the following example:

```
{
    "EngineDefaults": {
        "Parameters": [
            {
                "ParameterName": "agent_stack_sz",
                "ParameterValue": "1024",
                "Description": "You can use this parameter to determine the amount of
 memory that is allocated by Db2 for each agent thread stack.",
                "Source": "engine-default",
                "ApplyType": "static",
                "DataType": "integer",
                "AllowedValues": "256-32768",
                "IsModifiable": false
            },
            {
                "ParameterName": "agentpri",
                "ParameterValue": "-1",
                "Description": "This parameter controls the priority given to all
 agents and to other database manager instance processes and threads by the operating
 system scheduler. This priority determines how CPU time is allocated to the database
 manager processes, agents, and threads relative to other processes and threads running
 on the machine.",
                "Source": "engine-default",
                "ApplyType": "static",
                "DataType": "integer",
                "AllowedValues": "1-99",
                "IsModifiable": false
            },
            ...
        ]
    }
}
```

To list only the modifiable parameters for Db2 Standard Edition 11.5, run the following command:

For Linux, macOS, or Unix:

```
aws rds describe-engine-default-parameters \
    --db-parameter-group-family db2-se-11.5 \
    --query 'EngineDefaults.Parameters[?IsModifiable==`true`].
{ParameterName:ParameterName, DefaultValue:ParameterValue}'
```

For Windows:

```
aws rds describe-engine-default-parameters ^
    --db-parameter-group-family db2-se-11.5 ^
    --query 'EngineDefaults.Parameters[?IsModifiable==`true`].
{ParameterName:ParameterName, DefaultValue:ParameterValue}'
```

RDS API

To view the parameter values for a DB parameter group, use the `DescribeDBParameters` operation with the following required parameter.

- `DBParameterGroupName`

Viewing all parameters with Db2 commands

You can view the settings for database manager configuration parameters, database configuration parameters, and registry variables by using Db2 commands.

To view the settings

1. Connect to your Db2 database. In the following example, replace *database_name*, *master_username*, and *master_password* with your information.

   ```
   db2 "connect to database_name user master_username using master_password"
   ```

2. Find the supported Db2 version.

   ```
   db2 "select service_level, fixpack_num from table(sysproc.env_get_inst_info()) as
    instanceinfo"
   ```

3. View the parameters for a specific Db2 version.

   - View database manager configuration parameters by running the following command:

```
db2 "select cast(substr(name,1,24) as varchar(24)) as name, case
    when value_flags = 'NONE' then '' else value_flags end flags,
    cast(substr(value,1,64) as varchar(64)) as current_value
    from sysibmadm.dbmcfg
    order by name asc with UR"
```

- View all of your database configuration parameters by running the following command:

```
db2 "select cast(substr(name,1,24) as varchar(24)) as name, case
    when value_flags = 'NONE' then '' else value_flags end flags,
    cast(substr(value,1,64) as varchar(64)) as current_value
    from table(db_get_cfg(null)) order by name asc, member asc with UR"
```

- View the currently set registry variables by running the following command:

```
db2 "select cast(substr(reg_var_name,1,50) as varchar(50)) as reg_var_name,
    cast(substr(reg_var_value,1,50) as varchar(50)) as reg_var_value,
    level from table(env_get_reg_variables(null))
    order by reg_var_name,member with UR"
```

Modifying the parameters in parameter groups

You can modify the database manager configuration parameters and the registry variables in custom parameter groups by using the AWS Management Console, the AWS CLI, or the RDS API. For more information, see DB parameter groups for Amazon RDS DB instances.

Console

To modify database manager configuration parameters and registry variables

1. Create a custom parameter group. For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.
2. Modify the parameters in that custom parameter group. For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.

AWS CLI

To modify database manager configuration parameters and registry variables

1. Create a custom parameter group by running the create-db-parameter-group command.

   Include the following required options:

   - `--db-parameter-group-name` – A name for the parameter group that you are creating.
   - `--db-parameter-group-family` – The Db2 engine edition and major version. Valid values: `db2-se-11.5`, `db2-ae-11.5`.
   - `--description` – A description for this parameter group.

   For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.
2. Modify the parameters in the custom parameter group that you created by running the modify-db-parameter-group command.

   Include the following required options:

   - `--db-parameter-group-name` – The name of the parameter group that you created.
   - `--parameters` – An array of parameter names, values, and the application methods for the parameter update.

   For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.

RDS API

To modify database manager configuration parameters and registry variables

1. Create a custom DB parameter group by using the CreateDBParameterGroup operation.

   Include the following required parameters:

   - `DBParameterGroupName`
   - `DBParameterGroupFamily`

- `Description`

For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.

2. Modify the parameters in the custom parameter group that you created by using the ModifyDBParameterGroup operation.

   Include the following required parameters:

   - `DBParameterGroupName`
   - `Parameters`

   For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.

Modifying the database configuration parameters with Db2 commands

You can modify the database configuration parameters with Db2 commands.

To modify the database configuration parameters

1. Connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Change the database configuration parameters by calling the `rdsadmin.update_db_param` stored procedure. For more information, see rdsadmin.update_db_param.

```
db2 "call rdsadmin.update_db_param(
    'database_name',
    'parameter_to_modify',
    'changed_value',
    'restart_database')"
```

EBCDIC collation for Db2 databases on Amazon RDS

Amazon RDS for Db2 supports EBCDIC collation for Db2 databases. You can only specify an EBCDIC collation sequence for a database when you create the database by using the Amazon RDS the section called “rdsadmin.create_database” stored procedure.

When you create an RDS for Db2 DB instance by using the AWS Management Console, AWS CLI, or RDS API, you can specify a database name. If you specify a database name, Amazon RDS creates a database with the default collation of SYSTEM. If you need to create a database with EBCDIC collation, don't specify a database name when you create a DB instance.

The collation for a database in RDS for Db2 is set at the time of creation and is immutable.

To create a Db2 database with EBCDIC collation

1. If you don't have an RDS for Db2 DB instance, create a DB instance but don't specify a database name You can create a DB instance by using the AWS Management Console, AWS CLI, or RDS API. For more information, see Creating a DB instance.
2. Create a Db2 database and set the collation option to an EBCDIC value by calling the `rdsadmin.create_database` stored procedure. For more information, see rdsadmin.create_database.

   > **Important**
   >
   > After you create a database using the stored procedure, you can't change the collation sequence. If you want a database to use a different collation sequence, drop the database by calling the the section called “rdsadmin.drop_database” stored procedure. Then, create a database with the required collation sequence.

Local time zone for Amazon RDS for Db2 DB instances

The time zone of an Amazon RDS DB instance running Db2 is set by default. The default is Coordinated Universal Time (UTC). To match the time zone of your applications, you can set the time zone of your DB instance to a local time zone instead.

You set the time zone when you first create your DB instance. You can create your DB instance by using the AWS Management Console, the RDS API, or the AWS CLI. For more information, see Creating a DB instance.

If your DB instance is part of a Multi-AZ deployment, then when it fails over, its time zone remains the local time zone that you set.

You can restore your DB instance to a point in time that you specify. The time appears in your local time zone. For more information, see Restoring a DB instance to a specified time for Amazon RDS.

Setting the local time zone on your DB instance has the following limitations:

- You can't modify the time zone of an existing Amazon RDS for Db2 DB instance.
- You can't restore a snapshot from a DB instance in one time zone to a DB instance in a different time zone.
- We strongly recommend that you don't restore a backup file from one time zone to a different time zone. If you restore a backup file from one time zone to another, then you must audit your queries and applications for the effects of the time zone change.

Available time zones

You can use the following values for the time zone setting.

Zone	Time zone
Africa	Africa/Cairo, Africa/Casablanca, Africa/Harare, Africa/Lagos, Africa/Luanda, Africa/Monrovia, Africa/Nairobi, Africa/Tripoli, Africa/Windhoek
America	America/Araguaina, America/Argentina/Buenos_Aires, America/Asuncion, America/Bogota, America/Caracas, America/Chicago, America/Chihuahua, America/Cuiaba, America/Denver, America/Detroit, America/Fortaleza, America/Godthab, America/Guatemala, America/Halifax, America/Lima, America/Los_Angeles, America/Manaus, America/Matamoros, America/Mexico_City, America/Monterrey, America/Montevideo, America/New_York, America/Phoenix, America/Santiago, America/Sao_Paulo, America/Tijuana, America/Toronto
Asia	Asia/Amman, Asia/Ashgabat, Asia/Baghdad, Asia/Baku, Asia/Bangkok, Asia/Beirut, Asia/Calcutta, Asia/Damascus, Asia/Dhaka, Asia/Hong_Kong, Asia/Irkutsk, Asia/Jakarta, Asia/Jerusalem, Asia/Kabul, Asia/Karachi, Asia/Kathmandu, Asia/Kolkata, Asia/Krasnoyarsk, Asia/Magadan, Asia/Manila, Asia/Muscat, Asia/Novosibirsk, Asia/Rangoon, Asia/Riyadh, Asia/Seoul, Asia/

Zone	Time zone
	Shanghai, Asia/Singapore, Asia/Taipei, Asia/Tehran, Asia/Tokyo, Asia/Ulaanbaatar, Asia/Vladivostok, Asia/Yakutsk, Asia/Yerevan
Atlantic	Atlantic/Azores, Atlantic/Cape_Verde
Australia	Australia/Adelaide, Australia/Brisbane, Australia/Darwin, Australia/Eucla, Australia/Hobart, Australia/Lord_Howe, Australia/Perth, Australia/Sydney
Brazil	Brazil/DeNoronha, Brazil/East
Canada	Canada/Newfoundland, Canada/Saskatchewan
Etc	Etc/GMT-3
Europe	Europe/Amsterdam, Europe/Athens, Europe/Berlin, Europe/Dublin, Europe/Helsinki, Europe/Kaliningrad, Europe/London, Europe/Madrid, Europe/Moscow, Europe/Paris, Europe/Prague, Europe/Rome, Europe/Sarajevo, Europe/Stockholm
Pacific	Pacific/Apia, Pacific/Auckland, Pacific/Chatham, Pacific/Fiji, Pacific/Guam, Pacific/Honolulu, Pacific/Kiritimati, Pacific/Marquesas, Pacific/Samoa, Pacific/Tongatapu, Pacific/Wake
US	US/Alaska, US/Central, US/East-Indiana, US/Eastern, US/Pacific
UTC	UTC

Prerequisites for creating an Amazon RDS for Db2 DB instance

The following items are prerequisites before creating a DB instance.

Topics

- Administrator account
- Additional considerations

Administrator account

When you create a DB instance, you must designate an administrator account for the instance. Amazon RDS grants `DBADM` authority to this local database administrator account.

The administrator account has the following characteristics, capabilities, and limitations:

- Is a local user and not an AWS account.
- Doesn't have Db2 instance-level authorities such as `SYSADM`, `SYSMAINT`, or `SYSCTRL`.
- Can't stop or start a Db2 instance.
- Can't drop a Db2 database if you specified the name when you created the DB instance.
- Has full access to the Db2 database including catalog tables and views.
- Can create local users and groups by using Amazon RDS stored procedures.
- Can grant and revoke authorities and privileges.

The administrator account can perform the following tasks:

- Create, modify, or delete DB instances.
- Create DB snapshots.
- Initiate point-in-time restores.
- Create automated backups of DB snapshots.
- Create manual backups of DB snapshots.
- Use other Amazon RDS features.

Additional considerations

Before creating a DB instance, consider the following items:

- Each Amazon RDS for Db2 DB instance can host up to 50 Db2 databases. For more information, see Multiple databases on an Amazon RDS for Db2 DB instance.
- Initial database name
  - If you don't provide a database name when you create a DB instance, Amazon RDS doesn't create a database.
  - Don't provide a database name under the following circumstances:
    - You want to modify the `db2_compatibility_vector` parameter. For more information, see Setting the db2_compatibility_vector parameter.
- In the bring your own license (BYOL) model, you must first create a custom parameter group that contains your IBM Customer ID and your IBM Site ID. For more information, see Bring your own license (BYOL) for Db2.
- In the Db2 license through AWS Marketplace model, you need an active AWS Marketplace subscription for the particular IBM Db2 edition that you want to use. If you don't already have one, subscribe to Db2 in AWS Marketplace for the IBM Db2 edition that you want to use. For more information, see Db2 license through AWS Marketplace.

Multiple databases on an Amazon RDS for Db2 DB instance

You can create multiple databases on a single RDS for Db2 DB instance by calling the rdsadmin.create_database stored procedure. A single RDS for Db2 DB instance is limited to 50 databases. This number includes databases in both activated and deactivated states.

> **Note**
>
> If you create multiple databases on an RDS for Db2 DB instance that was created before November 15, 2024, then you must reboot the DB instance to enable support for multiple databases.

By default, Amazon RDS activates databases when you create them. To optimize memory resources, you can deactivate databases that you use infrequently and then activate them later when needed. For more information, see the section called “Deactivating a database” and the section called “Activating a database”.

The number of activated databases on a DB instance depends on the available memory resources on the server. Memory resources differ based on the DB instance class and the amount of memory configured for the database. For information about DB instance classes, see the section called “DB instance classes”. For information about how to update the memory for an RDS for Db2 database, see the section called “rdsadmin.update_db_param”.

We recommend that you choose a DB instance class that has 2 GB of memory for common database tasks, operating system requirements, and other Amazon RDS automation tasks such as backups. For more information about changing the DB instance class, see the section called “Modifying a DB instance”.

In addition, IBM recommends a minimum of 1 GB of memory for each active database. For more information, see Disk and memory requirements in the IBM documentation.

You can calculate the maximum number of active databases a DB instance can have with the following formula:

```
Active database limit = (total server memory - 2 GB) / 1 GB
```

The following example shows the maximum number of active databases for a DB instance with a db.m6i.xlarge DB instance class:

```
Active database limit = (total server memory - 2 GB) / 1 GB
                      = (16 GB - 2 GB) / 1 GB
                      = 14 databases
```

When Amazon RDS recovers a database after a crash, it activates the database if it was previously active. In certain cases, such as when you modify a DB instance class to a lower memory configuration, there might be insufficient memory to activate all databases on the DB instance. In those cases, Amazon RDS activates databases in the order in which they were created.

> **Note**
>
> Any databases that Amazon RDS can't activate because of insufficient memory remain in a deactivated state.

Connecting to your Db2 DB instance

After Amazon RDS provisions your Amazon RDS for Db2 DB instance, you can use any standard SQL client application to connect to the DB instance. Because Amazon RDS is a managed service, you can't sign in as `SYSADM`, `SYSCTRL`, `SECADM`, or `SYSMAINT`.

You can connect to a DB instance that is running the IBM Db2 database engine by using IBM Db2 CLP, IBM CLPPlus, DBeaver, or IBM Db2 Data Management Console.

> **Note**
>
> Connecting to a Db2 database can fail if your RDS for Db2 DB instance doesn't have sufficient memory. For more information, see the section called “Database connection error”.

Topics

- Finding the endpoint of your Amazon RDS for Db2 DB instance
- Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 CLP
- Connecting to your Amazon RDS for Db2 DB instance with IBM CLPPlus
- Connecting to your Amazon RDS for Db2 DB instance with DBeaver
- Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 Data Management Console
- Considerations for security groups with Amazon RDS for Db2

Finding the endpoint of your Amazon RDS for Db2 DB instance

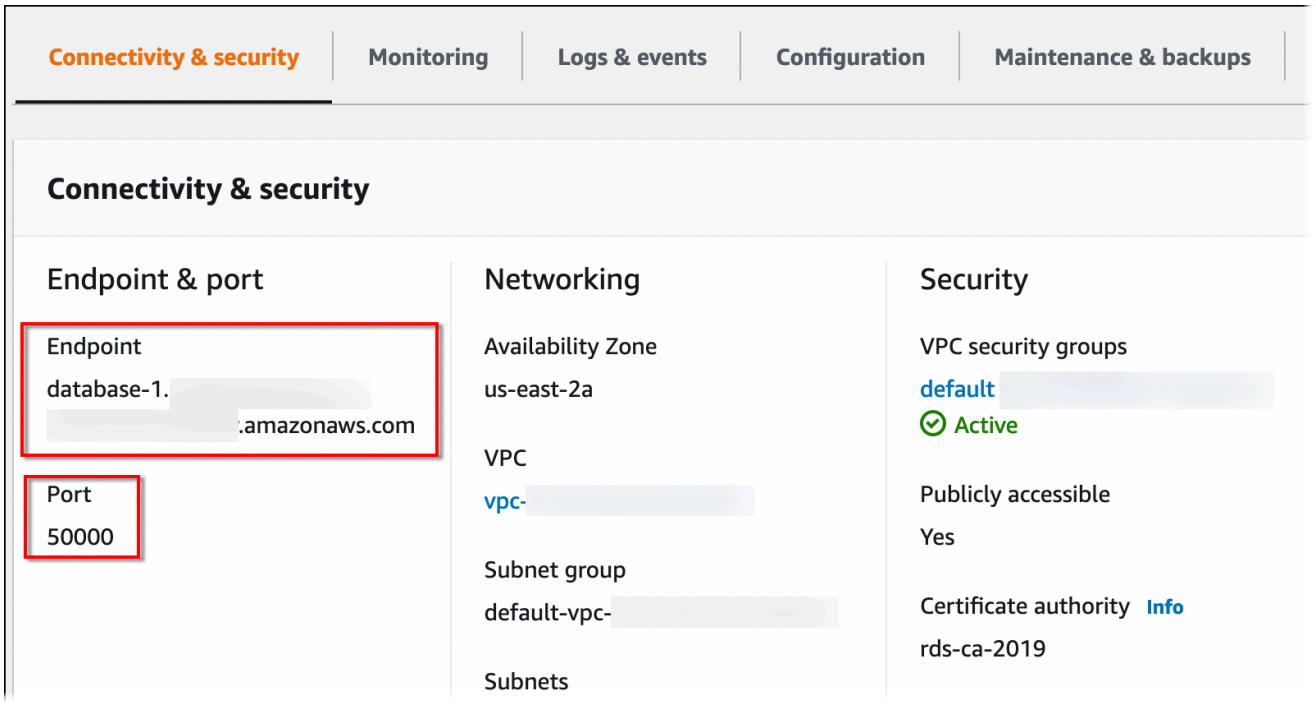
Each Amazon RDS DB instance has an endpoint, and each endpoint has the DNS name and port number for the DB instance. To connect to your Amazon RDS for Db2 DB instance with a SQL client application, you need the DNS name and port number for your DB instance.

You can find the endpoint for a DB instance by using the AWS Management Console or the AWS CLI.

Console

To find the endpoint of an RDS for Db2 DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the console, choose the AWS Region of your DB instance.
3. Find the DNS name and port number for your RDS for Db2 DB Instance.
   a. Choose **Databases** to display a list of your DB instances.
   b. Choose the RDS for Db2 DB instance name to display the instance details.
   c. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



AWS CLI

To find the endpoint of an RDS for Db2 DB instance, run the describe-db-instances command. In the following example, replace *database-1* with the name of your DB instance.

For Linux, macOS, or Unix:

```
aws rds describe-db-instances \
```

```
    --db-instance-identifier database-1 \
    --query 'DBInstances[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBName:DBName,Endpoint:Endpoint}' \
    --output json
```

For Windows:

```
aws rds describe-db-instances ^
    --db-instance-identifier database-1 ^
    --query 'DBInstances[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBName:DBName,Endpoint:Endpoint}' ^
    --output json
```

This command produces output similar to the following example. The `Address` line in the output contains the DNS name.

```
[
    {
        "DBInstanceIdentifier": "database-1",
        "DBName": "DB2DB",
        "Endpoint": {
            "Address": "database-1.123456789012.us-east-2.amazonaws.com",
            "Port": 50000,
            "HostedZoneId": "Z2OC4A7DETW6VH"
        }
    }
]
```

Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 CLP

You can use a command line utility such as IBM Db2 CLP to connect to Amazon RDS for Db2 DB instances. This utility is part of IBM Data Server Runtime Client. To download the client from IBM Fix Central, see IBM Data Server Client Packages Version 11.5 Mod 8 Fix Pack 0 in IBM Support.

Topics

- Terminology
- Installing the client
- Connecting to a DB instance
- Troubleshooting connections to your RDS for Db2 DB instance

Terminology

The following terms help explain commands used when connecting to your RDS for Db2 DB instance.

catalog tcpip node

This command registers a remote database node with a local Db2 client, which makes the node accessible to the client application. To catalog a node, you provide information such as the server's host name, port number, and communication protocol. The cataloged node then represents a target server where one or more remote databases reside. For more information, see CATALOG TCPIP/TCPIP4/TCPIP6 NODE command in the IBM Db2 documentation.

catalog database

This command registers a remote database with a local Db2 client, which makes the database accessible to the client application. To catalog a database, you provide information such as the database's alias, the node on which it resides, and the authentication type needed to connect to the database. For more information, see CATALOG DATABASE command in the IBM Db2 documentation.

Installing the client

After downloading the package for Linux, install the client using root or administrator privileges.

> **Note**
>
> To install the client on AIX or Windows, follow the same procedure but modify the commands for your operating system.

To install the client on Linux

1. Run **`./db2_install -f sysreq`** and choose **`yes`** to accept the license.
2. Choose the location to install the client.
3. Run **`clientInstallDir/instance/db2icrt -s client`** ***`instance_name`***. Replace *`instance_name`* with a valid operating system user on Linux. In Linux, the Db2 DB instance name is tied to the operating system username.

This command creates a **sqllib** directory in the home directory of the designated user on Linux.

Connecting to a DB instance

To connect to your RDS for Db2 DB instance, you need its DNS name and port number. For information about finding them, see Finding the endpoint. You also need to know the database name, master username, and master password that you defined when you created your RDS for Db2 DB instance. For more information about finding them, see Creating a DB instance.

To connect to an RDS for Db2 DB instance with IBM Db2 CLP

1. Sign in with the username that you specified during the IBM Db2 CLP client installation.
2. Catalog your RDS for Db2 DB instance. In the following example, replace *node_name*, *dns_name*, and *port* with a name for the node in the local catalog, the DNS name for your DB instance, and the port number.

   ```
   db2 catalog TCPIP node node_name remote dns_name server port
   ```

   Example

   ```
   db2 catalog TCPIP node remnode remote database-1.123456789012.us-
   east-1.amazonaws.com server 50000
   ```

3. Catalog the `rdsadmin` database and your database. This will allow you to connect to the `rdsadmin` database to perform some administrative tasks using Amazon RDS stored procedures. For more information, see Administering your RDS for Db2 DB instance.

   In the following example, replace *database_alias*, *node_name*, and *database_name* with aliases for this database, the name of the node defined in the previous step, and the name of your database. `server_encrypt` encrypts your username and password over the network.

   ```
   db2 catalog database rdsadmin [ as database_alias ] at node node_name
    authentication server_encrypt

   db2 catalog database database_name [ as database_alias ] at node node_name
    authentication server_encrypt
   ```

Example

```
db2 catalog database rdsadmin at node remnode authentication server_encrypt

db2 catalog database testdb as rdsdb2 at node remnode authentication server_encrypt
```

4. Connect to your RDS for Db2 database. In the following example, replace *rds_database_alias*, *master_username*, and *master_password* with the name of your database, the master username, and master password of your RDS for Db2 DB instance.

```
db2 connect to rds_database_alias user master_username using master_password
```

This command produces output similar to the following example:

```
Database Connection Information

    Database server        = DB2/LINUXX8664 11.5.9.0
    SQL authorization ID   = ADMIN
    Local database alias   = TESTDB
```

5. Run queries and view results. The following example shows a SQL statement that selects the database you created.

```
db2 "select current server from sysibm.dual"
```

This command produces output similar to the following example:

```
1
    ------------------
    TESTDB

    1 record(s) selected.
```

Troubleshooting connections to your RDS for Db2 DB instance

If you receive the following `NULLID` error, it usually indicates that your client and RDS for Db2 server versions don't match. For supported Db2 client versions, see Supported combinations of clients, drivers and server levels in the IBM Db2 documentation.

```
db2 "select * from syscat.tables"
SQL0805N Package "NULLID.SQLC2O29 0X4141414141454A69" was not found.
SQLSTATE=51002
```

After you receive this error, you must bind packages from your older Db2 client to a Db2 server version supported by RDS for Db2.

To bind packages from an older Db2 client to a newer Db2 server

1. Locate the bind files on the client machine. Typically, these files are located in the **bnd** directory of the Db2 client's installation path and have the extension **.bnd**.
2. Connect to the Db2 server. In the following example, replace *database_name* with the name of your Db2 server. Replace *master_username* and *master_password* with your information. This user has DBADM authority.

   ```
   db2 connect to database_name user master_username using master_password
   ```
3. Run the bind command to bind the packages.

   a. Navigate to the directory where the bind files exist on the client machine.

   b. Run the bind command for each file.

   The following options are required:

   - `blocking all` – Binds all packages in the bind file in a single database request.
   - `grant public` – Grants permission to public to execute the package.
   - `sqlerror continue` – Specifies that the bind process continues even if errors occur.

   For more information about the bind command see BIND command in the IBM Db2 documentation.
4. Verify that the bind was successful by either querying the `syscat.package` catalog view or checking the message returned after the bind command.

For more information, see DB2 v11.5 Bind File and Package Name List in IBM Support.

Connecting to your Amazon RDS for Db2 DB instance with IBM CLPPlus

You can use a utility such as IBM CLPPlus to connect to an Amazon RDS for Db2 DB instance. This utility is part of IBM Data Server Runtime Client. To download the client from IBM Fix Central, see IBM Data Server Client Packages Version 11.5 Mod 8 Fix Pack 0 in IBM Support.

Important

We recommend that you run IBM CLPPlus on an operating system that supports graphical user interfaces such as macOS, Windows, or Linux with Desktop. If running headless Linux, use switch **-nw** with CLPPlus commands.

Topics

- Installing the client
- Connecting to a DB instance
- Retrieving CLOB Data from DB2 Stored Procedures

Installing the client

After downloading the package for Linux, install the client.

Note

To install the client on AIX or Windows, follow the same procedure but modify the commands for your operating system.

To install the client on Linux

1. Run **`./db2_install`**.
2. Run **`clientInstallDir/instance/db2icrt -s client`** ***`instance_name`***. Replace *`instance_name`* with a valid operating system user on Linux. In Linux, the Db2 DB instance name is tied to the operating system username.

   This command creates a **`sqllib`** directory in the home directory of the designated user on Linux.

Connecting to a DB instance

To connect to your RDS for Db2 DB instance, you need its DNS name and port number. For information about finding them, see Finding the endpoint. You also need to know the database name, master username, and master password that you defined when you created your RDS for Db2 DB instance. For more information about finding them, see Creating a DB instance.

To connect to an RDS for Db2 DB instance with IBM CLPPlus

1. Review the command syntax. In the following example, replace *clientDir* with the location where the client is installed.

```
cd clientDir/bin
    ./clpplus -h
```

2. Configure your Db2 server. In the following example, replace *dsn_name*, *database_name*, *endpoint*, and *port* with the DSN name, database name, endpoint, and port for your RDS for Db2 DB instance. For more information, see Finding the endpoint of your Amazon RDS for Db2 DB instance.

```
db2cli writecfg add -dsn dsn_name -database database_name -host endpoint -port port
 -parameter "Authentication=SERVER_ENCRYPT"
```

3. Connect to your RDS for Db2 DB instance. In the following example, replace *master_username* and *dsn_name* with the master username and DSN name.

```
./clpplus -nw master_username@dsn_name
```

4. A Java Shell window opens. Enter the master password for your RDS for Db2 DB instance.

> **Note**
>
> If a Java Shell window doesn't open, run **`./clpplus -nw`** to use the same command line window.

```
Enter password: *********
```

A connection is made and produces output similar to the following example:

```
Database Connection Information :
--------------------------------
Hostname = database-1.abcdefghij.us-east-1.rds.amazonaws.com
Database server = DB2/LINUXX8664  SQL110590
SQL authorization ID = admin
Local database alias = DB2DB
Port = 50000
```

5. Run queries and view results. The following example shows a SQL statement that selects the database you created.

```
SQL > select current server from sysibm.dual;
```

This command produces output similar to the following example:

```
1
    --------------------
    DB2DB
    SQL>
```

Retrieving CLOB Data from DB2 Stored Procedures

Stored procedures like rdsadmin.db2pd_command return results in CLOB columns, which support up to 2 GB of data. However, DB2 CLP limits CLOB output to 8 KB (8192 bytes), truncating any data beyond this threshold. To retrieve the complete output, use CLPPLUS instead.

1. Get Task ID (task_id)

```
db2 "select task_id, task_type, database_name, lifecycle,
 varchar(bson_to_json(task_input_params), 500) as task_params,
cast(task_output as varchar(500)) as task_output, CREATED_AT, LAST_UPDATED_AT from
 table(rdsadmin.get_task_status(null,null,null))"
```

2. Execute CLPPLUS Command

After obtaining the task_id, execute the following command from the Unix prompt (replace TASK_ID with the actual numeric task ID):

```
$ (echo "select task_output from
 table(rdsadmin.get_task_status(task_id,null,null));" ; echo "disconnect;" ; echo
 "exit;") | clpplus -nw -silent masteruser/MasterUserPassword@hostname:port_num/
rdsadmin
```

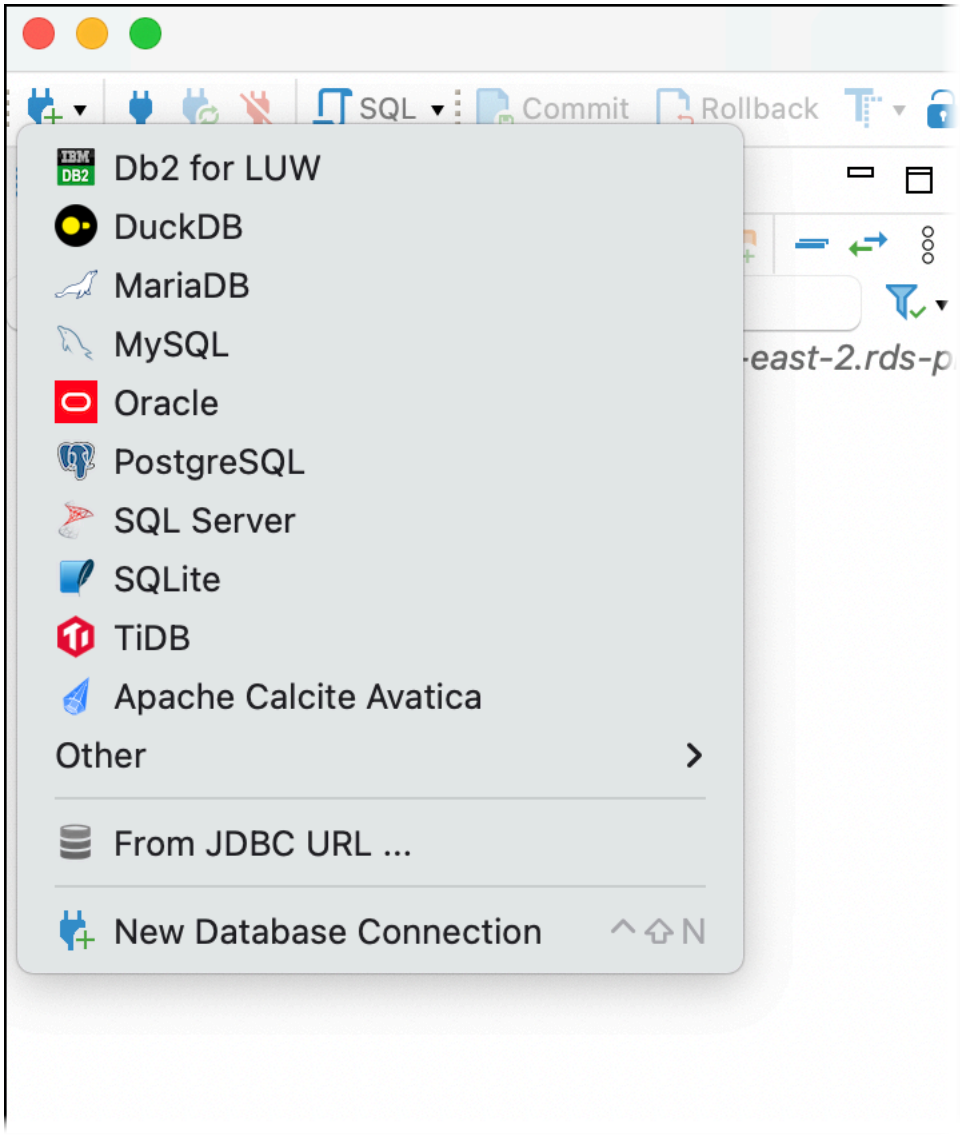
Connecting to your Amazon RDS for Db2 DB instance with DBeaver

You can use third-party tools such as DBeaver to connect to Amazon RDS for Db2 DB instances. To download this utility, see DBeaver Community.

To connect to your RDS for Db2 DB instance, you need its DNS name and port number. For information about finding them, see Finding the endpoint. You also need to know the database name, master username, and master password that you defined when you created your RDS for Db2 DB instance. For more information about finding them, see Creating a DB instance.

To connect to an RDS for Db2 DB instance with DBeaver

1. Start **DBeaver**.
2. Choose the **New Connection** icon in the toolbar and then choose **Db2 for LUW**.



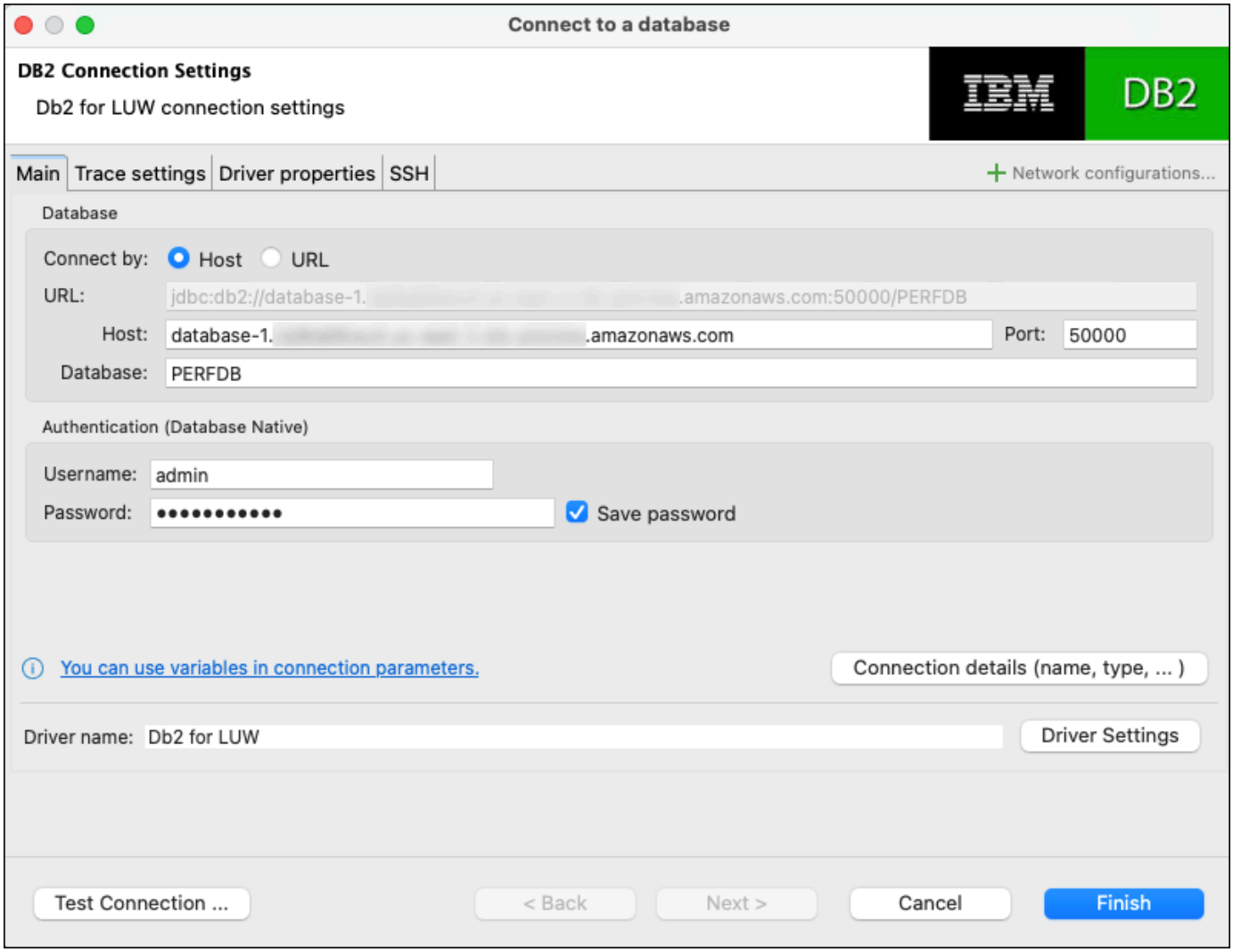
3. In the **Connect to a database** window, provide information for your RDS for Db2 DB instance.

   a. Enter the following information:

      - For **Host**, enter the DNS name of the DB instance.
      - For **Port**, enter the port number for the DB instance.
      - For **Database**, enter the name of the database.
      - For **Username**, enter the name of the database administrator for the DB instance.
      - For **Password**, enter the password of the database administrator for the DB instance.

   b. Select **Save password**.

   c. Choose **Driver Settings**.



4. In the **Edit Driver** window, specify additional security properties.

   a. Choose the **Driver properties** tab.

   b. Add two **User Properties**.

      i. Open the context (right-click) menu, and then choose **Add new property**.

      ii. For **Property Name**, add **encryptionAlgorithm**, and then choose **OK**.

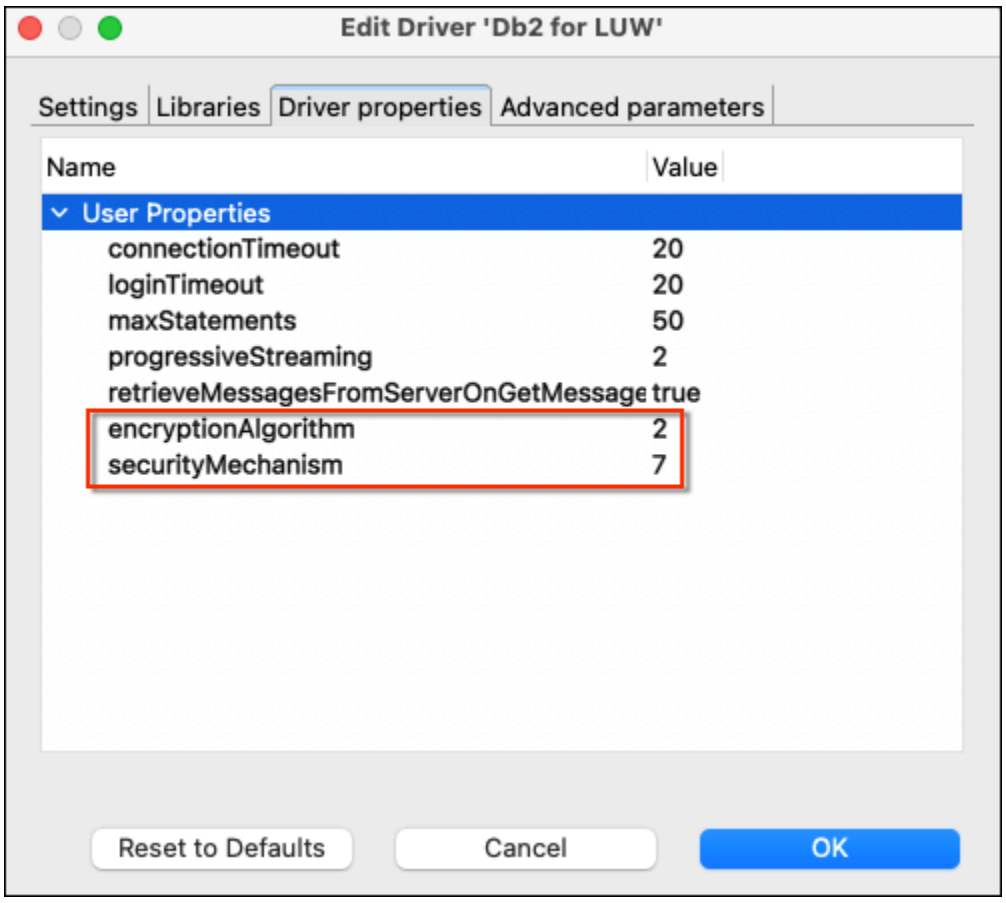
      iii. With the **encryptionAlgorithm** row selected, choose the **Value** column and add **2**.

      iv. Open the context (right-click) menu, and then choose **Add new property**.

      v. For **Property Name**, add **securityMechanism**, and then choose **OK**.

      vi. With the **securityMechanism** row selected, choose the **Value** column and add **7**.

   c. Choose **OK**.

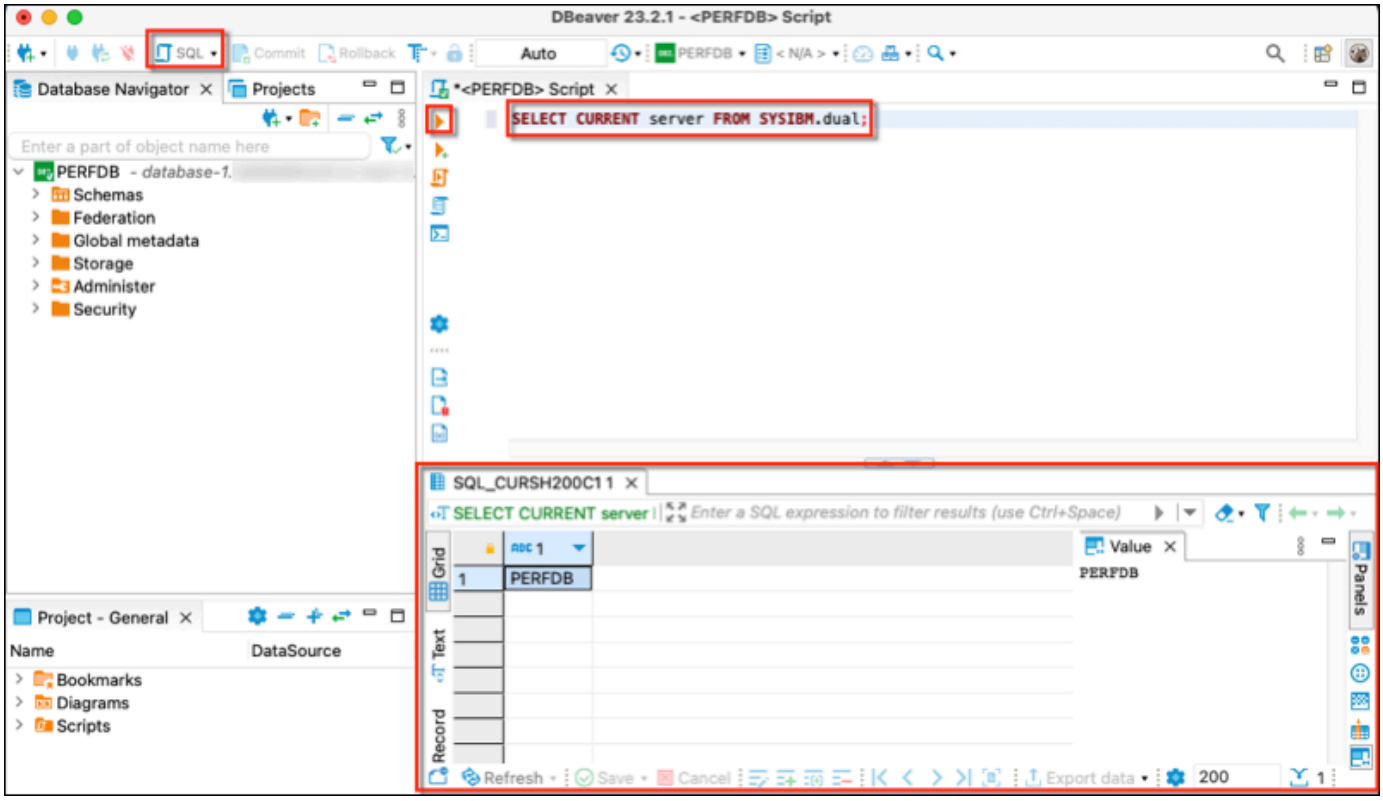


5. In the **Connect to a database** window, choose **Test Connection**. If you don't have a DB2 JBDC driver installed on your computer, then the driver automatically downloads.
6. Choose **OK**.
7. Choose **Finish**.
8. In the **Database Navigation** tab, choose the name of the database. You can now explore objects.

You are now ready to run SQL commands.

To run SQL commands and view the results

1. In the top menu, choose **SQL**. This opens a SQL script panel.
2. In the **Script** panel, enter a SQL command.
3. To run the command, choose the **Execute SQL query** button.
4. In the SQL results panel, view the results of your SQL queries.



Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 Data Management Console

You can connect to your Amazon RDS for Db2 DB instance with IBM Db2 Data Management Console. IBM Db2 Data Management Console can administer and monitor several RDS for Db2 DB instances.

> **Note**
>
> You must have an Amazon EC2 Linux or Windows machine that is on the same VPC and security group as your RDS for Db2 DB instance. The VPC and security group controls the connection to your DB instance through the internal network.

IBM Db2 Data Management Console requires a repository Db2 database to store metadata and performance metrics but can't automatically create a repository database for RDS for Db2. Instead, you must first create a repository database to monitor one or more RDS for Db2 DB instances. Then you can install IBM Db2 Data Management Console and connect to your RDS for Db2 DB instance with IBM Db2 Data Management Console.

Topics

- Step 1: Creating a repository database to monitor DB instances
- Step 2: Installing and setting up IBM Db2 Data Management Console
- Step 3: Configuring the repository database and connecting to RDS for Db2 DB instances
- Using IBM Db2 Data Management Console

Step 1: Creating a repository database to monitor DB instances

You can use an existing properly sized RDS for Db2 DB instance as a repository for IBM Db2 Data Management Console to monitor other RDS for Db2 DB instances. However, because the admin user doesn't have SYSCTRL authority to create buffer pools and tablespaces, using IBM Db2 Data Management Console repository creation to create a repository database fails. Instead, you must create a repository database. This repository database monitors your RDS for Db2 DB instances.

You can create a repository database in two different ways. You can create an RDS for Db2 database and then manually create a buffer pool, a user tablespace, and a system temporary tablespace. Or, you can create a separate Amazon EC2 instance to host an IBM Db2 Data Management Console repository database.

Topics

- Manually creating a buffer pool, a user tablespace, and a system temporary tablespace
- Creating an Amazon EC2 instance to host an IBM Db2 Data Management Console repository

Manually creating a buffer pool, a user tablespace, and a system temporary tablespace

To create a buffer pool, a user tablespace, and a system temporary tablespace

1. Connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 connect to rdsadmin user master_username using master_password
   ```

2. Create a buffer pool for IBM Db2 Data Management Console. In the following example, replace *`database_name`* with the name of the repository you created for IBM Db2 Data Management Console to monitor your RDS for Db2 DB instances.

   ```
   db2 "call rdsadmin.create_bufferpool('database_name',
   ```

```
     'BP4CONSOLE', 1000, 'Y', 'Y', 32768)"
```

3. Create a user tablespace for IBM Db2 Data Management Console. In the following example, replace *database_name* with the name of the repository you created for IBM Db2 Data Management Console to monitor your RDS for Db2 DB instances.

```
db2 "call rdsadmin.create_tablespace('database_name',
     'TS4CONSOLE', 'BP4CONSOLE', 32768)"
```

4. Create a system temporary tablespace for IBM Db2 Data Management Console. In the following example, replace *database_name* with the name of the repository you created for IBM Db2 Data Management Console to monitor your RDS for Db2 DB instances.

```
db2 "call rdsadmin.create_tablespace('database_name',
    'TS4CONSOLE_TEMP', 'BP4CONSOLE', 32768, 0, 0, 'S')"
```

You are now ready to install IBM Db2 Data Management Console. For more information about installation and setup, see Step 2: Installing and setting up IBM Db2 Data Management Console.

Creating an Amazon EC2 instance to host an IBM Db2 Data Management Console repository

You can create a separate Amazon Elastic Compute Cloud (Amazon EC2) instance to host an IBM Db2 Data Management Console repository. For information about creating an Amazon EC2 instance, see Tutorial: Get started with Amazon EC2 Linux instances in the *Amazon EC2 User Guide*.

Step 2: Installing and setting up IBM Db2 Data Management Console

After you create a buffer pool, a user tablespace, and a system temporary tablespace, you are ready to install and set up IBM Db2 Data Management Console.

> **Important**
>
> You must have an Amazon EC2 Linux or Windows machine that is on the same VPC and security group as your RDS for Db2 DB instance. The VPC and security group controls the connection to your DB instance through the internal network. Also, you must have already created a repository database for IBM Db2 Data Management Console.

To install and set up IBM Db2 Data Management Console

1. Download IBM Db2 Data Management Console from IBM Db2 Data Management Console Version 3.1x releases on the IBM Support website.
2. Install IBM Db2 Data Management Console.
3. Open IBM Db2 Data Management Console and use the IP address of your Amazon EC2 machine and the port number you used for the HTTP or HTTPS connection to your Amazon EC2 instance. For example, use `http://xx.xx.xx.xx:11080` or `https://xx.xx.xx.xx.11081`. Replace *xx.xx.xx.xx* with the IP address of your Amazon EC2 machine. `11080` and `11081` are the default ports for HTTP and HTTPS connections.
4. (Optional) If you want to use port 80 or 443 on your Amazon EC2 instance, you can use either Apache httpd or a Nginx HTTP server to proxy the IBM Db2 Data Management Console port to either port 80 or 443. For more information, see Apache HTTP Server Project and the nginx website.

   To allow connection to IBM Db2 Data Management Console, you must edit the inbound rules in your security group. If you use a proxy, change the TCP/IP port 80 or 443 to redirect to the IBM Db2 Data Management Console ports. If you aren't using a proxy, change the TCP/IP port 80 or 443 to the default ports 11080 (HTTP) or 11081 (HTTPS).

You are now ready to log in to IBM Db2 Data Management Console to configure the repository database and to connect to your RDS for Db2 DB instances. For more information, see Configuring the repository database and connecting to DB instances.

Step 3: Configuring the repository database and connecting to RDS for Db2 DB instances

When you connect to the repository database for the first time, IBM Db2 Data Management Console automatically configures the repository. After the repository database is configured, you can add database connections to IBM Db2 Data Management Console.

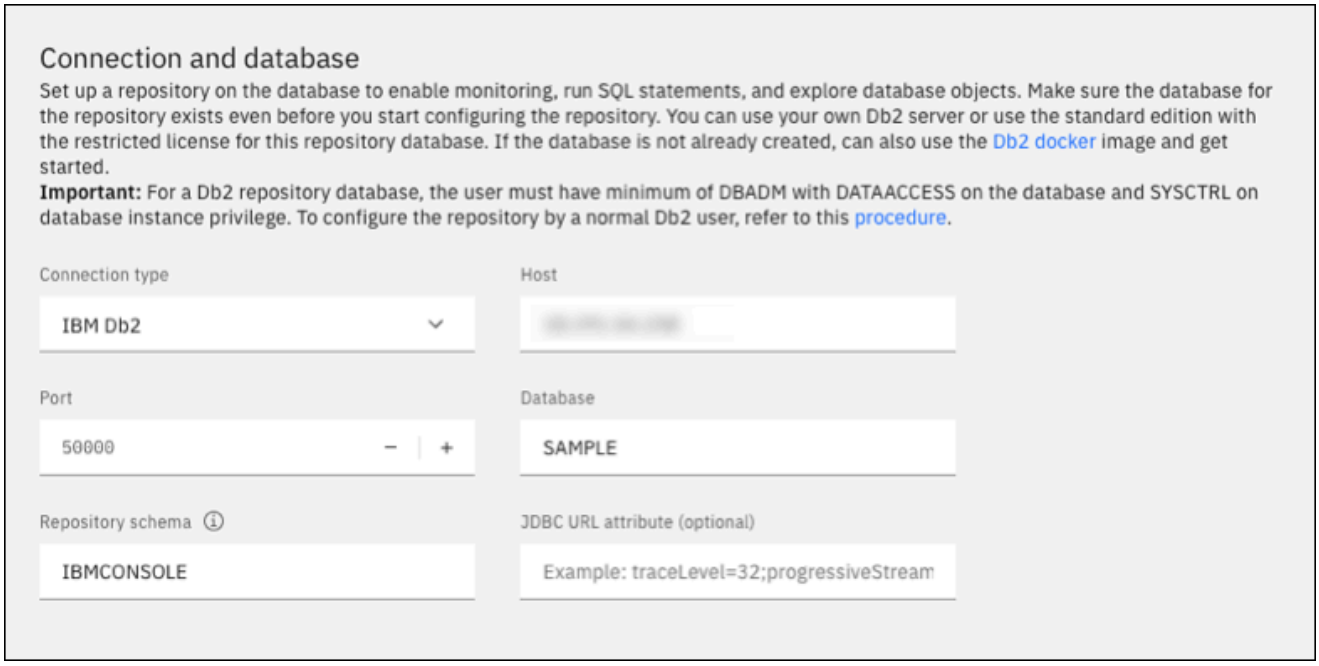
To connect to your RDS for Db2 DB instance, you need its DNS name and port number. For information about finding them, see Finding the endpoint. You also need to know the database name, master username, and master password that you defined when you created your RDS for Db2 DB instance. For more information about finding them, see Creating a DB instance. If you are connecting over the internet, allow traffic to the database port. For more information, see Creating a DB instance.

To connect to RDS for Db2 DB instances with IBM Db2 Data Management Console

1. Log in to IBM Db2 Data Management Console with the credentials you set during installation.
2. Configure the repository.

   a. In the **Connection and database** section, enter the following information for your RDS for Db2 DB instance:

   - For **Host**, enter the DNS name of the DB instance.
   - For **Port**, enter the port number for the DB instance.
   - For **Database**, enter the name of the database.



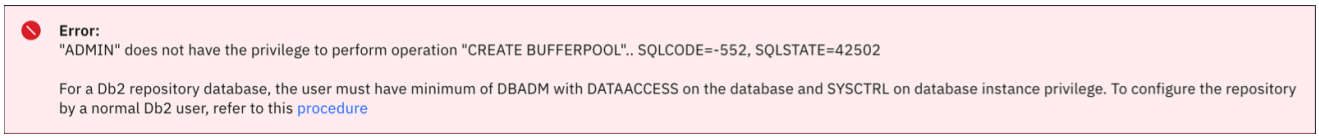
   b. In the **Security and credential** section, enter the following information for your RDS for Db2 DB instance:

   - For **Security type**, choose **Encrypted user and password**.
   - For **Username**, enter the name of the database administrator for the DB instance.
   - For **Password**, enter the password of the database administrator for the DB instance.

   c. Choose **Test connection**.

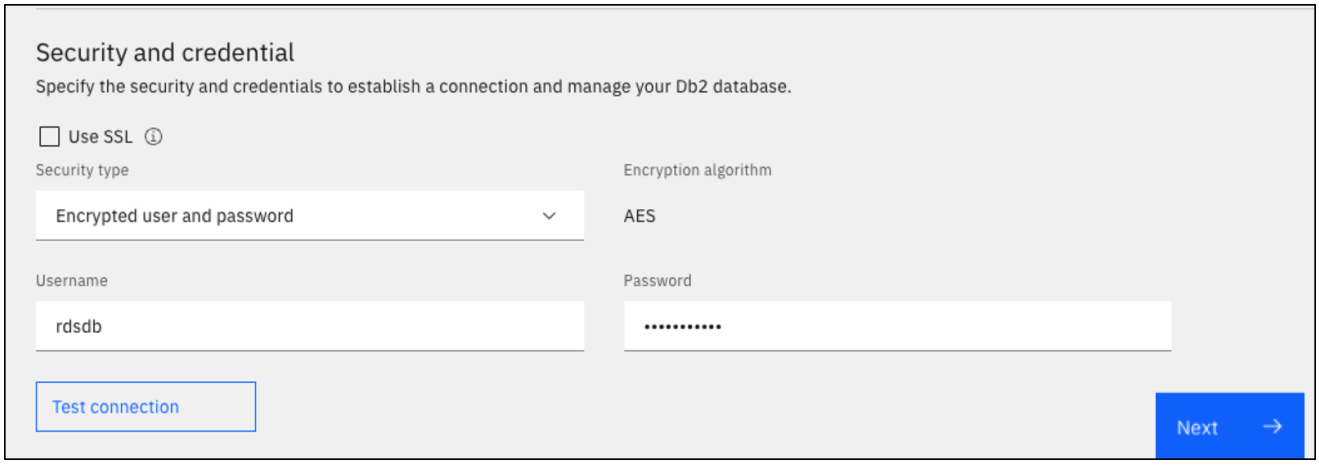
> **Note**
>
> If the connection is unsuccessful, confirm that the database port is open through the inbound rules in your security group. For more information, see Considerations for security groups with Amazon RDS for Db2.

If you didn't manually create a buffer pool, a user tablespace, and a system temporary tablespace in RDS for Db2, you might see the following error message:



Make sure that you created a buffer table, a tablespace, and objects for an IBM Db2 Data Management Console repository to monitor your RDS for Db2 DB instance. Or, you can use an Amazon EC2 Db2 DB instance to host an IBM Db2 Data Management Console repository to monitor your RDS for Db2 DB instance. For more information, see Step 1: Creating a repository database to monitor DB instances.

d. After you successfully test your connection, choose **Next**.

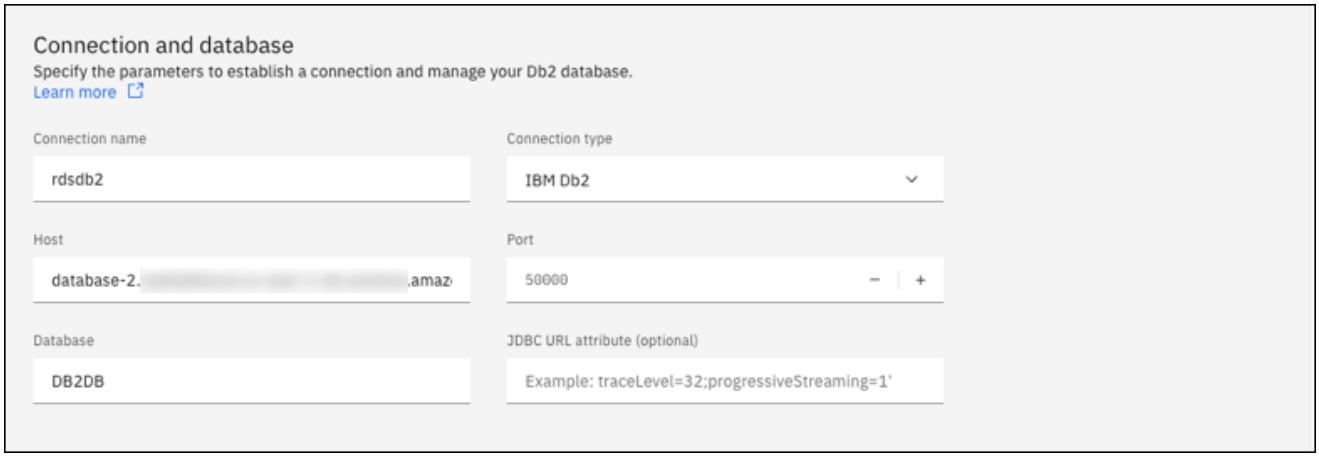


If IBM Db2 Data Management Console finds the buffer pool, the user tablespace, and the system temporary tablespace in the RDS for Db2 DB instance, then IBM Db2 Data Management Console automatically configures the repository database. If you use your Db2 instance on your Amazon EC2 instance as the repository database, then IBM Db2 Data Management Console automatically creates the buffer pool and other objects.

3. In the **Set statistics event monitor opt-in** window, choose **Next**.
4. (Optional) Add new connection. If you want to use a different RDS for Db2 DB instance for administration and monitoring, then add a connection to a non-repository RDS for Db2 DB instance.

   a. In the **Connection and database** section, enter the following information for the RDS for Db2 DB instance to use for administration and monitoring:

      - For **Connection name**, enter the Db2 database identifier.
      - For **Host**, enter the DNS name of the DB instance.
      - For **Port**, enter the port number for the DB instance.
      - For **Database**, enter the name of the database.



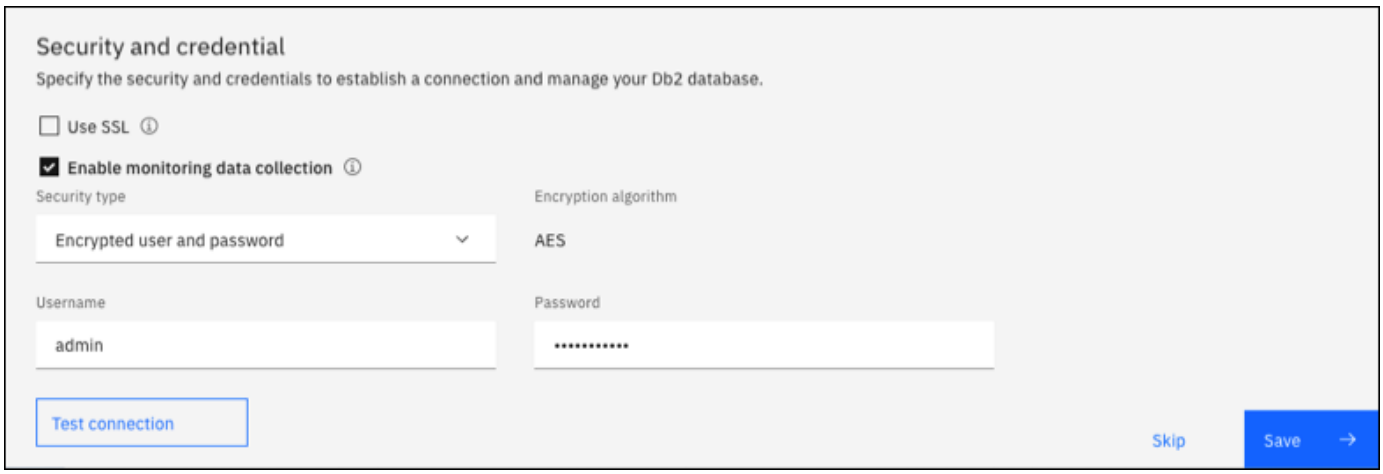
   b. In the **Security and credential** section, select **Enable monitoring data collection**.

   c. Enter the following information for your RDS for Db2 DB instance:

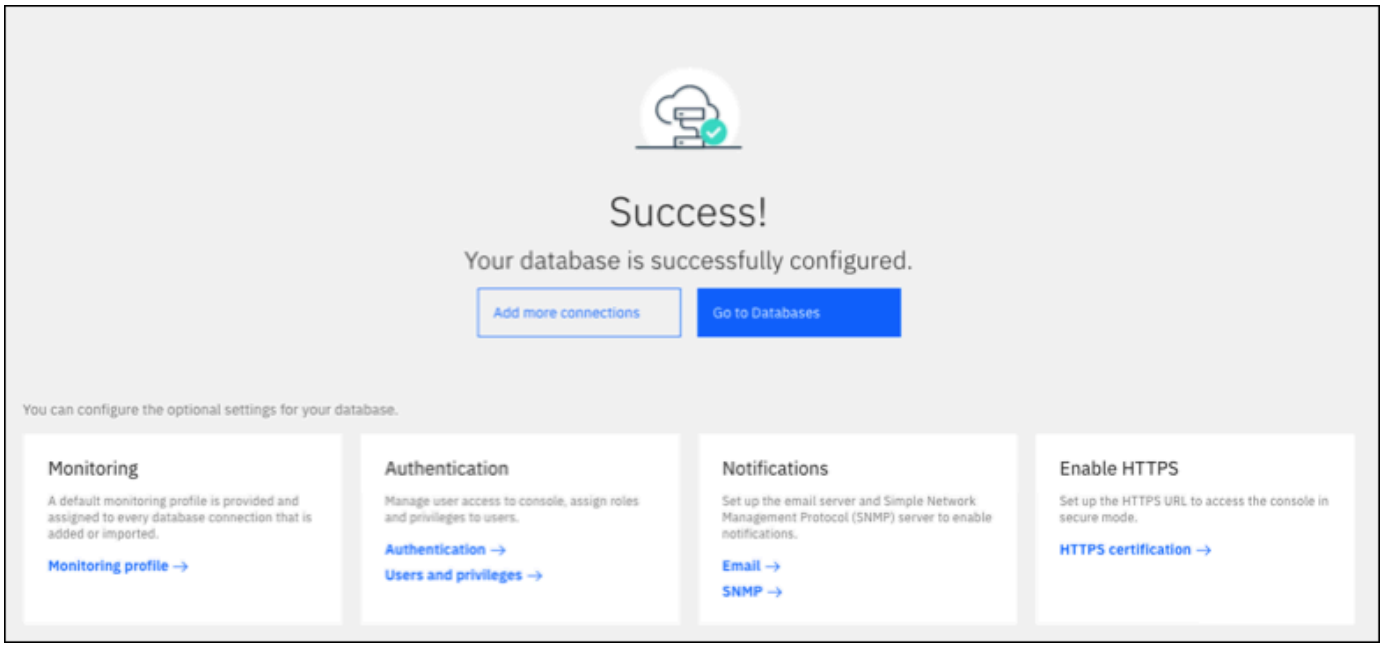
      - For **Username**, enter the name of the database administrator for the DB instance.
      - For **Password**, enter the password of the database administrator for the DB instance.

   d. Choose **Test connection**.

   e. After you successfully test your connection, choose **Save**.



After the connection is added, a window similar to the following appears. This window indicates that your database was successfully configured.



5. Choose **Go to Databases**. A Databases window similar to the following appears. This window is a dashboard that shows metrics, statuses, and connections.



You can now start using IBM Db2 Data Management Console.

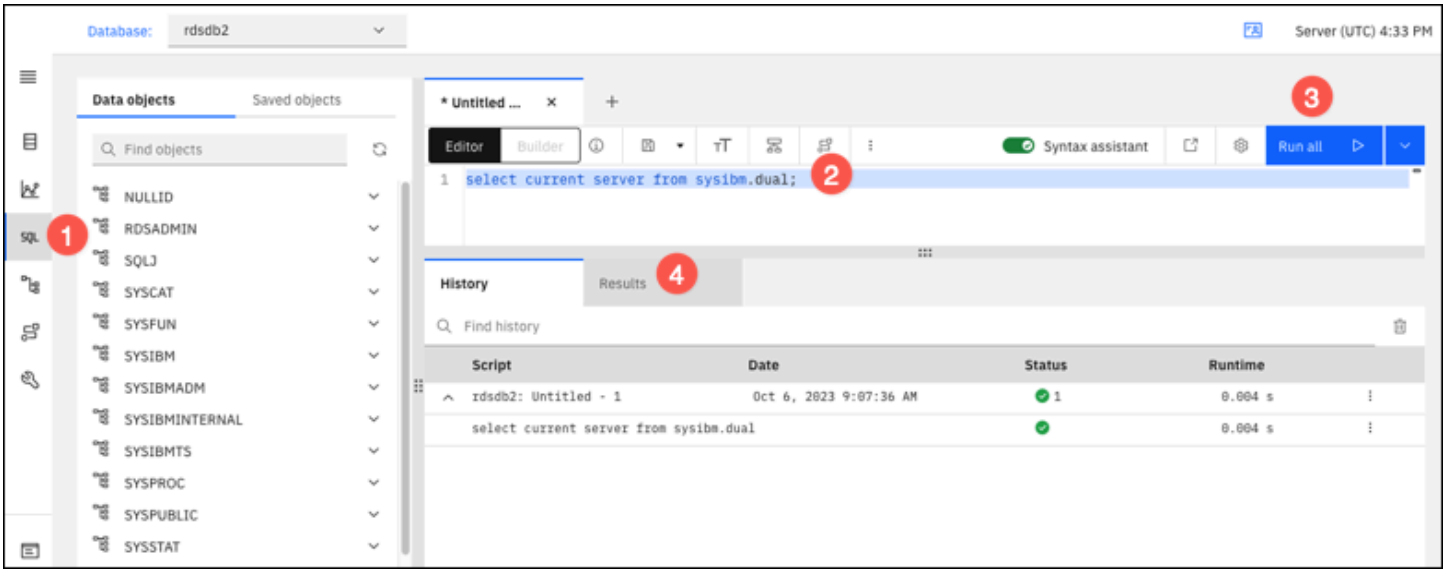
Using IBM Db2 Data Management Console

You can use IBM Db2 Data Management Console to do the following types of tasks:

- Manage multiple RDS for Db2 DB instances.
- Run SQL commands.
- Explore, create, or change data and database objects.
- Create `EXPLAIN PLAN` statements in SQL.
- Tune queries.

To run SQL commands and view the results

1. In the left navigation bar, choose **SQL**.
2. Enter a SQL command.
3. Choose **Run all**.
4. To view the results, choose the **Results** tab.



Considerations for security groups with Amazon RDS for Db2

For you to connect to your Amazon RDS for Db2 DB instance, it must be associated with a security group that contains the necessary IP addresses and network configuration. Your RDS for Db2 DB instance might use the default security group. If you assigned a default nonconfigured security group when you created the RDS for Db2 DB instance, then the firewall prevents internet connections. For information about creating a new security group, see Controlling access with security groups.

After you create the new security group, you modify your DB instance to associate it with the security group. For more information, see Modifying an Amazon RDS DB instance.

You can enhance security by using SSL to encrypt connections to your DB instance. For more information, see Using SSL/TLS with an Amazon RDS for Db2 DB instance.

Securing Amazon RDS for Db2 DB instance connections

Amazon RDS for Db2 supports ways to improve security for your RDS for Db2 DB instance.

Topics

- Using SSL/TLS with an Amazon RDS for Db2 DB instance
- Using Kerberos authentication for Amazon RDS for Db2

Using SSL/TLS with an Amazon RDS for Db2 DB instance

SSL is an industry-standard protocol for securing network connections between client and server. After SSL version 3.0, the name was changed to TLS, but we still often refer to the protocol as SSL. Amazon RDS supports SSL encryption for Amazon RDS for Db2 DB instances. Using SSL/TLS, you can encrypt a connection between your application client and your RDS for Db2 DB instance. SSL/TLS support is available in all AWS Regions for RDS for Db2.

To enable SSL/TLS encryption for an RDS for Db2 DB instance, add the Db2 SSL option to the parameter group associated with the DB instance. Amazon RDS uses a second port, as required by Db2, for SSL/TLS connections. Doing this allows both clear text and SSL-encrypted communication to occur at the same time between a DB instance and a Db2 client. For example, you can use the port with clear text communication to communicate with other resources inside a VPC while using the port with SSL-encrypted communication to communicate with resources outside the VPC.

Topics

- Creating an SSL/TLS connection
- Connect to your Db2 database server

Creating an SSL/TLS connection

To create an SSL/TLS connection, choose a certificate authority (CA), download a certificate bundle for all AWS Regions, and add parameters to a custom parameter group.

Step 1: Choose a CA and download a certificate

Choose a certificate authority (CA) and download a certificate bundle for all AWS Regions. For more information, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

Step 2: Update parameters in a custom parameter group

> **Important**
>
> If you're using the bring your own license (BYOL) model for RDS for Db2, modify the custom parameter group that you created for your IBM Customer ID and your IBM Site ID. If you're using a different licensing model for RDS for Db2, then follow the procedure to add parameters to a custom parameter group. For more information, see Amazon RDS for Db2 licensing options.

You can't modify default parameter groups for RDS for Db2 DB instances. Therefore, you must create a custom parameter group, modify it, and then attach it to your RDS for Db2 DB instances. For information about parameter groups, see DB parameter groups for Amazon RDS DB instances.

Use the parameter settings in the following table.

Parameter	Value
`DB2COMM`	`TCPIP,SSL` or `SSL`
`SSL_SVCENAME`	`<any port number except the number used for the non-SSL port>`

To update parameters in a custom parameter group

1. Create a custom parameter group by running the create-db-parameter-group command.

   Include the following required options:

   - `--db-parameter-group-name` – A name for the parameter group that you are creating.
   - `--db-parameter-group-family` – The Db2 engine edition and major version. Valid values: `db2-se-11-5`, `db2-ae-11.5`.
   - `--description` – A description for this parameter group.

   For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.

2. Modify the parameters in the custom parameter group that you created by running the modify-db-parameter-group command.

   Include the following required options:

   - `--db-parameter-group-name` – The name of the parameter group that you created.
   - `--parameters` – An array of parameter names, values, and the application methods for the parameter update.

   For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.
3. Associate the parameter group with your RDS for Db2 DB instance. For more information, see Associating a DB parameter group with a DB instance in Amazon RDS.

Connect to your Db2 database server

Instructions for connecting to your Db2 database server are language-specific.

Java

To connect to your Db2 database server using Java

1. Download the JDBC driver. For more information, see DB2 JDBC Driver Versions and Downloadsin the IBM Support documentation.
2. Create a shell script file with the following content. This script adds all certificates from the bundle to a Java KeyStore.

   Important

   Verify that `keytool` exists on the path in the script so that the script can locate it. If you use a Db2 client, you can locate the `keytool` under `~sqlib/java/jdk64/jre/bin`.

```
#!/bin/bash
PEM_FILE=$1
PASSWORD=$2
KEYSTORE=$3
```

```
# number of certs in the PEM file
CERTS=$(grep 'END CERTIFICATE' $PEM_FILE| wc -l)
for N in $(seq 0 $(($CERTS - 1))); do
    ALIAS="${PEM_FILE%.*}-$N"
    cat $PEM_FILE |
    awk "n==$N { print }; /END CERTIFICATE/ { n++ }" |
    keytool -noprompt -import -trustcacerts -alias $ALIAS -keystore $KEYSTORE -
storepass $PASSWORD
done
```

3. To run the shell script and import the PEM file with the certificate bundle into a Java KeyStore, run the following command. Replace *shell_file_name.sh* with the name of your shell script file and *password* with the password for your Java KeyStore.

```
./shell_file_name.sh global-bundle.pem password truststore.jks
```

4. To connect to your Db2 server, run the following command. Replace the following placeholders in the example with your RDS for Db2 DB instance information.

   - *ip_address* – The IP address for your DB instance endpoint.
   - *port* – The port number for the SSL connection. This can be any port number except the number that's used for the non-SSL port.
   - *database_name* – The name of your database in your DB instance.
   - *master_username* – The master username for your DB instance.
   - *master_password* – The master password for your DB instance.

```
export trustStorePassword=MyPassword
java -cp ~/dsdriver/jdbc_sqlj_driver/linuxamd64/db2jcc4.jar \
com.ibm.db2.jcc.DB2Jcc -url \
"jdbc:db2://ip_address:port/database_name:\
sslConnection=true;sslTrustStoreLocation=\
~/truststore.jks;\
sslTrustStorePassword=${trustStorePassword};\
sslVersion=TLSv1.2;\
encryptionAlgorithm=2;\
securityMechanism=7;" \
-user master_username -password master_password
```

Node.js

To connect to your Db2 database server using Node.js

1. Install the **node-ibm_db** driver. For more information, see Installing the node-ibm_db driver on Linux and UNIX systems in the IBM Db2 documentation.
2. Create a JavaScript file based on the following content. Replace the following placeholders in the example with your RDS for Db2 DB instance information.

   - *ip_address* – The IP address for your DB instance endpoint.
   - *master_username* – The master username for your DB instance.
   - *master_password* – The master password for your DB instance.
   - *database_name* – The name of your database in your DB instance.
   - *port* – The port number for the SSL connection. This can be any port number except the number that's used for the non-SSL port.

```
var ibmdb = require("ibm_db");
const hostname = "ip_address";
const username = "master_username";
const password = "master_password";
const database = "database_name";
const port = "port";
const certPath = "/root/qa-bundle.pem";
ibmdb.open("DRIVER={DB2};DATABASE=" + database + ";HOSTNAME=" +
 hostname + ";UID=" + username + ";PWD=" + password + ";PORT=" + port +
 ";PROTOCOL=TCPIP;SECURITY=SSL;SSLServerCertificate=" + certPath + ";", function
 (err, conn){
 if (err) return console.log(err);
 conn.close(function () {
 console.log('done');
 });
});
```

3. To run the JavaScript file, run the following command.

```
node ssl-test.js
```

Python

To connect to your Db2 database server using Python

1. Create a Python file with the following content. Replace the following placeholders in the example with your RDS for Db2 DB instance information.

   - *port* – The port number for the SSL connection. This can be any port number except the number that's used for the non-SSL port.
   - *master_username* – The master username for your DB instance.
   - *master_password* – The master password for your DB instance.
   - *database_name* – The name of your database in your DB instance.
   - *ip_address* – The IP address for your DB instance endpoint.

```
import click
import ibm_db
import sys

port = port;
master_user_id = "master_username" # Master id used to create your DB instance
master_password = "master_password" # Master password used to create your DB
 instance
db_name = "database_name" # If not given "db-name'
vpc_customer_private_ip = "ip_address" # Hosts end points - Customer private IP
 Addressicert_path = "/root/ssl/global-bundle.pem" # cert path

@click.command()
@click.option("--path", help="certificate path")
def db2_connect(path):

    try:
        conn =
 ibm_db.connect(f"DATABASE={db_name};HOSTNAME={vpc_customer_private_ip};PORT={port};

 PROTOCOL=TCPIP;UID={master_user_id};PWD={master_password};SECURITY=ssl;SSLServerCertifi
 "", "")
        try:
            ibm_db.exec_immediate(conn, 'create table tablename (a int);')
            print("Query executed successfully")
        except Exception as e:
```

```
            print(e)
        finally:
            ibm_db.close(conn)
            sys.exit(1)
    except Exception as ex:
        print("Trying to connect...")

if __name__ == "__main__":
    db2_connect()
```

2. Create the following shell script, which runs the Python file you created. Replace *python_file_name.py* with the name of your Python script file.

```
#!/bin/bash
PEM_FILE=$1
# number of certs in the PEM file
CERTS=$(grep 'END CERTIFICATE' $PEM_FILE| wc -l)

for N in $(seq 0 $(($CERTS - 1))); do
    ALIAS="${PEM_FILE%.*}-$N"
    cert=`cat $PEM_FILE | awk "n==$N { print }; /END CERTIFICATE/ { n++ }"`
    cat $PEM_FILE | awk "n==$N { print }; /END CERTIFICATE/ { n++ }" >
 $ALIAS.pem
    python3 <python_file_name.py> --path $ALIAS.pem
    output=`echo $?`
    if [ $output == 1 ]; then
        break
    fi
done
```

3. To import the PEM file with the certificate bundle and run the shell script, run the following command. Replace *shell_file_name.sh* with the name of your shell script file.

```
./shell_file_name.sh global-bundle.pem
```

Db2 CLP

To connect to your Db2 database server using Db2 CLP

1. To connect to your Db2 instance using Db2 CLP, you need GSKit, which you can download at IBM Fix Central. To use Db2 CLP, you also need the IBM Db2 client, which you can download from Download initial Version 11.5 clients and drivers in IBM Support.
2. Create a keystore.

```
gsk8capicmd_64 -keydb -create -db "directory/keystore-filename" -pw
 "changeThisPassword" -type pkcs12 -stash
```

3. Import the certificate bundles to the keystore.

```
gsk8capicmd_64 -cert -import -file global-bundle.pem -target directory/keystore-
filename> -target_stashed
```

4. Update the Db2 instance configuration.

```
db2 update dbm cfg using SSL_CLNT_KEYDB keystore-filename
 SSL_CLNT_STASH keystore stash file immediate
```

5. Catalog the node and database.

```
db2 catalog tcpip node ssluse1 REMOTE endpoint SERVER ssl_svcename security ssl

db2 catalog database testdb as ssltest at node ssluse1
```

6. Connect to the database.

```
db2 connect to ssltest user username using password
```

Using Kerberos authentication for Amazon RDS for Db2

You can use Kerberos authentication to authenticate users when they connect to your Amazon RDS for Db2 DB instance. In this configuration, your DB instance works with AWS Directory Service for Microsoft Active Directory, also called AWS Managed Microsoft AD. You add the domain and other information of your AWS Managed Microsoft AD directory to your RDS for Db2 DB instance. When users authenticate with an RDS for Db2 DB instance joined to the trusting domain,

authentication requests are forwarded to the AWS Managed Microsoft AD directory that you created with Directory Service.

Keeping all of your credentials in the same directory can save you time and effort. With this approach, you have a centralized place for storing and managing credentials for multiple DB instances. Using a directory can also improve your overall security profile.

In addition, you can access credentials from your own on-premises Microsoft Active Directory. To do so, create a trusting domain relationship so that the AWS Managed Microsoft AD directory trusts your on-premises Microsoft Active Directory. In this way, your users can access your RDS for Db2 DB instances with the same Windows single sign-on (SSO) experience as when they access workloads in your on-premises network.

For more information, see What is Directory Service? in the *AWS Directory Service Administration Guide*.

For information about Kerberos authentication, see the following topics:

Topics

- Setting up Kerberos authentication for Amazon RDS for Db2 DB instances
- Connecting to Amazon RDS for Db2 with Kerberos authentication

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information about version and Region availability of RDS for Db2 with Kerberos authentication, see Supported Regions and DB engines for Kerberos authentication in Amazon RDS.

> **Note**
>
> Kerberos authentication isn't supported for DB instance classes that are deprecated for RDS for Db2 DB instances. For more information, see Amazon RDS for Db2 instance classes.

Overview of Kerberos authentication for RDS for Db2 DB instances

To set up Kerberos authentication for an RDS for Db2 DB instance, complete the following general steps, which are described in more detail later:

1. Use AWS Managed Microsoft AD to create an AWS Managed Microsoft AD directory. You can use the AWS Management Console, the AWS Command Line Interface (AWS CLI), or Directory Service to create the directory. For more information, see Create your AWS Managed Microsoft AD directory in the *AWS Directory Service Administration Guide*.
2. Create an AWS Identity and Access Management (IAM) role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`. The IAM role allows Amazon RDS to make calls to your directory.

   For the IAM role to allow access, the AWS Security Token Service (AWS STS) endpoint must be activated in the correct AWS Region for your AWS account. AWS STS endpoints are active by default in all AWS Regions, and you can use them without any further actions. For more information, see Activating and deactivating AWS STS in an AWS Region in the *IAM User Guide*.
3. Create or modify an RDS for Db2 DB instance by using the AWS Management Console, the AWS CLI, or the RDS API with one of the following methods:
   - Create a new RDS for Db2 DB instance using the console, the create-db-instance command, or the CreateDBInstance API operation. For instructions, see Creating an Amazon RDS DB instance.
   - Modify an existing RDS for Db2 DB instance using the console, the modify-db-instance command, or the ModifyDBInstance API operation. For instructions, see Modifying an Amazon RDS DB instance.
   - Restore an RDS for Db2 DB instance from a DB snapshot using the console, the restore-db-instance-from-db-snapshot command, or the RestoreDBInstanceFromDBSnapshot API operation. For instructions, see Restoring to a DB instance.
   - Restore an RDS for Db2 DB instance to a point-in-time using the console, the restore-db-instance-to-point-in-time command, or the RestoreDBInstanceToPointInTime API operation. For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

   You can locate the DB instance in the same Amazon Virtual Private Cloud (VPC) as the directory or in a different AWS account or VPC. When you create or modify the RDS for Db2 DB instance, do the following tasks:
   - Provide the domain identifier (`d-*` identifier) that was generated when you created your directory.
   - Provide the name of the IAM role that you created.
   - Verify that the DB instance security group can receive inbound traffic from the directory security group.

4. Configure your Db2 client, and verify that traffic can flow between the client host and Directory Service for the following ports:
   - TCP/UDP port 53 – DNS
   - TCP 88 – Kerberos authentication
   - TCP 389 – LDAP
   - TCP 464 – Kerberos authentication

Managing a DB instance in a domain

You can use the AWS Management Console, the AWS CLI, or the RDS API to manage your DB instance and its relationship with your Microsoft Active Directory. For example, you can associate an Active Directory to enable Kerberos authentication. You can also remove the association for an Active Directory to disable Kerberos authentication. You can also move a DB instance to be externally authenticated by one Microsoft Active Directory to another.

For example, running the modify-db-instance CLI command, you can perform the following actions:

- Re-attempt enabling Kerberos authentication for a failed membership by specifying the current membership's directory ID for the `--domain` option.
- Disable Kerberos authentication on a DB instance by specifying `none` for the `--domain` option.
- Move a DB instance from one domain to another by specifying the domain identifier of the new domain for the `--domain` option.

Understanding domain membership

After you create or modify your DB instance, it becomes a member of the domain. You can view the status of the domain membership in the console or by running the describe-db-instances command. The status of the DB instance can be one of the following:

- `kerberos-enabled` – The DB instance has Kerberos authentication enabled.
- `enabling-kerberos` – AWS is in the process of enabling Kerberos authentication on this DB instance.
- `pending-enable-kerberos` – Enabling Kerberos authentication is pending on this DB instance.

- `pending-maintenance-enable-kerberos` – AWS will attempt to enable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `pending-disable-kerberos` – Disabling Kerberos authentication is pending on this DB instance.
- `pending-maintenance-disable-kerberos` – AWS will attempt to disable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `enable-kerberos-failed` – A configuration problem prevented AWS from enabling Kerberos authentication on the DB instance. Correct the configuration problem before re-issuing the command to modify the DB instance.
- `disabling-kerberos` – AWS is in the process of disabling Kerberos authentication on this DB instance.

A request to enable Kerberos authentication can fail because of a network connectivity issue or an incorrect IAM role. In some cases, the attempt to enable Kerberos authentication might fail when you create or modify a DB instance. If this happens, verify that you are using the correct IAM role, and then modify the DB instance to join the domain.

Setting up Kerberos authentication for Amazon RDS for Db2 DB instances

You use AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) to set up Kerberos authentication for an RDS for Db2 DB instance. To set up Kerberos authentication, follow these steps:

Topics

- Step 1: Create a directory using AWS Managed Microsoft AD
- Step 2: Create a trust
- Step 3: Create an IAM role for Amazon RDS to access Directory Service
- Step 4: Create and configure users
- Step 5: Create an RDS for Db2 admin group in AWS Managed Microsoft AD
- Step 6: Modify DB parameter
- Step 7: Create or modify an RDS for Db2 DB instance
- Step 8: Retrieve the Active Directory group SID in PowerShell
- Step 9: Add SID to GroupName mappings to your RDS for Db2 DB instance

- Step 10: Configure a Db2 client

Step 1: Create a directory using AWS Managed Microsoft AD

Directory Service creates a fully managed Active Directory in the AWS Cloud. When you create an AWS Managed Microsoft AD directory, Directory Service creates two domain controllers and DNS servers for you. The directory servers are created in different subnets in a VPC. This redundancy helps ensure that your directory remains accessible even if a failure occurs.

When you create an AWS Managed Microsoft AD directory, Directory Service performs the following tasks on your behalf:

- Sets up an Active Directory within your VPC.
- Creates a directory administrator account with the username `Admin` and the specified password. You use this account to manage your directory.

  > **Important**
  >
  > Make sure to save this password. Directory Service doesn't store this password, and it can't be retrieved or reset.

- Creates a security group for the directory controllers. The security group must permit communication with the RDS for Db2 DB instance.

When you launch AWS Directory Service for Microsoft Active Directory, AWS creates an organizational unit (OU) that contains all of your directory's objects. This OU, which has the NetBIOS name that you entered when you created your directory, is located in the domain root. The domain root is owned and managed by AWS.

The `Admin` account that was created with your AWS Managed Microsoft AD directory has permissions for the most common administrative activities for your OU:

- Create, update, or delete users.
- Add resources to your domain such as file or print servers, and then assign permissions for those resources to users in your OU.
- Create additional OUs and containers.
- Delegate authority.

- Restore deleted objects from the Active Directory Recycle Bin.
- Run Active Directory and Domain Name Service (DNS) modules for Windows PowerShell on the Directory Service.

The `Admin` account also has rights to perform the following domain-wide activities:

- Manage DNS configurations (add, remove, or update records, zones, and forwarders).
- View DNS event logs.
- View security event logs.

To create a directory with AWS Managed Microsoft AD

1. Sign in to the AWS Management Console and open the Directory Service console at https://console.aws.amazon.com/directoryservicev2/.
2. Choose **Set up directory**.
3. Choose **AWS Managed Microsoft AD**. AWS Managed Microsoft AD is the only option currently supported for use with Amazon RDS.
4. Choose **Next**.
5. On the **Enter directory information** page, provide the following information:

   - **Edition** – Choose the edition that meets your requirements.
   - **Directory DNS name** – The fully qualified name for the directory, such as `corp.example.com`.
   - **Directory NetBIOS name** – An optional short name for the directory, such as `CORP`.
   - **Directory description** – An optional description for the directory.
   - **Admin password** – The password for the directory administrator. The directory creation process creates an administrator account with the username `Admin` and this password.

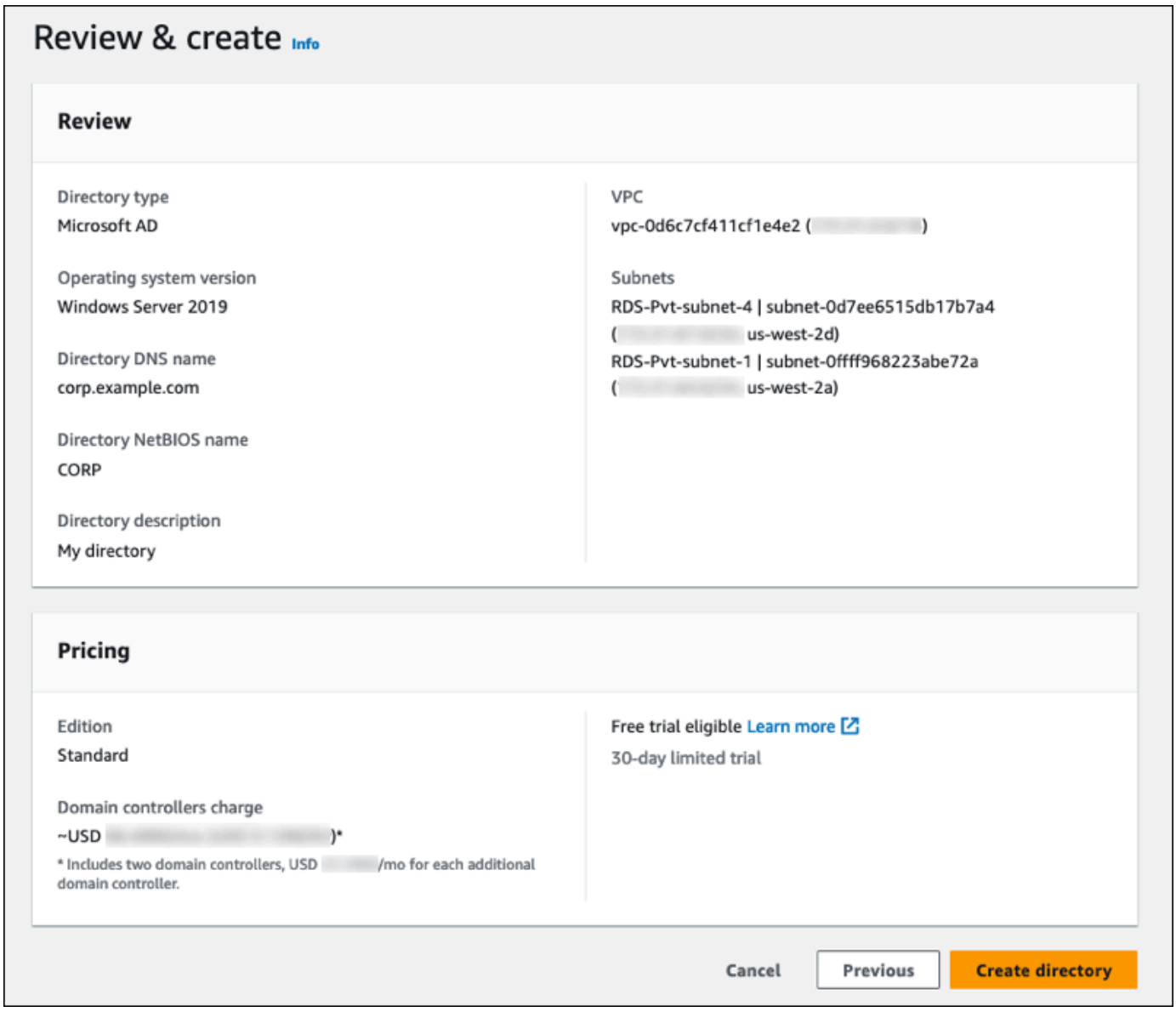
     The directory administrator password can't include the word "admin." The password is case-sensitive and must be 8–64 characters in length. It must also contain at least one character from three of the following four categories:

     - Lowercase letters (a–z)
     - Uppercase letters (A–Z)
     - Numbers (0–9)

- Nonalphanumeric characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)
- Confirm password – Retype the administrator password.

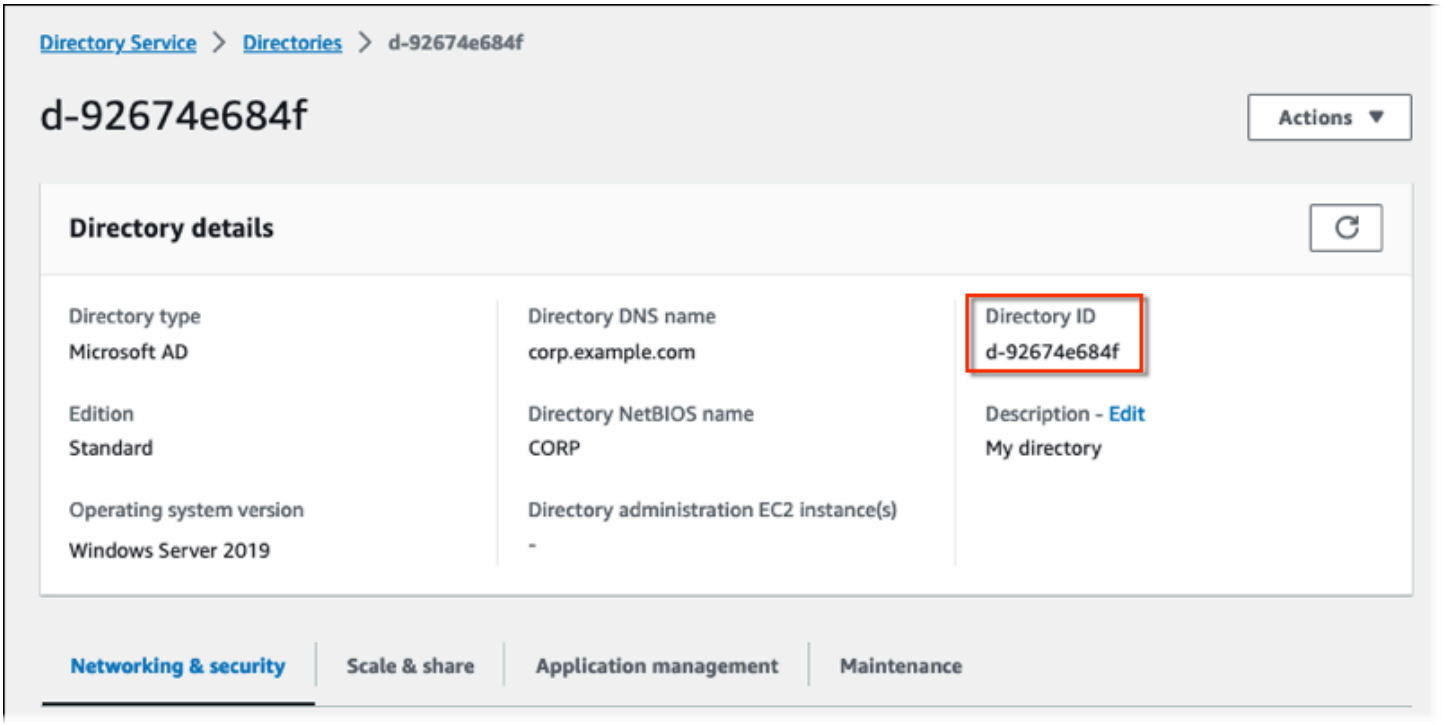
> **Important**
>
> Make sure that you save this password. Directory Service doesn't store this password, and it can't be retrieved or reset.

6. Choose **Next**.
7. On the **Choose VPC and subnets** page, provide the following information:
   - **VPC** – Choose the VPC for the directory. You can create the RDS for Db2 DB instance in this same VPC or in a different VPC.
   - **Subnets** – Choose the subnets for the directory servers. The two subnets must be in different Availability Zones.
8. Choose **Next**.
9. Review the directory information. If changes are needed, choose **Previous** and make the changes. When the information is correct, choose **Create directory**.



It takes several minutes for the directory to be created. When it has been successfully created, the **Status** value changes to **Active**.

To see information about your directory, choose the directory ID under **Directory ID**. Make a note of the **Directory ID** value. You need this value when you create or modify your RDS for Db2 DB instance.



Step 2: Create a trust

If you plan to use AWS Managed Microsoft AD only, skip to Step 3: Create an IAM role for Amazon RDS to access Directory Service.

To enable Kerberos authentication using your self-managed Active Directory, you must create a forest trust relationship between your self-managed Active Directory and the . A forest trust is a trust relationship between a Microsoft AD and a self-managed Active Directory and the AWS Managed Microsoft AD created in the previous step. The trust can also be two-way, where both Active Directories trust each other. For more information about setting up forest trusts using Directory Service, see When to create a trust relationship in the *AWS Directory Service Administration Guide*.

Step 3: Create an IAM role for Amazon RDS to access Directory Service

For Amazon RDS to call Directory Service for you, your AWS account needs an IAM role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`. This role allows Amazon RDS to make calls to Directory Service.

When you create a DB instance using the AWS Management Console and your console user account has the `iam:CreateRole` permission, the console creates the needed IAM role automatically. In this case, the role name is `rds-directoryservice-kerberos-access-role`. Otherwise, you

must create the IAM role manually. When you create this IAM role, choose `Directory Service`, and attach the AWS managed policy `AmazonRDSDirectoryServiceAccess` to it.

For more information about creating IAM roles for a service, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Note**
>
> The IAM role used for Windows Authentication for RDS for Microsoft SQL Server can't be used for RDS for Db2.

As an alternative to using the `AmazonRDSDirectoryServiceAccess` managed policy, you can create policies with the required permissions. In this case, the IAM role must have the following IAM trust policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The role must also have the following IAM role policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

Step 4: Create and configure users

You can create users by using the Active Directory Users and Computers tool. This is one of the Active Directory Domain Services and Active Directory Lightweight Directory Services tools. For more information, see Add Users and Computers to the Active Directory domain in the Microsoft documentation. In this case, users are individuals or other entities, such as their computers, that are part of the domain and whose identities are being maintained in the directory.

To create users in an Directory Service directory, you must be connected to a Windows-based Amazon EC2 instance that's a member of the Directory Service directory. At the same time, you must be signed in as a user that has privileges to create users. For more information, see Create a user in the *AWS Directory Service Administration Guide*.

Step 5: Create an RDS for Db2 admin group in AWS Managed Microsoft AD

RDS for Db2 doesn't support Kerberos authentication for the master user or the two Amazon RDS reserved users `rdsdb` and `rdsadmin`. Instead, you need to create a new group called `masterdba` in AWS Managed Microsoft AD. For more information, see Create a Group Account in Active Directory in the Microsoft documentation. Any users that you add to this group will have master user privileges.

After you enable Kerberos authentication, the master user loses the `masterdba` role. As a result, the master user won't be able to access the instance local user group membership unless you disable Kerberos authentication. To continue to use the master user with password login, create a user on AWS Managed Microsoft AD with the same name as the master user. Then, add that user to the group `masterdba`.

Step 6: Modify DB parameter

If you plan to use AWS Managed Microsoft AD only, skip to Step 7: Create or modify an RDS for Db2 DB instance.

To enable Kerberos authentication using your self-managed Active Directory, you must set the parameter `rds.active_directory_configuration` to `AWS_MANAGED_AD_WITH_TRUST` in your parameter group. By default, this parameter is set to `AWS_MANAGED_AD` for using AWS Managed Microsoft AD only.

For information about modifying DB parameters, see Modifying the parameters in parameter groups.

Step 7: Create or modify an RDS for Db2 DB instance

Create or modify an RDS for Db2 DB instance for use with your directory. You can use the AWS Management Console, the AWS CLI, or the RDS API to associate a DB instance with a directory. You can do this in one of the following ways:

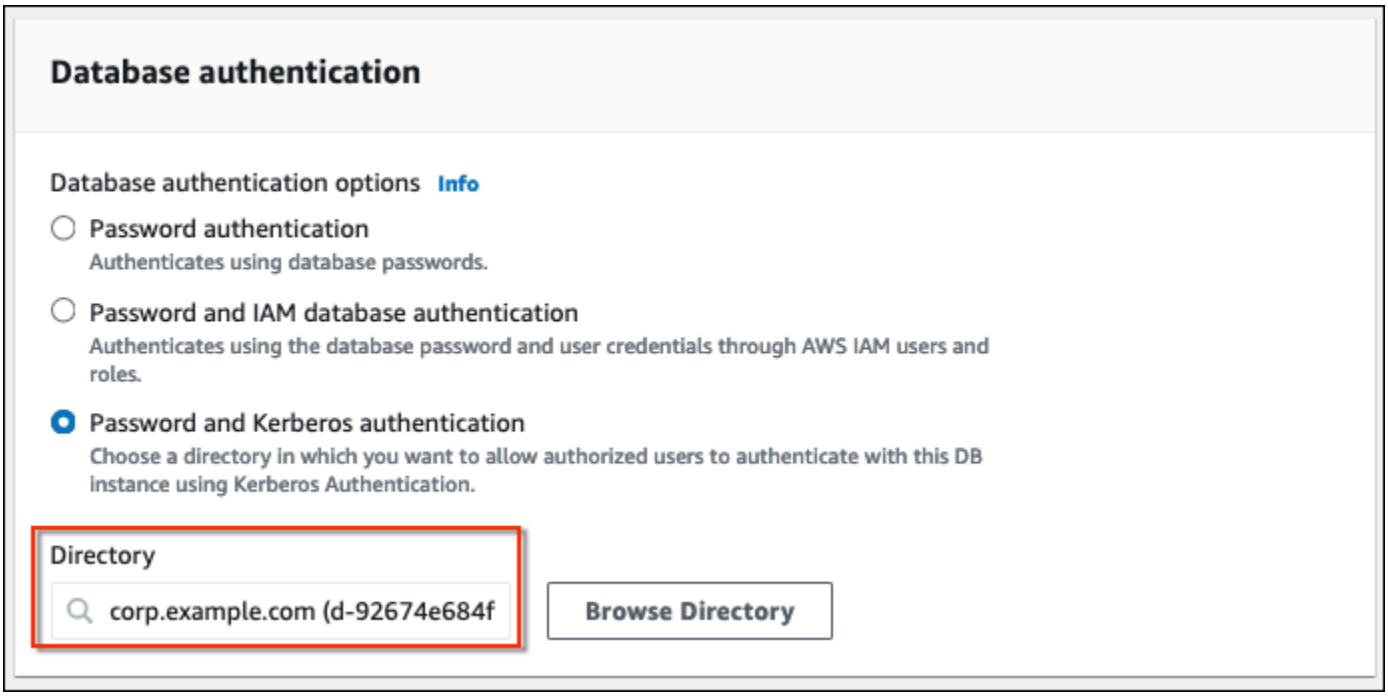
- Create a new RDS for Db2 DB instance using the console, the create-db-instance command, or the CreateDBInstance API operation. For instructions, see Creating an Amazon RDS DB instance.
- Modify an existing RDS for Db2 DB instance using the console, the modify-db-instance command, or the ModifyDBInstance API operation. For instructions, see Modifying an Amazon RDS DB instance.
- Restore an RDS for Db2 DB instance from a DB snapshot using the console, the restore-db-instance-from-db-snapshot command, or the RestoreDBInstanceFromDBSnapshot API operation. For instructions, see Restoring to a DB instance.
- Restore an RDS for Db2 DB instance to a point-in-time using the console, the restore-db-instance-to-point-in-time command, or the RestoreDBInstanceToPointInTime API operation. For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

Kerberos authentication is only supported for RDS for Db2 DB instances in a VPC. The DB instance can be in the same VPC as the directory, or in a different VPC. The DB instance must use a

security group that allows ingress and egress within the directory's VPC so the DB instance can communicate with the directory.

Console

When you use the console to create, modify, or restore a DB instance, choose **Password and Kerberos authentication** in the **Database authentication** section. Then choose **Browse Directory**. Select the directory or choose **Create directory** to use the Directory Service.



AWS CLI

When you use the AWS CLI, the following parameters are required for the DB instance to be able to use the directory that you created:

- For the `--domain` parameter, use the domain identifier ("`d-*`" identifier) generated when you created the directory.
- For the `--domain-iam-role-name` parameter, use the role you created that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

The following example modifies a DB instance to use a directory. Replace the following placeholders in the example with your own values:

- *`db_instance_name`* – The name of your RDS for Db2 DB instance.

- *directory_id* – The ID of the AWS Directory Service for Microsoft Active Directory directory that you created.
- *role_name* – The name of the IAM role that you created.

```
aws rds modify-db-instance --db-instance-identifier db_instance_name --domain
 d-directory_id --domain-iam-role-name role_name
```

> **Important**
>
> If you modify a DB instance to enable Kerberos authentication, reboot the DB instance after making the change.

Step 8: Retrieve the Active Directory group SID in PowerShell

A security ID (SID) uniquely identifies a security principal or security group. When a security group or account is created in Active Directory, Active Directory assigns a SID to the group. To retrieve the AD security group SID from Active Directory, use the `Get-ADGroup` cmdlet in a Windows client machine that is part of the Active Directory domain. The `Identity` parameter specifies the Active Directory group name that you want the SID for.

The following example returns the SID of the Active Directory group `adgroup1`.

```
C:\Users\Admin> Get-ADGroup -Identity adgroup1 | select SID

             SID
-----------------------------------------------
S-1-5-21-3168537779-1985441202-1799118680-1612
```

You must generate this mapping for all the groups that are relevant to the database.

Step 9: Add SID to GroupName mappings to your RDS for Db2 DB instance

You need to add the SID to GroupName mappings created in the previous step to your RDS for Db2 DB instance. For each mapping, call the following stored procedure. Replace the *SID* and *group_name* with your own information.

```
db2 connect to rdsadmin
db2 "call rdsadmin.set_sid_group_mapping(?, 'SID','group_name')"
```

For more information, see rdsadmin.set_sid_group_mapping.

For information about checking the task status, see rdsadmin.get_task_status.

Step 10: Configure a Db2 client

To configure a Db2 client

1. Create an **/etc/krb5.conf** file (or equivalent) to point to the domain.

   > **Note**
   >
   > For Windows operating systems, create a **C:\windows\krb5.ini** file.

2. Verify that traffic can flow between the client host and Directory Service. Use a network utility such as Netcat for the following tasks:

   a. Verify traffic over DNS for port 53.

   b. Verify traffic over TCP/UDP for port 53 and for Kerberos, which includes ports 88 and 464 for Directory Service.

3. Verify that traffic can flow between the client host and the DB instance over the database port. You can use the command db2 to connect and access the database.

The following example is /etc/krb5.conf file content for AWS Managed Microsoft AD:

```
[libdefaults]
default_realm = EXAMPLE.COM
[realms]
EXAMPLE.COM = {
kdc = example.com
admin_server = example.com
}
[domain_realm]
.example.com = EXAMPLE.COM
example.com = EXAMPLE.COM
```

Connecting to Amazon RDS for Db2 with Kerberos authentication

Use the following procedure to connect to your Amazon RDS for Db2 DB instance with Kerberos authentication.

To connect to RDS for Db2 with Kerberos authentication

1. At a command prompt, run the following command. In the following example, replace *username* with your Microsoft Active Directory username.

```
kinit username
```

2. If the RDS for Db2 DB instance is using a publicly accessible VPC, add the IP address for your DB instance endpoint to your `/etc/hosts` file on the Amazon EC2 client. The following example obtains the IP address and then adds it to the `/etc/hosts` file.

```
% dig +short Db2-endpoint.AWS-Region.rds.amazonaws.com
;; Truncated, retrying in TCP mode.
ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
34.210.197.118

% echo "34.210.197.118  Db2-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/hosts
```

3. Use the following command to log in to an RDS for Db2 DB instance that is associated with Active Directory. Replace *database_name* with the name of your RDS for Db2 database.

```
db2 connect to database_name
```

Administering your Amazon RDS for Db2 DB instance

This topic covers the common management tasks that you perform with an Amazon RDS for Db2 DB instance. Some tasks are the same for all Amazon RDS DB instances. Other tasks are specific to RDS for Db2.

The following tasks are common to all RDS databases. There are also tasks specific to RDS for Db2, such as connecting to an RDS for Db2 database with a standard SQL client.

Task area	Relevant documentation
Instance classes, storage, and PIOPS If you are creating a production instance, learn how instance classes, storage types, and Provisioned IOPS work in Amazon RDS.	DB instance classes Amazon RDS storage types
Multi-AZ deployments A production DB instance should use Multi-AZ deployments. Multi-AZ deployments provide increased availability, data durability, and fault tolerance for DB instances.	Configuring and managing a Multi-AZ deployment for Amazon RDS
Amazon VPC If your AWS account has a default virtual private cloud (VPC), then your DB instance is automatically created inside the default VPC. If your account doesn't have a default VPC, and you want the DB instance in a VPC, create the VPC and subnet groups before you create the DB instance.	Working with a DB instance in a VPC
Security groups By default, DB instances use a firewall that prevents access. Make sure that you create a security group with the correct IP addresses and network configuration to access the DB instance.	Controlling access with security groups
Parameter groups	Adding IBM IDs to a parameter group for RDS for Db2 DB instances

Task area	Relevant documentation
Because your RDS for Db2 DB instance requires that you add the `rds.ibm_customer_id` and `rds.ibm_site_id` parameters, create a parameter group before you create the DB instance. If your DB instance requires other specific database parameters, also add them to this parameter group before you create the DB instance.	Parameter groups for Amazon RDS
Option groups If your DB instance requires specific database options, create an option group before you create the DB instance.	Options for Amazon RDS for Db2 DB instances
Connecting to your DB instance After creating a security group and associating it to a DB instance, you can connect to the DB instance with any standard SQL client application such as IBM Db2 CLP.	Connecting to your Db2 DB instance
Backup and restore You can configure your DB instance to take automated storage backups, or take manual storage snapshots, and then restore instances from the backups or snapshots.	Backing up, restoring, and exporting data
Monitoring You can monitor an RDS for Db2 DB instance with IBM Db2 Data Management Console. You can also monitor an RDS for Db2 DB instance by using CloudWatch Amazon RDS metrics, events, and enhanced monitoring.	Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 Data Management Console Viewing metrics in the Amazon RDS console Viewing Amazon RDS events Monitoring OS metrics with Enhanced Monitoring

Task area	Relevant documentation
Log files You can access the log files for your RDS for Db2 DB instance.	Monitoring Amazon RDS log files

Topics

- Performing common system tasks for Amazon RDS for Db2 DB instances
- Performing common database tasks for Amazon RDS for Db2 DB instances

Performing common system tasks for Amazon RDS for Db2 DB instances

You can perform certain common database administrator tasks related to the system on your Amazon RDS DB instances running Db2. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances, and restricts access to certain system procedures and tables that require advanced privileges.

For information about granting and revoking privileges and attaching to the remote database for RDS for Db2, see the following topics.

Topics

- Granting and revoking privileges for RDS for Db2
- Attaching to the remote RDS for Db2 DB instance

Creating a custom database endpoint

When you migrate to Amazon RDS for Db2, you can use custom database endpoint URLs to minimize changes to your application. For example, if you use `db2.example.com` as your current DNS record, you can add it to Amazon Route 53. In Route 53, you can use private hosted zones to map your current DNS database endpoint to an RDS for Db2 database endpoint. To add a custom `A` or `CNAME` record for an Amazon RDS database endpoint, see Registering and managing domains using Amazon Route 53 in the *Amazon Route 53 Developer Guide*.

> **Note**
>
> If you can't transfer your domain to Route 53, you can use your DNS provider to create a CNAME record for the RDS for Db2 database endpoint URL. Consult your DNS provider documentation.

Granting and revoking privileges for RDS for Db2

Users gain access to databases through membership in groups that are attached to databases.

Use the following procedures to grant and revoke privileges to control access to your database.

These procedures use IBM Db2 CLP running on a local machine to connect to an RDS for Db2 DB instance. Be sure to catalog the TCPIP node and the database to connect to your RDS for Db2 DB instance running on your local machine. For more information, see Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 CLP.

Topics

- Granting a user access to your database
- Changing a user's password
- Adding groups to a user
- Removing groups from a user
- Removing a user
- Listing users
- Creating a role
- Granting a role
- Revoking a role
- Dropping a role
- Granting database authorization
- Revoking database authorization

Granting a user access to your database

To grant a user access to your database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 connect to rdsadmin user master_username using master_password
   ```

   This command produces output similar to the following example:

   ```
   Database Connection Information

   Database server        = DB2/LINUXX8664 11.5.8.0
   SQL authorization ID   = ADMIN
   Local database alias   = RDSADMIN
   ```

2. Add a user to your authorization list by calling `rdsadmin.add_user`. For more information, see rdsadmin.add_user.

   ```
   db2 "call rdsadmin.add_user(
       'username',
       'password',
       'group_name,group_name')"
   ```

3. (Optional) Add additional groups to the user by calling `rdsadmin.add_groups`. For more information, see rdsadmin.add_groups.

   ```
   db2 "call rdsadmin.add_groups(
       'username',
       'group_name,group_name')"
   ```

4. Confirm the authorities that are available to the user. In the following example, replace *`rds_database_alias`*, *`master_user`*, and *`master_password`* with your own information. Also, replace *`username`* with the user's username.

   ```
   db2 terminate
   db2 connect to rds_database_alias user master_user using master_password
   db2 "SELECT SUBSTR(AUTHORITY,1,20) AUTHORITY, D_USER, D_GROUP, D_PUBLIC
   ```

```
      FROM TABLE (SYSPROC.AUTH_LIST_AUTHORITIES_FOR_AUTHID ('username', 'U') ) AS
T
      ORDER BY AUTHORITY"
```

This command produces output similar to the following example:

```
AUTHORITY            D_USER D_GROUP D_PUBLIC
-------------------- ------ ------- --------
ACCESSCTRL           N      N       N
BINDADD              N      N       N
CONNECT              N      N       N
CREATETAB            N      N       N
CREATE_EXTERNAL_ROUT N      N       N
CREATE_NOT_FENCED_RO N      N       N
CREATE_SECURE_OBJECT N      N       N
DATAACCESS           N      N       N
DBADM                N      N       N
EXPLAIN              N      N       N
IMPLICIT_SCHEMA      N      N       N
LOAD                 N      N       N
QUIESCE_CONNECT      N      N       N
SECADM               N      N       N
SQLADM               N      N       N
SYSADM               *      N       *
SYSCTRL              *      N       *
SYSMAINT             *      N       *
SYSMON               *      N       *
WLMADM               N      N       N
```

5. Grant the RDS for Db2 roles ROLE_NULLID_PACKAGES, ROLE_TABLESPACES, and ROLE_PROCEDURES to the group that you added the user to. For more information, see Amazon RDS for Db2 default roles.

> **Note**
>
> We create RDS for Db2 DB instances in RESTRICTIVE mode. Therefore, the RDS for Db2 roles ROLE_NULLID_PACKAGES, ROLE_TABLESPACES, and ROLE_PROCEDURES grant execute privileges on NULLID packages for IBM Db2 CLP and Dynamic SQL. These roles also grant user privileges on tablespaces.

a. Connect to your Db2 database. In the following example, replace *database_name*, *master_user*, and *master_password* with your own information.

```
db2 connect to database_name user master_user using master_password
```

b. Grant the role ROLE_NULLED_PACKAGES to a group. In the following example, replace *group_name* with the name of the group that you want to add the role to.

```
db2 "grant role ROLE_NULLID_PACKAGES to group group_name"
```

c. Grant the role ROLE_TABLESPACES to the same group. In the following example, replace *group_name* with the name of the group that you want to add the role to.

```
db2 "grant role ROLE_TABLESPACES to group group_name"
```

d. Grant the role ROLE_PROCEDURES to the same group. In the following example, replace *group_name* with the name of the group that you want to add the role to.

```
db2 "grant role ROLE_PROCEDURES to group group_name"
```

6. Grant connect, bindadd, createtab, and IMPLICIT_SCHEMA authorities to the group that you added the user to. In the following example, replace *group_name* with the name of the second group that you added the user to.

```
db2 "grant usage on workload SYSDEFAULTUSERWORKLOAD to public"
db2 "grant connect, bindadd, createtab, implicit_schema on database to
 group group_name"
```

7. Repeat steps 4 through 6 for each additional group that you added the user to.

8. Test the user's access by connecting as the user, creating a table, inserting values into the table, and returning data from the table. In the following example, replace *rds_database_alias*, *username*, and *password* with the name of the database and the user's username and password.

```
db2 connect to rds_database_alias user username using password
db2 "create table t1(c1 int not null)"
db2 "insert into t1 values (1),(2),(3),(4)"
db2 "select * from t1"
```

Changing a user's password

To change a user's password

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Change the password by calling `rdsadmin.change_password`. For more information, see rdsadmin.change_password.

```
db2 "call rdsadmin.change_password(
    'username',
    'new_password')"
```

Adding groups to a user

To add groups to a user

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Add groups to a user by calling `rdsadmin.add_groups`. For more information, see rdsadmin.add_groups.

```
db2 "call rdsadmin.add_groups(
    'username',
    'group_name,group_name')"
```

Removing groups from a user

To remove groups from a user

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 connect to rdsadmin user master_username using master_password
   ```

2. Remove groups by calling `rdsadmin.remove_groups`. For more information, see rdsadmin.remove_groups.

   ```
   db2 "call rdsadmin.remove_groups(
       'username',
       'group_name,group_name')"
   ```

Removing a user

To remove a user from the authorization list

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 connect to rdsadmin user master_username using master_password
   ```

2. Remove a user from your authorization list by calling `rdsadmin.remove_user`. For more information, see rdsadmin.remove_user.

   ```
   db2 "call rdsadmin.remove_user('username')"
   ```

Listing users

To list users on an authorization list, call the `rdsadmin.list_users` stored procedure. For more information, see rdsadmin.list_users.

```
db2 "call rdsadmin.list_users()"
```

Creating a role

You can use the rdsadmin.create_role stored procedure to create a role.

To create a role

1. Connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Set Db2 to output content.

```
db2 set serveroutput on
```

3. Create a role. For more information, see the section called "rdsadmin.create_role".

```
db2 "call rdsadmin.create_role(
    'database_name',
    'role_name')"
```

4. Set Db2 to not output content.

```
db2 set serveroutput off
```

Granting a role

You can use the rdsadmin.grant_role stored procedure to assign a role to a role, user, or group.

To assign a role

1. Connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Set Db2 to output content.

```
db2 set serveroutput on
```

3. Assign a role. For more information, see the section called "rdsadmin.grant_role".

```
db2 "call rdsadmin.grant_role(
    'database_name',
    'role_name',
    'grantee',
    'admin_option')"
```

4. Set Db2 to not output content.

```
db2 set serveroutput off
```

Revoking a role

You can use the rdsadmin.revoke_role stored procedure to revoke a role from a role, user, or group.

To revoke a role

1. Connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Revoke a role. For more information, see the section called “rdsadmin.revoke_role”.

```
db2 "call rdsadmin.revoke_role(
    ?,
    'database_name',
    'role_name',
    'grantee')"
```

Dropping a role

You can use the rdsadmin.drop_role stored procedure to drop a role.

To drop a role

1. Connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Drop a role. For more information, see the section called “rdsadmin.drop_role”.

```
db2 "call rdsadmin.drop_role(
    ?,
    'database_name',
    'role_name')"
```

Granting database authorization

The master user, who has DBADM authorization, can grant DBADM, ACCESSCTRL, or DATAACCESS authorization to a role, user, or group.

To grant database authorization

1. Connect to the rdsadmin database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *master_username* and *master_password* with your own information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Grant a user access by calling rdsadmin.dbadm_grant. For more information, see rdsadmin.dbadm_grant.

```
db2 "call rdsadmin.dbadm_grant(
    ?,
    'database_name,
    'authorization',
    'grantee')"
```

Example use case

The following procedure walks you through creating a role, granting DBADM authorization to the role, assigning the role to a user, and granting the role to a group.

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Create a role called `PROD_ROLE` for a database called `TESTDB`. For more information, see rdsadmin.create_role.

```
db2 "call rdsadmin.create_role(
    'TESTDB',
    'PROD_ROLE')"
```

3. Assign the role to a user called `PROD_USER`. The `PROD_USER` is given admin authorization to assign roles. For more information, see rdsadmin.grant_role.

```
db2 "call rdsadmin.grant_role(
    ?,
    'TESTDB',
    'PROD_ROLE',
    'USER PROD_USER',
    'Y')"
```

4. (Optional) Provide additional authorization or privileges. The following example grants `DBADM` authorization to a role named `PROD_ROLE` for a database called `FUNDPROD`. For more information, see rdsadmin.dbadm_grant.

```
db2 "call rdsadmin.dbadm_grant(
    ?,
    'FUNDPROD',
    'DBADM',
    'ROLE PROD_ROLE')"
```

5. Terminate your session.

```
db2 terminate
```

6. Connect to the `TESTDB` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 connect to TESTDB user master_username using master_password
```

7. Add more authorizations to the role.

```
db2 "grant connect, implicit_schema on database to role PROD_ROLE"
```

8. Grant the role PROD_ROLE to a group.

```
db2 "grant role PROD_ROLE to group PRODGRP"
```

Users who belong to the group PRODGRP can now perform actions such as connecting to the TESTDB database, creating tables, or creating schemas.

Revoking database authorization

The master user, who has DBADM authorization, can revoke DBADM, ACCESSCTRL, or DATAACCESS authorization from a role, user, or group.

To revoke database authorization

1. Connect to the rdsadmin database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *master_username* and *master_password* with your own information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Revoke user access by calling rdsadmin.dbadm_revoke. For more information, see rdsadmin.dbadm_revoke.

```
db2 "call rdsadmin.dbadm_revoke(
    ?,
    'database_name,
    'authorization',
    'grantee')"
```

Attaching to the remote RDS for Db2 DB instance

Use the following steps to attach to your remote RDS for Db2 DB instance and run `get snapshot` operations.

To attach to the remote RDS for Db2 DB instance

1. Run a client-side IBM Db2 CLP session. For information about cataloging your RDS for Db2 DB instance and database, see Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 CLP. Make a note of the master username and master password for your RDS for Db2 DB instance.
2. Attach to the RDS for Db2 DB instance. In the following example, replace *node_name*, *master_username*, and *master_password* with the TCPIP node name that you catalogued and the master username and master password for your RDS for Db2 DB instance.

```
db2 attach to node_name user master_username using master_password
```

After attaching to the remote RDS for Db2 DB instance, you can run the following commands and other `get snapshot` commands. For more information, see GET SNAPSHOT command in the IBM Db2 documentation.

```
db2 list applications
db2 get snapshot for all databases
db2 get snapshot for database manager
db2 get snapshot for all applications
```

Performing common database tasks for Amazon RDS for Db2 DB instances

You can perform certain common DBA tasks related to databases on your Amazon RDS for Db2 DB instances. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. Also, the master user can't run commands or utilities requiring SYSADM, SYSMAINT, or SYSCTRL authorities.

For information about common tasks for buffer pools, databases, and tablespaces, see the following topics.

Topics

- Common tasks for buffer pools
- Common tasks for databases
- Common tasks for tablespaces

Common tasks for buffer pools

You can create, alter, or drop buffer pools for an RDS for Db2 database. Creating, altering, or dropping buffer pools requires higher-level SYSADM or SYSCTRL authority, which isn't available to the master user. Instead, use Amazon RDS stored procedures.

You can also flush buffer pools.

Topics

- Creating a buffer pool
- Altering a buffer pool
- Dropping a buffer pool
- Flushing the buffer pools

Creating a buffer pool

To create a buffer pool for your RDS for Db2 database, call the `rdsadmin.create_bufferpool` stored procedure. For more information, see CREATE BUFFERPOOL statement in the IBM Db2 documentation.

To create a buffer pool

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Create a buffer pool by calling `rdsadmin.create_bufferpool`. For more information, see rdsadmin.create_bufferpool.

   ```
   db2 "call rdsadmin.create_bufferpool(
   ```

```
    'database_name',
    'buffer_pool_name',
    buffer_pool_size,
    'immediate',
    'automatic',
    page_size,
    number_block_pages,
    block_size)"
```

Altering a buffer pool

To alter a buffer pool for your RDS for Db2 database, call the `rdsadmin.alter_bufferpool` stored procedure. For more information, see ALTER BUFFERPOOL statement in the IBM Db2 documentation.

To alter a buffer pool

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Alter a buffer pool by calling `rdsadmin.alter_bufferpool`. For more information, see rdsadmin.alter_bufferpool.

   ```
   db2 "call rdsadmin.alter_bufferpool(
       'database_name',
       'buffer_pool_name',
       buffer_pool_size,
       'immediate',
       'automatic',
       change_number_blocks,
       number_block_pages,
       block_size)"
   ```

Dropping a buffer pool

To drop a buffer pool for your RDS for Db2 database, call the `rdsadmin.drop_bufferpool` stored procedure. For more information, see Dropping buffer pools in the IBM Db2 documentation.

Important

Make sure that no tablespaces are assigned to the buffer pool that you want to drop.

To drop a buffer pool

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Drop a buffer pool by calling `rdsadmin.drop_bufferpool`. For more information, see rdsadmin.drop_bufferpool.

```
db2 "call rdsadmin.drop_bufferpool(
    'database_name',
    'buffer_pool_name')"
```

Flushing the buffer pools

You can flush the buffer pools to force a checkpoint so that RDS for Db2 writes pages from memory to storage.

Note

You don't need to flush the buffer pools. Db2 writes logs synchronously before it commits transactions. The dirty pages might still be in a buffer pool, but Db2 writes them to storage asynchronously. Even if the system shuts down unexpectedly, when you restart the database, Db2 automatically performs crash recovery. During crash recovery, Db2 writes committed changes to the database or rolls back changes for uncommitted transactions.

To flush the buffer pools

1. Connect to your Db2 database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *rds_database_alias*, *master_username*, and *master_password* with your own information.

```
db2 connect to rds_database_alias user master_username using master_password
```

2. Flush the buffer pools.

```
db2 flush bufferpools all
```

Common tasks for databases

You can create, drop, or restore databases on your RDS for Db2 DB instance. Creating, dropping, or restoring databases requires higher-level SYSADM authority, which isn't available to the master user. Instead, use Amazon RDS stored procedures.

You can also perform common management tasks such as monitoring, maintenance, and the collection of information about your databases.

Topics

- Creating a database
- Configuring settings for a database
- Modifying database parameters
- Configuring log retention
- Listing log information
- Using fine-grained access control (FGAC)
- Deactivating a database
- Activating a database
- Reactivating a database
- Dropping a database
- Backing up a database
- Copying archive logs to Amazon S3

- Restoring a database
- Listing databases
- Collecting information about databases
- Forcing applications off of databases
- Generating performance reports

Creating a database

To create a database on your RDS for Db2 DB instance, call the `rdsadmin.create_database` stored procedure. For more information, see CREATE DATABASE command in the IBM Db2 documentation.

> **Note**
>
> If you plan on modifying the `db2_compatibility_vector` parameter, modify the parameter before creating a database. For more information, see Setting the db2_compatibility_vector parameter.

To create a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Create a database by calling `rdsadmin.create_database`. For more information, see rdsadmin.create_database.

   ```
   db2 "call rdsadmin.create_database(
       'database_name',
       'database_page_size',
       'database_code_set',
       'database_territory',
       'database_collation',
       'database_autoconfigure_str',
       'database_non-restrictive')"
   ```

3. (Optional) Create additional databases by calling `rdsadmin.create_database` for each database you want to create. Each Db2 DB instance can contain up to 50 databases. For more information, see rdsadmin.create_database.

```
db2 "call rdsadmin.create_database('database_name')"
```

4. (Optional) Confirm that your database was created by using one of the following methods:

   - Call `rdsadmin.list_databases`. For more information, see rdsadmin.list_databases.
   - Run the following SQL command:

```
db2 "select varchar(r.task_type,25) as task_type, r.database_name,
    varchar(r.lifecycle,15) as lifecycle, r.created_at, r.database_name,
    varchar(bson_to_json(task_input_params),256) as input_params,
    varchar(r.task_output,1024) as task_output
    from table(rdsadmin.get_task_status(null,null,'create_database'))
    as r order by created_at desc"
```

Configuring settings for a database

To configure the settings for a database on your RDS for Db2 DB instance, call the `rdsadmin.set_configuration` stored procedure. For example, you could configure the number of buffers or buffer manipulators to create during a restore operation.

To configure settings for a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. (Optional) Check your current configuration settings by calling `rdsadmin.show_configuration`. For more information, see the section called "rdsadmin.show_configuration".

```
db2 "call rdsadmin.show_configuration('name')"
```

3. Configure the settings for the database by calling `rdsadmin.set_configuration`. For more information, see the section called "rdsadmin.set_configuration".

```
db2 "call rdsadmin.set_configuration(
    'name',
    'value')"
```

Modifying database parameters

Amazon RDS for Db2 uses three types of parameters: database manager configuration parameters, registry variables, and database configuration parameters. You can update the first two types through parameter groups and the last type through the rdsadmin.update_db_param stored procedure.

> **Note**
>
> You can only modify the values of existing parameters. You can't add new parameters that RDS for Db2 doesn't support.

For more information these parameters and how to modify their values, see the section called "Db2 parameters".

Configuring log retention

To configure how long Amazon RDS retains log files for your RDS for Db2 database, call the `rdsadmin.set_archive_log_retention` stored procedure.

To configure log retention for a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. (Optional) Check your current configuration for log retention by calling `rdsadmin.show_archive_log_retention`. For more information, see the section called "rdsadmin.show_archive_log_retention".

```
db2 "call rdsadmin.show_archive_log_retention(
    ?,
    'database_name')"
```

3. Configure log retention for the database by calling `rdsadmin.set_archive_log_retention`. For more information, see the section called "rdsadmin.set_archive_log_retention".

```
db2 "call rdsadmin.set_archive_log_retention(
    ?,
    'database_name',
    'archive_log_retention_hours')"
```

Listing log information

To list details about archive log files, including such details as total storage size used, call the `rdsadmin.list_archive_log_information` stored procedure.

To list log information for a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Return a list of log file information by calling `rdsadmin.list_archive_log_information`. For more information, see the section called "rdsadmin.list_archive_log_information".

```
db2 "call rdsadmin.list_archive_log_information(
    ?,
    'database_name')"
```

Using fine-grained access control (FGAC)

To use fine-grained access control commands to control access to table data in a database on an RDS for Db2 DB instance, call the `rdsadmin.fgac_command` stored procedure. You might want to

use FGAC to limit access to data based on user roles or data attributes. For example, you could limit access to patient health care data based on the type of data or to certain medical care providers.

To use fine-grained access control to control access to table data in a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Run various fine-grained access control commands by calling `rdsadmin.fgac_command`. For more information, see rdsadmin.fgac_command.

   ```
   db2 "call rdsadmin.fgac_command(
       ?,
       'database_name',
       'fgac_command')"
   ```

Deactivating a database

To deactivate a database on your RDS for Db2 DB instance, call the `rdsadmin.deactivate_database` stored procedure.

By default, Amazon RDS activates a database when you create a database on your RDS for Db2 DB instance. You can deactivate infrequently used databases to conserve memory resources.

To deactivate a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Deactivate a database by calling `rdsadmin.deactivate_database`. For more information, see rdsadmin.deactivate_database.

   ```
   db2 "call rdsadmin.deactivate_database(
       ?,
   ```

```
    'database_name')"
```

Activating a database

To activate a database on a standalone RDS for Db2 DB instance, call the `rdsadmin.activate_database` stored procedure.

By default, Amazon RDS activates a database when you create a database on your RDS for Db2 DB instance. You can deactivate infrequently used databases to conserve memory resources, and then later activate a deactivated database.

To activate a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Activate a database by calling `rdsadmin.activate_database`. For more information, see rdsadmin.activate_database.

   ```
   db2 "call rdsadmin.activate_database(
       ?,
       'database_name')"
   ```

Reactivating a database

To reactivate a database on a replica source RDS for Db2 DB instance, call the `rdsadmin.reactivate_database` stored procedure. After you make changes to database configurations, you might need to reactivate a database on an RDS for Db2 DB instance. To determine if you need to reactivate a database, connect to the database and run `db2 get db cfg show detail`.

You can also call this stored procedure to reactivate a database on a standalone RDS for Db2 DB instance after you make changes to database configurations. Or, you could reactivate a database on a standalone RDS for Db2 DB instance by first calling the `rdsadmin.deactivate_database`

stored procedure and then the `rdsadmin.activate_database` stored procedure. For more information, see Deactivating a database and Activating a database.

To reactivate a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Reactivate a database by calling `rdsadmin.reactivate_database`. For more information, see rdsadmin.reactivate_database.

```
db2 "call rdsadmin.reactivate_database(
    ?,
    'database_name')"
```

Dropping a database

To drop a database from your RDS for Db2 DB instance, call the `rdsadmin.drop_database` stored procedure. For more information, see Dropping databases in the IBM Db2 documentation.

> **Note**
>
> You can drop a database by calling the stored procedure only if certain conditions are met. For more information, see the section called "Usage notes" for `rdsadmin.drop_database`.

To drop a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Drop a database by calling `rdsadmin.drop_database`. For more information, see rdsadmin.drop_database.

```
db2 "call rdsadmin.drop_database('database_name')"
```

Backing up a database

To back up a database in your RDS for Db2 DB instance to Amazon S3, call the `rdsadmin.backup_database` stored procedure. For more information, see BACKUP DATABASE command in the IBM Db2 documentation.

> **Note**
>
> This stored procedure uses the integration with Amazon S3. Make sure that you have configured the integration before proceeding. For more information, see Integrating an Amazon RDS for Db2 DB instance with Amazon S3.

To back up a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Create a VPC gateway endpoint for S3. If you already have a VPC gateway endpoint for S3, skip to Step 4.

   For an RDS for Db2 DB instance to be able to interact with Amazon S3, you must have a VPC and an Amazon S3 gateway endpoint for private subnets to use. For more information, see Step 1: Create a VPC gateway endpoint for Amazon S3.
3. Confirm the VPC gateway endpoint for S3. For more information, see Step 2: Confirm that your VPC gateway endpoint for Amazon S3 exists.
4. Back up a database by calling `rdsadmin.backup_database`. For more information, see rdsadmin.backup_database.

```
db2 "call rdsadmin.backup_database(
```

```
    ?,
    'database_name',
    's3_bucket_name',
    's3_prefix',
    'backup_type',
    'compression_option',
    'util_impact_priority',
    'num_files',
    'parallelism',
    'num_buffers')"
```

5. Terminate your connection.

```
terminate
```

6. (Optional) Confirm that the backup files were uploaded to your Amazon S3 bucket under *s3_prefix/dbi_resource_id/db_name*. If the files don't appear at *s3_prefix/dbi_resource_id/db_name*, check the status of backing up your database to identify any issues. For more information, see rdsadmin.get_task_status. If you can't resolve any identified issues, contact AWS Support.
7. (Optional) After the backup to Amazon S3 completes, you can restore the backup to an RDS for Db2 DB instance or to another location such as a local server. For information about restoring to an RDS for Db2 DB instance, see Restoring a database.

Copying archive logs to Amazon S3

Db2 archive logs can now be copied from your RDS for Db2 DB instance to Amazon S3. The archive logs combined with native backup created using `rdsadmin.backup_database` can be used to restore and rollforward database to point in time on another RDS for Db2 instance or EC2 database.

Before configuring this feature, use the stored procedure `rdsadmin.backup_database` to set up RDS for Db2 database.

This feature operates at the RDS for Db2 DB instance level, though archive log copying can be enabled or disabled per database.

To configure archive log copying to Amazon S3

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB Instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Setup archive log backup to S3 by calling the section called “rdsadmin.set_configuration”.

```
db2 "call rdsadmin.set_configuration(
    'name',
    'value')"
```

Example:

```
db2 "call rdsadmin.set_configuration('ARCHIVE_LOG_COPY_TARGET_S3_ARN',
 'arn:aws:s3:::my_rds_db2_backups/archive-log-copy/')"
```

3. Enable archive log copying for a database by calling `rdsadmin.enable_archive_log_copy`. Replace *`database_name`* with your database name.

```
db2 "call rdsadmin.enable_archive_log_copy(?, 'database_name')"
```

4. Similarly,to disable archive log copying for a database, call `rdsadmin.disable_archive_log_copy`.

```
db2 "call rdsadmin.disable_archive_log_copy(?, 'database_name')"
```

5. Confirm the archive log copy status by calling `rdsadmin.list_databases`.

```
db2 "select * from table(rdsadmin.list_databases())"
```

Sample output:

```
DATABASE_NAME   CREATE_TIME                DATABASE_UNIQUE_ID
                         ARCHIVE_LOG_RETENTION_HOURS ARCHIVE_LOG_COPY
 ARCHIVE_LOG_LAST_UPLOAD_FILE ARCHIVE_LOG_LAST_UPLOAD_FILE_TIME
 ARCHIVE_LOG_COPY_STATUS
```

```
-------------- -------------------------
 ------------------------------------------------ -------------------------
 --------------- --------------------------- ------------------------------
 ------------------------------
RDSADMIN        2026-01-06-02.03.42.569069 RDSADMIN
                                    0 DISABLED         -
  -                                 -
FOO             2026-01-06-02.13.42.885650 F0D81C7E-7213-4565-B376-4F33FCF420E3
                                    7 ENABLED          S0006536.LOG
  2026-01-28-19.15.10.000000        UPLOADING
CODEP           2026-01-14-19.42.42.508476 106EEF95-6E30-4FFF-85AE-B044352DF095
                                    0 DISABLED         -
  -                                 -
...
```

Restoring a database

To move a database from an Amazon S3 bucket to your RDS for Db2 DB instance, call the `rdsadmin.restore_database` stored procedure. For more information, see RESTORE DATABASE command in the IBM Db2 documentation.

To restore a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. (Optional) Check your current configuration settings to optimize the restore operation by calling `rdsadmin.show_configuration`. For more information, see the section called “rdsadmin.show_configuration”.

   ```
   db2 "call rdsadmin.show_configuration('name')"
   ```

3. Configure the settings to optimize the restore operation by calling `rdsadmin.set_configuration`. Explicitly setting these values can improve the performance when restoring databases with large volumes of data. For more information, see the section called “rdsadmin.set_configuration”.

```
db2 "call rdsadmin.set_configuration(
    'name',
    'value')"
```

4. Restore the database by calling `rdsadmin.restore_database`. For more information, see the section called "rdsadmin.restore_database".

```
db2 "call rdsadmin.restore_database(
    ?,
    'database_name',
    's3_bucket_name',
    's3_prefix',
    restore_timestamp,
    'backup_type')"
```

5. (Optional) Confirm that your database was restored by calling `rdsadmin.list_databases` and checking that the restored database is listed. For more information, see rdsadmin.list_databases.

6. Bring the database back online and apply additional transaction logs by calling `rdsadmin.rollforward_database`. For more information, see the section called "rdsadmin.rollforward_database".

```
db2 "call rdsadmin.rollforward_database(
    ?,
    'database_name',
    's3_bucket_name',
    s3_prefix,
    'rollforward_to_option',
    'complete_rollforward')"
```

7. (Optional) Check the status of the `rdsadmin.rollforward_database` stored procedure by calling the the section called "rdsadmin.rollforward_status" stored procedure.

8. If you set `complete_rollforward` to `FALSE` in the previous step, then you must finish bringing the database back online by calling `rdsadmin.complete_rollforward`. For more information, see the section called "rdsadmin.complete_rollforward".

```
db2 "call rdsadmin.complete_rollforward(
    ?,
    'database_name')"
```

9. (Optional) Check the status of the `rdsadmin.complete_rollforward` stored procedure by calling the the section called "rdsadmin.rollforward_status" stored procedure.

Listing databases

You can list all of your databases running on Amazon RDS for Db2 by calling the `rdsadmin.list_databases` user-defined function.

To list your databases

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. List your databases by calling `rdsadmin.list_databases`. For more information, see rdsadmin.list_databases.

```
db2 "select * from table(rdsadmin.list_databases())"
```

Collecting information about databases

To collect information about a database on a RDS for Db2 DB instance, call the `rdsadmin.db2pd_command` stored procedure. This information can help with monitoring your databases or troubleshooting issues.

To collect information about a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Collect information about the database by calling `rdsadmin.db2pd_command`. For more information, see rdsadmin.db2pd_command.

```
db2 "call rdsadmin.db2pd_command('db2pd_cmd')"
```

Forcing applications off of databases

To force applications off of a database on your RDS for Db2 DB instance, call the `rdsadmin.force_application` stored procedure. Before you perform maintenance on your databases, force applications off of your databases.

To force applications off of a database

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Force applications off of a database by calling `rdsadmin.force_application`. For more information, see rdsadmin.force_application.

   ```
   db2 "call rdsadmin.force_application(
       ?,
       'applications')"
   ```

Generating performance reports

You can generate performance reports with a procedure or a script. For information about using a procedure, see DBSUMMARY procedure - Generate a summary report of system and application performance metrics in the IBM Db2 documentation.

Db2 includes a `db2mon.sh` file in its `~sqllib/sample/perf` directory. Running the script produces a low-cost, extensive SQL metrics report. To download the `db2mon.sh` file and related script files, see the perf directory in the IBM db2-samples GitHub repository.

To generate performance reports with the script

1. Connect to your Db2 database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 connect to rdsadmin user master_username using master_password
```

2. Create a buffer pool named db2monbp with a page size of 4096 by calling `rdsadmin.create_bufferpool`. For more information, see rdsadmin.create_bufferpool.

```
db2 "call rdsadmin.create_bufferpool('database_name','db2monbp',4096)"
```

3. Create a temporary tablespace named db2montmptbsp that uses the db2monbp buffer pool by calling `rdsadmin.create_tablespace`. For more information, see rdsadmin.create_tablespace.

```
db2 "call rdsadmin.create_tablespace('database_name',\
    'db2montmptbsp','db2monbp',4096,1000,100,'T')"
```

4. Open the db2mon.sh script, and modify the line about connecting to a database.

   a. Remove the following line.

   ```
   db2 -v connect to $dbName
   ```

   b. Replace the line in the previous step with the following line. In the following example, replace *master_username* and *master_password* with the master username and master password for your RDS for Db2 DB instance.

   ```
   db2 -v connect to $dbName user master_username using master_password
   ```

   c. Remove the following lines.

   ```
   db2 -v create bufferpool db2monbp

   db2 -v create user temporary tablespace db2montmptbsp bufferpool db2monbp

   db2 -v drop tablespace db2montmptbsp

   db2 -v drop bufferpool db2monbp
   ```

5. Run the db2mon.sh script to output a report at specified intervals. In the following example, replace *absolute_path* with the complete path to the script file, *rds_database_alias* with the name of your database, and *seconds* with the number of seconds (0 to 3600) between report generation.

```
absolute_path/db2mon.sh rds_database_alias seconds | tee -a db2mon.out
```

Examples

The following example shows that the script file is located in the `perf` directory under the home directory.

```
/home/db2inst1/sqllib/samples/perf/db2mon.sh rds_database_alias seconds | tee -a
 db2mon.out
```

6. Drop the buffer pool and the tablespace that were created for the `db2mon.sh` file. In the following example, replace *master_username* and *master_password* with the master username and master password for your RDS for Db2 DB instance. Replace *database_name* with the name of your database. For more information, see rdsadmin.drop_tablespace and rdsadmin.drop_bufferpool.

```
db2 connect to rdsadmin user master_username using master_password

db2 "call rdsadmin.drop_tablespace('database_name','db2montmptbsp')"

db2 "call rdsadmin.drop_bufferpool('database_name','db2monbp')"
```

Managing storage

Db2 uses automatic storage to manage the physical storage for database objects such as tables, indexes, and temporary files. Instead of manually allocating storage space and keeping track of which storage paths are being used, automatic storage allows the Db2 system to create and manage storage paths as needed. This can simplify administration of Db2 databases and reduce the likelihood of errors due to human mistakes. For more information, see Automatic storage in the IBM Db2 documentation.

With RDS for Db2, you can dynamically increase the storage size with automatic expansion of the logical volumes and the file system. For more information, see Working with storage for Amazon RDS DB instances.

Common tasks for tablespaces

You can create, alter, rename, or drop tablespaces for an RDS for Db2 database. Creating, altering, renaming, or dropping tablespaces requires higher-level SYSADM authority, which isn't available to the master user. Instead, use Amazon RDS stored procedures.

Topics

- Creating a tablespace
- Altering a tablespace
- Renaming a tablespace
- Dropping a tablespace
- Checking the status of a tablespace
- Returning detailed information about tablespaces
- Listing the state and storage group for a tablespace
- Listing the tablespaces of a table
- Listing tablespace containers

Creating a tablespace

To create a tablespace for your RDS for Db2 database, call the `rdsadmin.create_tablespace` stored procedure. For more information, see CREATE TABLESPACE statement in the IBM Db2 documentation.

> **Important**
>
> To create a tablespace, you must have a buffer pool of the same page size to associate with the tablespace. For more information, see Common tasks for buffer pools.

To create a tablespace

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Create a tablespace by calling `rdsadmin.create_tablespace`. For more information, see rdsadmin.create_tablespace.

```
db2 "call rdsadmin.create_tablespace(
    'database_name',
    'tablespace_name',
    'buffer_pool_name',
    tablespace_initial_size,
    tablespace_increase_size,
    'tablespace_type')"
```

Altering a tablespace

To alter a tablespace for your RDS for Db2 database, call the `rdsadmin.alter_tablespace` stored procedure. You can use this stored procedure to change the buffer pool of a tablespace, lower the high water mark, or bring a tablespace online. For more information, see ALTER TABLESPACE statement in the IBM Db2 documentation.

To alter a tablespace

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

```
db2 "connect to rdsadmin user master_username using master_password"
```

2. Alter a tablespace by calling `rdsadmin.alter_tablespace`. For more information, see rdsadmin.alter_tablespace.

```
db2 "call rdsadmin.alter_tablespace(
    'database_name',
    'tablespace_name',
    'buffer_pool_name',
    buffer_pool_size,
    tablespace_increase_size,
    'max_size', 'reduce_max',
    'reduce_stop',
    'reduce_value',
    'lower_high_water',
    'lower_high_water_stop',
```

```
    'switch_online')"
```

Renaming a tablespace

To change the name of a tablespace for your RDS for Db2 database, call the `rdsadmin.rename_tablespace` stored procedure. For more information, see RENAME TABLESPACE statement in the IBM Db2 documentation.

To rename a tablespace

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Rename a tablespace by calling `rdsadmin.rename_tablespace`. For more information, including restrictions on what you can name a tablespace, see rdsadmin.rename_tablespace.

   ```
   db2 "call rdsadmin.rename_tablespace(
       'database_name',
       'source_tablespace_name',
       'target_tablespace_name')"
   ```

Dropping a tablespace

To drop a tablespace for your RDS for Db2 database, call the `rdsadmin.drop_tablespace` stored procedure. Before you drop a tablespace, first drop any objects in the tablespace such as tables, indexes, or large objects (LOBs). For more information, see Dropping table spaces in the IBM Db2 documentation.

To drop a tablespace

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 "connect to rdsadmin user master_username using master_password"
   ```

2. Drop a tablespace by calling `rdsadmin.drop_tablespace`. For more information, see rdsadmin.drop_tablespace.

```
db2 "call rdsadmin.drop_tablespace(
    'database_name',
    'tablespace_name')"
```

Checking the status of a tablespace

You can check the status of a tablespace by using the `cast` function.

To check the status of a tablespace

1. Connect to your Db2 database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *rds_database_alias*, *master_username*, and *master_password* with your own information.

```
db2 connect to rds_database_alias user master_username using master_password
```

2. Return a summary output.

   For a summary output:

```
db2 "select cast(tbsp_id as smallint) as tbsp_id,
cast(tbsp_name as varchar(35)) as tbsp_name,
cast(tbsp_type as varchar(3)) as tbsp_type,
cast(tbsp_state as varchar(10)) as state,
cast(tbsp_content_type as varchar(8)) as contents from
 table(mon_get_tablespace(null,-1)) order by tbsp_id"
```

Returning detailed information about tablespaces

You can return information about a tablespace for one member or all members by using the `cast` function.

To return detailed information about tablespaces

1. Connect to your Db2 database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *rds_database_alias*, *master_username*, and *master_password* with your own information.

```
db2 connect to rds_database_alias user master_username using master_password
```

2. Return details about all tablespaces in the database for one member or for all members.

   For one member:

```
db2 "select cast(member as smallint) as member,
cast(tbsp_id as smallint) as tbsp_id,
cast(tbsp_name as varchar(35)) as tbsp_name,
cast(tbsp_type as varchar(3)) as tbsp_type,
cast(tbsp_state as varchar(10)) as state,
cast(tbsp_content_type as varchar(8)) as contents,
cast(tbsp_total_pages as integer) as total_pages,
cast(tbsp_used_pages as integer) as used_pages,
cast(tbsp_free_pages as integer) as free_pages,
cast(tbsp_page_top as integer) as page_hwm,
cast(tbsp_page_size as integer) as page_sz,
cast(tbsp_extent_size as smallint) as extent_sz,
cast(tbsp_prefetch_size as smallint) as prefetch_sz,
cast(tbsp_initial_size as integer) as initial_size,
cast(tbsp_increase_size_percent as smallint) as increase_pct,
cast(storage_group_name as varchar(12)) as stogroup from
 table(mon_get_tablespace(null,-1)) order by member, tbsp_id "
```

   For all members:

```
db2 "select cast(member as smallint) as member
cast(tbsp_id as smallint) as tbsp_id,
cast(tbsp_name as varchar(35)) as tbsp_name,
cast(tbsp_type as varchar(3)) as tbsp_type,
cast(tbsp_state as varchar(10)) as state,
cast(tbsp_content_type as varchar(8)) as contents,
cast(tbsp_total_pages as integer) as total_pages,
cast(tbsp_used_pages as integer) as used_pages,
cast(tbsp_free_pages as integer) as free_pages,
cast(tbsp_page_top as integer) as page_hwm,
```

```
cast(tbsp_page_size as integer) as page_sz,
cast(tbsp_extent_size as smallint) as extent_sz,
cast(tbsp_prefetch_size as smallint) as prefetch_sz,
cast(tbsp_initial_size as integer) as initial_size,
cast(tbsp_increase_size_percent as smallint) as increase_pct,
cast(storage_group_name as varchar(12)) as stogroup from
 table(mon_get_tablespace(null,-2)) order by member, tbsp_id "
```

Listing the state and storage group for a tablespace

You can list the state and storage group for a tablespace by running a SQL statement.

To list the state and storage group for a tablespace, run the following SQL statement:

```
db2 "SELECT varchar(tbsp_name, 30) as tbsp_name,
                  varchar(TBSP_STATE, 30) state,
                  tbsp_type,
                  varchar(storage_group_name,30) storage_group
FROM TABLE(MON_GET_TABLESPACE('',-2)) AS t"
```

Listing the tablespaces of a table

You can list the tablespaces for a table by running a SQL statement.

To list the tablespaces of a table, run the following SQL statement. In the following example, replace *SCHEMA_NAME* and *TABLE_NAME* with the names of your schema and table:

```
db2 "SELECT
   VARCHAR(SD.TBSPACE,30) AS DATA_SPACE,
   VARCHAR(SL.TBSPACE,30) AS LONG_SPACE,
   VARCHAR(SI.TBSPACE,30) AS INDEX_SPACE
 FROM
   SYSCAT.DATAPARTITIONS P
   JOIN SYSCAT.TABLESPACES SD ON SD.TBSPACEID = P.TBSPACEID
   LEFT JOIN SYSCAT.TABLESPACES SL ON SL.TBSPACEID = P.LONG_TBSPACEID
   LEFT JOIN SYSCAT.TABLESPACES SI ON SI.TBSPACEID = P.INDEX_TBSPACEID
 WHERE
    TABSCHEMA = 'SCHEMA_NAME'
 AND TABNAME   = 'TABLE_NAME'"
```

Listing tablespace containers

You can list all tablespace containers or specific tablespace containers by using the `cast` command.

To list the tablespace containers for a tablespace

1. Connect to your Db2 database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`rds_database_alias`*, *`master_username`*, and *`master_password`* with your own information:

   ```
   db2 connect to rds_database_alias user master_username using master_password
   ```

2. Return a list of all tablespace containers in the database or specific tablespace containers.

   For all tablespace containers:

   ```
   db2 "select cast(member as smallint) as member,
   cast(tbsp_name as varchar(35)) as tbsp_name,
   cast(container_id as smallint) as id,
   cast(container_name as varchar(60)) as container_path, container_type as type from
    table(mon_get_container(null,-2)) order by member,tbsp_id,container_id"
   ```

   For specific tablespace containers:

   ```
   db2 "select cast(member as smallint) as member,
   cast(tbsp_name as varchar(35)) as tbsp_name,
   cast(container_id as smallint) as id,
   cast(container_name as varchar(60)) as container_path, container_type as type from
    table(mon_get_container('TBSP_1',-2)) order by member, tbsp_id,container_id"
   ```

Integrating an Amazon RDS for Db2 DB instance with Amazon S3

You can transfer files between your Amazon RDS for Db2 DB instance and an Amazon Simple Storage Service (Amazon S3) bucket with Amazon RDS stored procedures. For more information, see Amazon RDS for Db2 stored procedure reference.

> **Note**
>
> Your DB instance and your Amazon S3 bucket must be in the same AWS Region.

For RDS for Db2 to integrate with Amazon S3, your DB instance must have access to an Amazon S3 bucket where your RDS for Db2 resides. If you don't currently have an S3 bucket, create a bucket.

Topics

- Step 1: Create an IAM policy
- Step 2: Create an IAM role and attach your IAM policy
- Step 3: Add your IAM role to your RDS for Db2 DB instance

Step 1: Create an IAM policy

In this step, you create an AWS Identity and Access Management (IAM) policy with the permissions required to transfer files from your Amazon S3 bucket to your RDS DB instance. This step assumes that you have already created an S3 bucket. For more information, see Creating a bucket in the *Amazon S3 User Guide*.

Before you create the policy, note the following pieces of information:

- The Amazon Resource Name (ARN) for your bucket
- The ARN for your AWS Key Management Service (AWS KMS) key, if your bucket uses SSE-KMS or SSE-S3 encryption.

The IAM policy that you create should contain the following information. Replace *`{amzn-s3-demo-bucket}`* with the name of your S3 bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:GetObjectVersion",
                "s3:ListMultipartUploadParts",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::${amzn-s3-demo-bucket}/*",
                "arn:aws:s3:::${amzn-s3-demo-bucket}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

You can create an IAM policy by using the AWS Management Console or the AWS Command Line Interface (AWS CLI).

Console

To create an IAM policy to allow Amazon RDS to access your Amazon S3 bucket

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**, and then choose **JSON**.
4. Add actions by service. To transfer files from an Amazon S3 bucket to Amazon RDS, you must select bucket permissions and object permissions.
5. Expand **Resources**. You must specify your bucket and object resources.
6. Choose **Next**.
7. For **Policy name**, enter a name for this policy.
8. (Optional) For **Description**, enter a description for this policy.
9. Choose **Create policy**.

AWS CLI

To create an IAM policy to allow Amazon RDS to access your Amazon S3 bucket

1. Create a JSON file that contains the following JSON policy document. Replace *{amzn-s3-demo-bucket}* with the name of your S3 bucket.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt",
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
```

```
                "s3:ListBucket",
                "s3:GetObjectVersion",
                "s3:ListMultipartUploadParts",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::${amzn-s3-demo-bucket}/*",
                "arn:aws:s3:::${amzn-s3-demo-bucket}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

2. Run the create-policy command. In the following example, replace *iam_policy_name* and *iam_policy_file_name* with a name for your IAM policy and the name of the JSON file you created in Step 1.

   For Linux, macOS, or Unix:

```
aws iam create-policy \
    --policy-name iam_policy_name \
    --policy-document '{
      "Version": "2012-10-17"                    ,
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt",
            "s3:PutObject",
            "s3:GetObject",
            "s3:AbortMultipartUpload",
```

```
            "s3:ListBucket",
            "s3:DeleteObject",
            "s3:GetObjectVersion",
            "s3:ListMultipartUploadParts"
          ],
          "Resource": [
            "arn:aws:s3:::s3_bucket_name/*",
            "arn:aws:s3:::s3_bucket_name"
          ]
        }
      ]
    }'
```

For Windows:

```
aws iam create-policy ^
    --policy-name iam_policy_name ^
    --policy-document '{
      "Version": "2012-10-17"                    ,
      "Statement": [
        {
          "Effect": "Allow",
            "Action": [
              "s3:PutObject",
              "s3:GetObject",
              "s3:AbortMultipartUpload",
              "s3:ListBucket",
              "s3:DeleteObject",
              "s3:GetObjectVersion",
              "s3:ListMultipartUploadParts"
            ],
            "Resource": [
              "arn:aws:s3:::s3_bucket_name/*",
              "arn:aws:s3:::s3_bucket_name"
            ]
        }
      ]
    }'
```

3. After the policy is created, note the ARN of the policy. You need the ARN for [Step 2: Create an IAM role and attach your IAM policy].

For information about creating an IAM policy, see Creating IAM policies in the IAM User Guide.

Step 2: Create an IAM role and attach your IAM policy

This step assumes that you have created the IAM policy in Step 1: Create an IAM policy. In this step, you create a IAM role for your RDS for Db2 DB instance and then attach your IAM policy to the role.

You can create an IAM role for your DB instance by using the AWS Management Console or the AWS CLI.

Console

To create an IAM role and attach your IAM policy to it

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. For **Trusted entity type**, select **AWS service**.
5. For **Service or use case**, select **RDS**, and then select **RDS – Add Role to Database**.
6. Choose **Next**.
7. For **Permissions policies**, search for and select the name of the IAM policy that you created.
8. Choose **Next**.
9. For **Role name**, enter a role name.
10. (Optional) For **Description**, enter a description for the new role.
11. Choose **Create role**.

AWS CLI

To create an IAM role and attach your IAM policy to it

1. Create a JSON file that contains the following JSON policy document:

   JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Run the create-role command. In the following example, replace *iam_role_name* and *iam_assume_role_policy_file_name* with a name for your IAM role and the name of the JSON file that you created in Step 1.

   For Linux, macOS, or Unix:

```
aws iam create-role \
    --role-name iam_role_name \
    --assume-role-policy-document '{
      "Version": "2012-10-17"                    ,
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "rds.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }'
```

   For Windows:

```
aws iam create-role ^
    --role-name iam_role_name ^
    --assume-role-policy-document '{
      "Version": "2012-10-17"                    ,
      "Statement": [
        {
```

```
          "Effect": "Allow",
          "Principal": {
            "Service": "rds.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }'
```

3. After the role is created, note the ARN of the role. You need the ARN for Step 3: Add your IAM role to your RDS for Db2 DB instance.
4. Run the attach-role-policy command. In the following example, replace *iam_policy_arn* with the ARN of the IAM policy that you created in Step 1: Create an IAM policy. Replace *iam_role_name* with the name of the IAM role that you just created.

   For Linux, macOS, or Unix:

   ```
   aws iam attach-role-policy \
      --policy-arn iam_policy_arn \
      --role-name iam_role_name
   ```

   For Windows:

   ```
   aws iam attach-role-policy ^
      --policy-arn iam_policy_arn ^
      --role-name iam_role_name
   ```

For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.

Step 3: Add your IAM role to your RDS for Db2 DB instance

In this step, you add your IAM role to your RDS for Db2 DB instance. Note the following requirements:

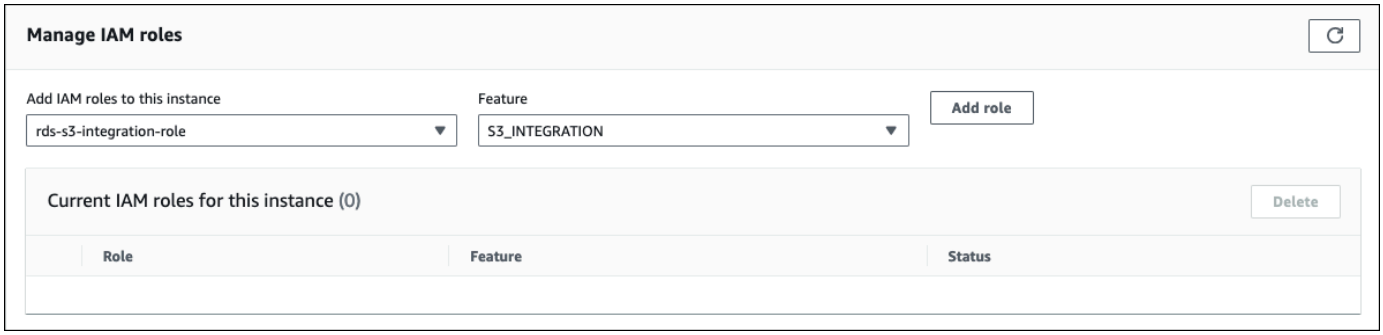
- You must have access to an IAM role with the required Amazon S3 permissions policy attached to it.
- You can only associate one IAM role with your RDS for Db2 DB instance at a time.
- Your RDS for Db2 DB instance must be in the **Available** state.

You can add an IAM role to your DB instance by using the AWS Management Console or the AWS CLI.

Console

To add an IAM role to your RDS for Db2 DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose your RDS for Db2 DB instance name.
4. On the **Connectivity & security** tab, scroll down to the **Manage IAM roles** section at the bottom of the page.
5. For **Add IAM roles to this instance**, choose the role that you created in Step 2: Create an IAM role and attach your IAM policy.
6. For **Feature**, choose **S3_INTEGRATION**.
7. Choose **Add role**.



AWS CLI

To add an IAM role to your RDS for Db2 DB instance, run the add-role-to-db-instance command. In the following example, replace *region*, *db_instance_name*, and *iam_role_arn* with the name of the AWS Region where your DB instance exists, the name of your DB instance, and the ARN of the IAM role that you created in Step 2: Create an IAM role and attach your IAM policy.

For Linux, macOS, or Unix:

```
aws rds add-role-to-db-instance \
    --region $region \
    --db-instance-identifier $db_instance_name \
```

```
    --feature-name S3_INTEGRATION \
    --role-arn $iam_role_arn \
```

For Windows:

```
aws rds add-role-to-db-instance ^
    --region $region \
    --db-instance-identifier db_instance_name ^
    --feature-name S3_INTEGRATION ^
    --role-arn iam_role_arn ^
```

To confirm that the role was successfully added to your RDS for Db2 DB instance, run the describe-db-instances command. In the following example, replace *db_instance_name* with the name of your DB instance.

For Linux, macOS, or Unix:

```
aws rds describe-db-instances \
    --filters "Name=db-instance-id,Values=db_instance_name" \
    --query 'DBInstances[].AssociatedRoles'
```

For Windows:

```
aws rds describe-db-instances ^
    --filters "Name=db-instance-id,Values=db_instance_name" ^
    --query 'DBInstances[].AssociatedRoles'
```

This command produces output similar to the following example:

```
[
    [
        {
            "RoleArn": "arn:aws:iam::0123456789012:role/rds-db2-s3-role",
            "FeatureName": "S3_INTEGRATION",
            "Status": "ACTIVE"
        }
    ]
]
```

Migrating data to Amazon RDS for Db2

You can migrate self-managed Db2 databases to Amazon RDS for Db2 by using either AWS or native Db2 tools.

For information about migrating from your Db2 database to Amazon RDS for Db2 using AWS services, see Using AWS services to migrate data from Db2 to Amazon RDS for Db2.

For information about migrating from your Db2 database to Amazon RDS for Db2 using native Db2 tools, see Using native Db2 tools to migrate data from Db2 to Amazon RDS for Db2.

Using AWS services to migrate data from Db2 to Amazon RDS for Db2

In Amazon RDS, there are several ways you can migrate data from a Db2 database to Amazon RDS for Db2. You can perform a one-time migration of your Db2 database from Linux, AIX, or Windows environments to Amazon RDS for Db2. To minimize downtime, you can perform a near-zero downtime migration. You can migrate your data by saving it to Amazon S3 and loading it one table at a time into your Db2 database. You can also perform a synchronous migration through replication or use AWS Database Migration Service.

For one-time migrations for Linux-based Db2 databases, Amazon RDS only supports offline and online backups. Amazon RDS doesn't support incremental and Delta backups. For near-zero downtime migrations for Linux-based Db2 databases, Amazon RDS requires online backups. We recommend that you use online backups for near-zero downtime migrations and offline backups for migrations that can handle downtime.

Topics

- Migrating from Linux to Linux for Amazon RDS for Db2
- Migrating from Linux to Linux with near-zero downtime for Amazon RDS for Db2
- Migrating synchronously from Linux to Linux for Amazon RDS for Db2
- Migrating from AIX or Windows to Linux for Amazon RDS for Db2
- Migrating Db2 data through Amazon S3 to Amazon RDS for Db2
- Migrating to Amazon RDS for Db2 with AWS Database Migration Service (AWS DMS)

Migrating from Linux to Linux for Amazon RDS for Db2

With this migration approach, you back up your self-managed Db2 database to an Amazon S3 bucket. Then, you use Amazon RDS stored procedures to restore your Db2 database to an Amazon

RDS for Db2 DB instance. For more information about using Amazon S3, see Integrating an Amazon RDS for Db2 DB instance with Amazon S3.

Backup and restore for RDS for Db2 follows the IBM Db2 supported upgrade paths and restrictions. For more information, see Supported upgrade paths for Db2 servers and Upgrade restrictions for Db2 servers in the IBM Db2 documentation.

Topics

- Limitations and recommendations for using native restore
- Backing up your database to Amazon S3
- Creating a default automatic storage group
- Restoring your Db2 database

Limitations and recommendations for using native restore

The following limitations and recommendations apply to using native restore:

- Amazon RDS only supports migrating on-premises versions of Db2 that match supported RDS for Db2 versions. For more information about the supported versions, see Upgrade management for Amazon RDS Db2 instances.
- Amazon RDS only supports offline and online backups for native restore. Amazon RDS doesn't support incremental or Delta backups.
- You can't restore from an Amazon S3 bucket in an AWS Region that is different from the Region where your RDS for Db2 DB instance is located.
- Amazon S3 limits the size of files that are uploaded to an Amazon S3 bucket to 5 TB. If your database backup file exceeds 5 TB, then split the backup file into smaller files.
- Amazon RDS doesn't support non-fenced external routines, incremental restores, or Delta restores.
- You can't restore from an encrypted source database, but you can restore to an encrypted Amazon RDS DB instance.

The restoration process differs depending on your configuration.

If you set `USE_STREAMING_RESTORE` to `TRUE`, Amazon RDS directly streams your backup from your S3 bucket during restoration. Streaming significantly reduces storage requirements. You only

need to provision storage space equal to or greater than either the size of the backup or the size of the original database, whichever is larger.

If you set `USE_STREAMING_RESTORE` to `FALSE`, Amazon RDS first downloads the backup to your RDS for Db2 DB instance and then extracts the backup. Extraction requires additional storage space. You must provision storage space equal to or greater than the sum of the size of the backup plus the size of the original database.

The maximum size of the restored database equals the maximum supported database size minus any space required for temporary storage during the restoration process.

Backing up your database to Amazon S3

To back up your database on Amazon S3, you need the following AWS components:

- *An Amazon S3 bucket to store your backup files*: Upload any backup files that you want to migrate to Amazon RDS. We recommend that you use offline backups for migrations that can handle downtime. If you already have an S3 bucket, you can use that bucket. If you don't have an S3 bucket, see Creating a bucket in the *Amazon S3 User Guide*.

  > **Note**
  >
  > If your database is large and would take a long time to transfer to an S3 bucket, you can order an AWS Snow Family device and ask AWS to perform the backup. After you copy your files to the device and return it to the Snow Family team, the team transfers your backed-up images to your S3 bucket. For more information, see the AWS Snow Family documentation.

- *An IAM role to access the S3 bucket*: If you already have an IAM role, you can use that role. If you don't have a role, see Step 2: Create an IAM role and attach your IAM policy.
- *An IAM policy with trust relationships and permissions attached to your IAM role*: For more information, see Step 1: Create an IAM policy.
- *The IAM role added to your RDS for Db2 DB instance*: For more information, see Step 3: Add your IAM role to your RDS for Db2 DB instance.

Creating a default automatic storage group

Your source database must have a default automatic storage group. If your database doesn't have a default automatic storage group, you must create one.

To create a default automatic storage group

1. Connect to your source database. In the following example, replace *source_database* with the name of your database.

```
db2 connect to source_database
```

2. Create an automatic storage group and set it as the default. In the following example, replace *storage_path* with the absolute path to where the storage group is located.

```
db2 "create stogroup IBMSTOGROUP ON storage_path set as default"
```

3. Terminate backend processes.

```
db2 terminate
```

4. Deactivate the database and stop all database services. In the following example, replace *source_database* with the name of the database that you created the storage group for.

```
db2 deactivate db source_database
```

5. Back up the database. In the following example, replace *source_database* with the name of the database that you created the storage group for. Replace *file_system_path* with the absolute path to where you want to back up the database.

```
db2 backup database source_database to file_system_path
```

Restoring your Db2 database

After you back up your database on Amazon S3 and create an automatic storage group, you are ready to restore your Db2 database to your RDS for Db2 DB instance.

To restore your Db2 database from your Amazon S3 bucket to your RDS for Db2 DB instance

1. Connect to your RDS for Db2 DB instance. For more information, see Connecting to your Db2 DB instance.
2. (Optional) To ensure that your database is configured with the optimal settings, check the values for the following parameters by calling the section called "rdsadmin.show_configuration":

- RESTORE_DATABASE_NUM_BUFFERS
- RESTORE_DATABASE_PARALLELISM
- RESTORE_DATABASE_NUM_MULTI_PATHS
- USE_STREAMING_RESTORE

Use the section called "rdsadmin.set_configuration" to modify these values as needed. Properly configuring these parameters can significantly improve performance when restoring databases with large volumes of data. For most migration scenarios, we recommend setting USE_STREAMING_RESTORE to TRUE because it reduces storage requirements and can improve restoration speed.

3. Restore your database by calling `rdsadmin.restore_database`. For more information, see rdsadmin.restore_database.

Migrating from Linux to Linux with near-zero downtime for Amazon RDS for Db2

With this migration approach, you migrate a Linux-based Db2 database from one self-managed Db2 database (source) to Amazon RDS for Db2. This approach results in minimal to no outage or downtime for the application or users. This approach backs up your database and restores it with log replay, which helps prevent disruptions to ongoing operations and provides high availability of your database.

To achieve near-zero downtime migration, RDS for Db2 implements restore with log replay. This approach takes a backup of your self-managed Linux-based Db2 database and restores it on the RDS for Db2 server. With Amazon RDS stored procedures, you then apply subsequent transaction logs to bring the database up to date.

Topics

- Limitations and recommendations for near-zero downtime migration
- Backing up your database to Amazon S3
- Creating a default automatic storage group
- Migrating your Db2 database

Limitations and recommendations for near-zero downtime migration

The following limitations and recommendations apply to using near-zero downtime migration:

- Amazon RDS requires an online backup for near-zero downtime migration. This is because Amazon RDS keeps your database in a rollforward pending state as you upload your archived transaction logs. For more information, see the section called “Migrating your Db2 database”.
- You can't restore from an Amazon S3 bucket in an AWS Region that is different from the Region where your RDS for Db2 DB instance is located.
- Amazon S3 limits the size of files uploaded to an S3 bucket to 5 TB. If your database backup file exceeds 5 TB, then split the backup file into smaller files.
- Amazon RDS doesn't support non-fenced external routines, incremental restores, or Delta restores.
- You can't restore from an encrypted source database, but you can restore to an encrypted Amazon RDS DB instance.

The restoration process differs depending on your configuration.

If you set `USE_STREAMING_RESTORE` to `TRUE`, Amazon RDS directly streams your backup from your S3 bucket during restoration. Streaming significantly reduces storage requirements. You only need to provision storage space equal to or greater than either the size of the backup or the size of the original database, whichever is larger.

If you set `USE_STREAMING_RESTORE` to `FALSE`, Amazon RDS first downloads the backup to your RDS for Db2 DB instance and then extracts the backup. Extraction requires additional storage space. You must provision storage space equal to or greater than the sum of the size of the backup plus the size of the original database.

The maximum size of the restored database equals the maximum supported database size minus any space required for temporary storage during the restoration process.

Backing up your database to Amazon S3

To back up your database on Amazon S3, you need the following AWS components:

- *An Amazon S3 bucket to store your backup files*: Upload any backup files that you want to migrate to Amazon RDS. Amazon RDS requires an online backup for near-zero downtime migration. If you already have an S3 bucket, you can use that bucket. If you don't have an S3 bucket, see Creating a bucket in the *Amazon S3 User Guide*.

> **Note**
>
> If your database is large and would take a long time to transfer to an S3 bucket, you can order an AWS Snow Family device and ask AWS to perform the backup. After you copy your files to the device and return it to the Snow Family team, the team transfers your backed-up images to your S3 bucket. For more information, see the AWS Snow Family documentation.

- *An IAM role to access the S3 bucket*: If you already have an AWS Identity and Access Management (IAM) role, you can use that role. If you don't have a role, see Step 2: Create an IAM role and attach your IAM policy.
- *An IAM policy with trust relationships and permissions attached to your IAM role*: For more information, see Step 1: Create an IAM policy.
- *The IAM role added to your RDS for Db2 DB instance*: For more information, see Step 3: Add your IAM role to your RDS for Db2 DB instance.

Creating a default automatic storage group

Your source database must have a default automatic storage group. If your database doesn't have a default automatic storage group, you must create one.

To create a default automatic storage group

1. Connect to your source database. In the following example, replace *source_database* with the name of your database.

```
db2 connect to source_database
```

2. Create an automatic storage group and set it as the default. In the following example, replace *storage_path* with the absolute path to where the storage group is located.

```
db2 "create stogroup IBMSTOGROUP ON storage_path set as default"
```

3. Terminate backend processes.

```
db2 terminate
```

Migrating your Db2 database

After you set up for near-zero downtime migration, you are ready to migrate your Db2 database from your Amazon S3 bucket to your RDS for Db2 DB instance.

To perform a near-zero downtime migration of backup files from your Amazon S3 bucket to your RDS for Db2 DB instance

1. Perform an online backup of your source database. For more information, see BACKUP DATABASE command in the IBM Db2 documentation.
2. Copy the backup of your database to an Amazon S3 bucket. For information about using Amazon S3, see the Amazon Simple Storage Service User Guide.
3. Connect to the `rdsadmin` server with the *`master_username`* and *`master_password`* for your RDS for Db2 DB instance.

   ```
   db2 connect to rdsadmin user master_username using master_password
   ```

4. (Optional) To ensure that your database is configured with the optimal settings, check the values for the following parameters by calling the section called "rdsadmin.show_configuration":

   - `RESTORE_DATABASE_NUM_BUFFERS`
   - `RESTORE_DATABASE_PARALLELISM`
   - `RESTORE_DATABASE_NUM_MULTI_PATHS`
   - `USE_STREAMING_RESTORE`

   Use the section called "rdsadmin.set_configuration" to modify these values as needed. Properly configuring these parameters can significantly improve performance when restoring databases with large volumes of data. For most migration scenarios, we recommend setting `USE_STREAMING_RESTORE` to `TRUE` because it reduces storage requirements and can improve restoration speed.
5. Restore the backup on the RDS for Db2 server by calling `rdsadmin.restore_database`. Set `backup_type` to `ONLINE`. For more information, see rdsadmin.restore_database.
6. Copy your archive logs from your source server to your S3 bucket. For more information, see Archive logging in the IBM Db2 documentation.

7. Apply archive logs as many times as needed by calling `rdsadmin.rollforward_database`. Set `complete_rollforward` to `FALSE` to keep the database in a `ROLL-FORWARD PENDING` state. For more information, see rdsadmin.rollforward_database.
8. After you apply all of the archive logs, bring the database online by calling `rdsadmin.complete_rollforward`. For more information, see rdsadmin.complete_rollforward.
9. Switch application connections to the RDS for Db2 server by either updating your application endpoints for the database or by updating the DNS endpoints to redirect traffic to the RDS for Db2 server. You can also use the Db2 automatic client reroute feature on your self-managed Db2 database with the RDS for Db2 database endpoint. For more information, see Automatic client reroute description and setup in the IBM Db2 documentation.
10. (Optional) Shut down your source database.

Migrating synchronously from Linux to Linux for Amazon RDS for Db2

With this migration approach, you set up replication between your self-managed Db2 database and your Amazon RDS for Db2 DB instance. Changes made to the self-managed database replicates to the RDS for Db2 DB instance in near real-time. This approach can provide continuous availability and minimize downtime during the migration process.

Migrating from AIX or Windows to Linux for Amazon RDS for Db2

With this migration approach, you use native Db2 tools to back up your self-managed Db2 database to an Amazon S3 bucket. Native Db2 tools include the `export` utility, the `db2move` system command, or the `db2look` system command. Your Db2 database can either be self-managed or in Amazon Elastic Compute Cloud (Amazon EC2). You can move data from your AIX or Windows system to your Amazon S3 bucket. Then, use a Db2 client to load data directly from the S3 bucket to your Amazon RDS for Db2 database. Downtime depends on the size of your database. For more information about using Amazon S3, see Integrating an Amazon RDS for Db2 DB instance with Amazon S3.

To migrate your Db2 database to RDS for Db2

1. Prepare to back up your database. Configure sufficient storage amount to hold the backup on your self-managed Db2 system.
2. Back up your database.

a. Run the db2look system command to extract the data definition language (DDL) file for all objects.

b. Run either the Db2 export utility, the db2move system command, or a CREATE EXTERNAL TABLE statement to unload the Db2 table data to storage on your Db2 system.

3. Move your backup to an Amazon S3 bucket. For more information, see Integrating an Amazon RDS for Db2 DB instance with Amazon S3.

> **Note**
>
> If your database is large and would take a long time to transfer to an S3 bucket, you can order an AWS Snow Family device and ask AWS to perform the backup. After you copy your files to the device and return it to the Snow Family team, the team transfers your backed-up images to your S3 bucket. For more information, see the AWS Snow Family documentation.

4. Use a Db2 client to load data directly from your S3 bucket to your RDS for Db2 database. For more information, see Migrating with Amazon S3.

Migrating Db2 data through Amazon S3 to Amazon RDS for Db2

With this migration approach, you first save data from a single table into a data file that you place in an Amazon S3 bucket. Then, you use the LOAD command to load the data from that data file into a table in your Amazon RDS for Db2 database. For more information about using Amazon S3, see Integrating an Amazon RDS for Db2 DB instance with Amazon S3.

Topics

- Saving your data to Amazon S3
- Loading your data into RDS for Db2 tables

Saving your data to Amazon S3

To save data from a single table to Amazon S3, use a database utility to extract the data from your database management system (DBMS) into a CSV file. Then, upload the data file to Amazon S3.

For storing data files on Amazon S3, you need the following AWS components:

- *An Amazon S3 bucket to store your backup files*: If you already have an S3 bucket, you can use that bucket. If you don't have an S3 bucket, see Creating a bucket in the *Amazon S3 User Guide*.
- *An IAM role to access the S3 bucket*: If you already have an IAM role, you can use that role. If you don't have a role, see Step 2: Create an IAM role and attach your IAM policy.
- *An IAM policy with trust relationships and permissions attached to your IAM role*: For more information, see Step 1: Create an IAM policy.
- *The IAM role added to your RDS for Db2 DB instance*: For more information, see Step 3: Add your IAM role to your RDS for Db2 DB instance.

Loading your data into RDS for Db2 tables

After you save your data files to Amazon S3, you can load the data from these files into individual tables on your RDS for Db2 DB instance.

To load your Db2 table data into your RDS for Db2 DB database table

1. Connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 connect to rdsadmin user master_username using master_password
   ```

2. Catalog a storage access alias that points to the Amazon S3 bucket where your saved files are stored. Take note of the name of this alias for use in the next step. You only need to perform this step once if you plan to load multiple tables from data files stored in the same Amazon S3 bucket.

   The following example catalogs an alias named *`my_s3_alias`* that grants a user named *`jorge_souza`* access to a bucket named *`amzn-s3-demo-bucket`*.

   ```
   db2 "call rdsadmin.catalog_storage_access(?, 'my_s3_alias', 'amzn-s3-demo-bucket',
    'USER', 'jorge_souza')"
   ```

   For more information about this stored procedure, See the section called "rdsadmin.catalog_storage_access".

3. Run the LOAD command using the storage access alias that points to your Amazon S3 bucket.

> **Note**
>
> If the LOAD command returns an error, then you might need to create a VPC gateway endpoint for Amazon S3 and add outbound rules to the security group. For more information, see the section called “File I/O error”.

The following example loads data from a data file named *my_s3_datafile.csv* into a table named *my_db2_table*. The example assumes that the data file is in the Amazon S3 bucket that the alias named *my_s3_alias* points to.

```
db2 "load from db2remote://my_s3_alias//my_s3_datafile.csv of DEL insert
 into my_db2_table";
```

The following example loads LOBs from a data file named *my_table1_export.ixf* into a table named *my_db2_table*. The example assumes that the data file is in the Amazon S3 bucket that the alias named *my_s3_alias* points to.

```
db2 "call sysproc.admin_cmd('load from
 "db2remote://my_s3_alias//my_table1_export.ixf" of ixf
        lobs from "db2remote://my_s3_alias//" xml from "db2remote://my_s3_alias//"
        modified by lobsinfile implicitlyhiddeninclude identityoverride
 generatedoverride periodoverride transactionidoverride
        messages on server
        replace into "my_schema"."my_db2_table"
                              nonrecoverable
        indexing mode incremental allow no access')"
```

Repeat this step for each data file in the Amazon S3 bucket that you want to load into a table in your RDS for Db2 DB instance.

For more information about the LOAD command, see LOAD command.

Migrating to Amazon RDS for Db2 with AWS Database Migration Service (AWS DMS)

You can use AWS DMS for one-time migrations and then synchronize from Db2 on Linux, Unix (such as AIX), and Windows to Amazon RDS for Db2. For more information, see What is AWS Database Migration Service?.

Using native Db2 tools to migrate data from Db2 to Amazon RDS for Db2

You can use several native Db2 tools, utilities, and commands to move data directly from a Db2 database to an Amazon RDS for Db2 database. To use these native Db2 tools, you must be able to connect your client machine to an RDS for Db2 DB instance. For more information, see Connecting a client machine to an Amazon RDS for Db2 DB instance.

> **Note**
>
> Another way to move your data is to first save it to an Amazon S3 bucket, and then use the `LOAD` command to transfer that data into a table in your RDS for Db2 database. This method provides the best performance when migrating a large amount of data because of good network connectivity between RDS for Db2 and S3. For more information, see the section called "Migrating with Amazon S3".

Tool name	Use case	Limitations
db2look	Copying metadata from a self-managed Db2 database to an RDS for Db2 database.	• You must modify the syntax for creating buffer pools, creating tablespaces, and creating roles to match the syntax used by the RDS for Db2 stored procedures.
IMPORT command	Migrating small tables and tables with large objects (LOBs) from a client machine	• Slower than the `LOAD` utility because of `INSERT` and `DELETE` logging operations.

Tool name	Use case	Limitations
	to the RDS for Db2 DB instance.	• Poor performance with limited network bandwidth.
INGEST utility	Continually streaming data from files and pipes *without* large objects (LOBs) on the client machine to the RDS for Db2 DB instance. Supports `INSERT` and `MERGE` operation s.	• Can't stream data files that contain LOBs. Use the `IMPORT` command instead. • Connectivity required between self-managed Db2 database and RDS for Db2 database.
INSERT command	Copying data in small tables from a self-managed Db2 database to an RDS for Db2 database.	• Connectivity required between self-managed Db2 database and RDS for Db2 database. • Poor performance with limited network bandwidth.
LOAD CLIENT command	Migrating small tables *without* large objects (LOBs) from a client machine to the RDS for Db2 DB instance.	• Can't migrate data files that contain LOBs. Use the `IMPORT` command instead. • Poor performance with limited network bandwidth.

Connecting a client machine to an Amazon RDS for Db2 DB instance

To use any of the native Db2 tools to move data from a Db2 database to an Amazon RDS for Db2 database, you must first connect your client machine to an RDS for Db2 DB instance.

The client machine can be any of the following:

- An Amazon Elastic Compute Cloud (Amazon EC2) instance on Linux, Windows, or macOS. This instance should be in the same virtual private cloud (VPC) as your RDS for Db2 DB instance, AWS Cloud9, or AWS CloudShell.

- A self-managed Db2 instance in an Amazon EC2 instance. The instances should be in the same VPC.
- A self-managed Db2 instance in an Amazon EC2 instance. The instances can be in different VPCs if you enabled VPC peering. For more information, see Create a VPC peering connection in the *Amazon Virtual Private Cloud VPC Peering Guide*.
- A local machine running Linux, Windows, or macOS in a self-managed environment. You must either have public connectivity to RDS for Db2 or enable VPN connectivity between self-managed Db2 instances and AWS.

To connect your client machine to your RDS for Db2 DB instance, log in to your client machine with IBM Db2 Data Management Console. For more information, see Creating an Amazon RDS DB instance and IBM Db2 Data Management Console.

You can use AWS Database Migration Service (AWS DMS) to run queries against the database, run an SQL execution plan, and monitor the database. For more information, see What is AWS Database Migration Service? in the *AWS Database Migration Service User Guide*.

After you successfully connect your client machine to your RDS for Db2 DB instance, you are ready to use any native Db2 tool to copy data. For more information, see Using native Db2 tools to migrate data from Db2 to Amazon RDS for Db2.

Copying database metadata from Db2 to Amazon RDS for Db2 with db2look

`db2look` is a native Db2 tool that extracts data definition language (DDL) files, objects, authorizations, configurations, WLM, and database layouts. You can use `db2look` to copy database metadata from a self-managed Db2 database to an Amazon RDS for Db2 database. For more information, see Mimicking databases using db2look in the IBM Db2 documentation.

To copy the database metadata

1. Run the `db2look` tool on your self-managed Db2 system to extract the DDL file. In the following example, replace *`database_name`* with the name of your Db2 database.

   ```
   db2look -d database_name -e -l -a -f -wlm -cor -createdb -printdbcfg -o db2look.sql
   ```

2. If your client machine has access to the source (self-managed Db2) database and the RDS for Db2 DB instance, you can create the `db2look.sql` file on the client machine by directly attaching to the remote instance. Then catalog the remote self-managed Db2 instance.

a. Catalog the node. In the following example, replace *dns_ip_address* and *port* with the DNS name or the IP address and the port number of the self-managed Db2 database.

```
db2 catalog tcpip node srcnode REMOTE dns_ip_address server port
```

b. Catalog the database. In the following example, replace *source_database_name* and *source_database_alias* with the name of the self-managed Db2 database and the alias that you want to use for this database.

```
db2 catalog database source_database_name as source_database_alias at node
 srcnode \
    authentication server_encrypt
```

c. Attach to the source database. In the following example, replace *source_database_alias*, *user_id*, and *user_password* with the alias that you created in the previous step and the user ID and password for the self-managed Db2 database.

```
db2look -d source_database_alias -i user_id -w user_password -e -l -a -f -wlm \
    -cor -createdb -printdbcfg -o db2look.sql
```

3. If you can't access the remote self-managed Db2 database from the client machine, copy the `db2look.sql` file to the client machine. Then catalog the RDS for Db2 DB instance.

a. Catalog the node. In the following example, replace *dns_ip_address* and *port* with the DNS name or the IP address and the port number of the RDS for Db2 DB instance.

```
db2 catalog tcpip node remnode REMOTE dns_ip_address server port
```

b. Catalog the database. In the following example, replace *rds_database_name* and *rds_database_alias* with the name of the RDS for Db2 database and the alias that you want to use for this database.

```
db2 catalog database rds_database_name as rds_database_alias at node remnode \
    authentication server_encrypt
```

c. Catalog the admin database that manages RDS for Db2. You can't use this database to store any data.

```
db2 catalog database rdsadmin as rdsadmin at node remnode authentication
 server_encrypt
```

4. Create buffer pools and tablespaces. The administrator doesn't have privileges to create buffer pools or tablespaces. However, you can use Amazon RDS stored procedures to create them.

   a. Find the names and definitions of the buffer pools and tablespaces in the `db2look.sql` file.

   b. Connect to Amazon RDS using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

   ```
   db2 connect to rdsadmin user master_username using master_password
   ```

   c. Create a buffer pool by calling `rdsadmin.create_bufferpool`. For more information, see rdsadmin.create_bufferpool.

   ```
   db2 "call rdsadmin.create_bufferpool(
       'database_name',
       'buffer_pool_name',
       buffer_pool_size,
       'immediate',
       'automatic',
       page_size,
       number_block_pages,
       block_size)"
   ```

   d. Create a tablespace by calling `rdsadmin.create_tablespace`. For more information, see rdsadmin.create_tablespace.

   ```
   db2 "call rdsadmin.create_tablespace(
       'database_name',
       'tablespace_name',
       'buffer_pool_name',
       tablespace_initial_size,
       tablespace_increase_size,
       'tablespace_type')"
   ```

   e. Repeat steps c or d for each additional buffer pool or tablespace that you want to add.

   f. Terminate your connection.

```
db2 terminate
```

5. Create tables and objects.

   a. Connect to your RDS for Db2 database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *rds_database_name*, *master_username*, and *master_password* with your own information.

   ```
   db2 connect to rds_database_name user master_username using master_password
   ```

   b. Run the `db2look.sql` file.

   ```
   db2 -tvf db2look.sql
   ```

   c. Terminate your connection.

   ```
   db2 terminate
   ```

Importing data from a client machine to Amazon RDS for Db2 with the IMPORT command

You can use the `IMPORT` command from a client machine to import your data into the Amazon RDS for Db2 server.

> **Important**
>
> The `IMPORT` command method is useful for migrating small tables and tables that include large objects (LOBs). The `IMPORT` command is slower than the `LOAD` utility because of the `INSERT` and `DELETE` logging operations. If your network bandwidth between the client machine and RDS for Db2 is limited, we recommend that you use a different migration approach. For more information, see Using native Db2 tools to migrate data from Db2 to Amazon RDS for Db2.

To import data into the RDS for Db2 server

1. Log in to your client machine with IBM Db2 Data Management Console. For more information, see Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 Data Management Console.
2. Catalog the RDS for Db2 database on the client machine.

   a. Catalog the node. In the following example, replace *dns_ip_address* and *port* with the DNS name or the IP address and the port number of the self-managed Db2 database.

   ```
   db2 catalog tcpip node srcnode REMOTE dns_ip_address server port
   ```

   b. Catalog the database. In the following example, replace *source_database_name* and *source_database_alias* with the name of the self-managed Db2 database and the alias that you want to use for this database.

   ```
   db2 catalog database source_database_name as source_database_alias at node
    srcnode \
       authentication server_encrypt
   ```

3. Attach to the source database. In the following example, replace *source_database_alias*, *user_id*, and *user_password* with the alias you created in the previous step and the user ID and password for the self-managed Db2 database.

   ```
   db2look -d source_database_alias -i user_id -w user_password -e -l -a -f -wlm \
       -cor -createdb -printdbcfg -o db2look.sql
   ```

4. Generate the data file by using the EXPORT command on your self-managed Db2 system. In the following example, replace *directory* with the directory on your client machine where your data file exists. Replace *file_name* and *table_name* with the name of the data file and the name of the table.

   ```
   db2 "export to /directory/file_name.txt of del lobs to /directory/lobs/ \
       modified by coldel\| select * from table_name"
   ```

5. Connect to your RDS for Db2 database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *rds_database_alias*, *master_username,* and *master_password* with your own information.

```
db2 connect to rds_database_alias user master_username using master_password
```

6. Use the IMPORT command to import data from a file on the client machine into the remote RDS for Db2 database. For more information, see IMPORT command in the IBM Db2 documentation. In the following example, replace *directory* and *file_name* with the directory on your client machine where your data file exists and the name of the data file. Replace *SCHEMA_NAME* and *TABLE_NAME* with the name of your schema and table.

```
db2 "IMPORT from /directory/file_name.tbl OF DEL LOBS FROM /directory/lobs/ \
    modified by coldel\| replace into SCHEMA_NAME.TABLE_NAME"
```

7. Terminate your connection.

```
db2 terminate
```

Importing data from a client machine to Amazon RDS for Db2 with the LOAD command

You can use the LOAD CLIENT command to load data from a file on a client machine to the RDS for Db2 server. Because no SSH connectivity exists to the RDS for Db2 server, you can use the LOAD CLIENT command on either your self-managed Db2 server or your Db2 client machine.

> **Important**
>
> The LOAD CLIENT command method is useful for migrating small tables. If your network bandwidth between the client and RDS for Db2 is limited, we recommend that you use a different migration approach. For more information, see the Using native Db2 tools to migrate data from Db2 to Amazon RDS for Db2.
>
> If your data file includes references to large object file names, then the LOAD command won't work because large objects (LOBs) need to reside on the Db2 server. If you try to load LOBs from the client machine to the RDS for Db2 server, you will receive an SQL3025N error. Use the IMPORT command instead.

To load data to the RDS for Db2 server

1. Log in to your client machine with IBM Db2 Data Management Console. For more information, see Connecting to your Amazon RDS for Db2 DB instance with IBM Db2 Data Management Console.
2. Catalog the RDS for Db2 database on the client machine.

   a. Catalog the node. In the following example, replace *dns_ip_address* and *port* with the DNS name or the IP address and the port number of the self-managed Db2 database.

   ```
   db2 catalog tcpip node srcnode REMOTE dns_ip_address server port
   ```

   b. Catalog the database. In the following example, replace *source_database_name* and *source_database_alias* with the name of the self-managed Db2 database and the alias that you want to use for this database.

   ```
   db2 catalog database source_database_name as source_database_alias at node
    srcnode \
       authentication server_encrypt
   ```

3. Attach to the source database. In the following example, replace *source_database_alias*, *user_id*, and *user_password* with the alias you that created in the previous step and the user ID and password for the self-managed Db2 database.

   ```
   db2look -d source_database_alias -i user_id -w user_password -e -l -a -f -wlm \
       -cor -createdb -printdbcfg -o db2look.sql
   ```

4. Generate the data file by using the EXPORT command on your self-managed Db2 system. In the following example, replace *directory* with the directory on your client machine where your data file exists. Replace *file_name* and *TABLE_NAME* with the name of the data file and the name of the table.

   ```
   db2 "export to /directory/file_name.txt of del modified by coldel\| \
       select * from TPCH.TABLE_NAME"
   ```

5. Connect to your RDS for Db2 database using the master username and master password for your RDS for Db2 DB instance. In the following example, replace *rds_database_alias*, *master_username*, and *master_password* with your own information.

```
db2 connect to rds_database_alias user master_username using master_password
```

6. Use the LOAD command to load data from a file on the client machine to the remote RDS for Db2 database. For more information, see LOAD command in the IBM Db2 documentation. In the following example, replace *directory* with the directory on your client machine where your data file exists. Replace *file_name* and *TABLE_NAME* with the name of the data file and the name of the table.

```
db2 "LOAD CLIENT from /directory/file_name.txt \
    modified by coldel\| replace into TPCH.TABLE_NAME \
    nonrecoverable without prompting"
```

7. Terminate your connection.

```
db2 terminate
```

Importing data from Db2 to Amazon RDS for Db2 with the INSERT command

You can use the INSERT command from a self-managed Db2 server to insert your data into an Amazon RDS for Db2 database. With this migration approach, you use a nickname for the remote RDS for Db2 DB instance. Your self-managed Db2 database (source) must be able to connect to the RDS for Db2 database (target).

> **Important**
>
> The INSERT command method is useful for migrating small tables. If your network bandwidth between your self-managed Db2 database and RDS for Db2 database is limited, we recommend that you use a different migration approach. For more information, see Using native Db2 tools to migrate data from Db2 to Amazon RDS for Db2.

To copy data from a self-managed Db2 database to an RDS for Db2 database

1. Catalog the RDS for Db2 DB instance on the self-managed Db2 instance.

   a. Catalog the node. In the following example, replace *dns_ip_address* and *port* with the DNS name or the IP address and the port number of the self-managed Db2 database.

```
db2 catalog tcpip node remnode REMOTE dns_ip_address SERVER port
```

b. Catalog the database. In the following example, replace *rds_database_name* with the name of the database on your RDS for Db2 DB instance.

```
db2 catalog database rds_database_name as remdb at node remnode \
    authentication server_encrypt
```

2. Enable federation on the self-managed Db2 instance. In the following example, replace *source_database_name* with the name of your database on the self-managed Db2 instance.

```
db2 update dbm cfg using FEDERATED YES source_database_name
```

3. Create tables on the RDS for Db2 DB instance.

a. Catalog the node. In the following example, replace *dns_ip_address* and *port* with the DNS name or the IP address and the port number of the self-managed Db2 database.

```
db2 catalog tcpip node srcnode REMOTE dns_ip_address server port
```

b. Catalog the database. In the following example, replace *source_database_name* and *source_database_alias* with the name of the self-managed Db2 database and the alias that you want to use for this database.

```
db2 catalog database source_database_name as source_database_alias at node
 srcnode \
    authentication server_encrypt
```

4. Attach to the source database. In the following example, replace *source_database_alias*, *user_id*, and *user_password* with the alias that you created in the previous step and the user ID and password for the self-managed Db2 database.

```
db2look -d source_database_alias -i user_id -w user_password -e -l -a -f -wlm \
    -cor -createdb -printdbcfg -o db2look.sql
```

5. Set up federation, and create a nickname for the RDS for Db2 database table on the self-managed Db2 instance.

a. Connect to your local database. In the following example, replace *source_database_name* with the name of the database on your self-managed Db2 instance.

```
db2 connect to source_database_name
```

b. Create a wrapper to access Db2 data sources.

```
db2 create wrapper drda
```

c. Define a data source on a federated database. In the following example, replace *admin* and *admin_password* with your credentials for your self-managed Db2 instance. Replace *rds_database_name* with the name of the database on your RDS for Db2 DB instance.

```
db2 "create server rdsdb2 type DB2/LUW version '11.5.9.0' \
    wrapper drda authorization "admin" password "admin_password" \
    options( dbname 'rds_database_name', node 'remnode')"
```

d. Map the users on the two databases. In the following example, replace *master_username* and *master_password* with your credentials for your RDS for Db2 DB instance.

```
db2 "create user mapping for user server rdsdb2 \
    options (REMOTE_AUTHID 'master_username', REMOTE_PASSWORD
 'master_password')"
```

e. Verify the connection to the RDS for Db2 server.

```
db2 set passthru rdsdb2
```

f. Create a nickname for the table in the remote RDS for Db2 database. In the following example, replace *NICKNAME* and *TABLE_NAME* with a nickname for the table and the name of the table.

```
db2 create nickname REMOTE.NICKNAME for RDSDB2.TABLE_NAME.NICKNAME
```

6. Insert data into the table in the remote RDS for Db2 database. Use the nickname in a `select` statement on the local table in the self-managed Db2 instance. In the following example, replace *NICKNAME* and *TABLE_NAME* with a nickname for the table and the name of the table.

```
db2 "INSERT into REMOTE.NICKNAME select * from RDS2DB2.TABLE_NAME.NICKNAME"
```

Importing data from Db2 to Amazon RDS for Db2 with the INGEST utility

You can use the INGEST utility to continually stream data from files and pipes on a client machine to a target Amazon RDS for Db2 DB instance. The INGEST utility supports INSERT and MERGE operations. For more information, see Ingest utility in the IBM Db2 documentation.

Because the INGEST utility supports nicknames, you can use the utility to transfer data from your self-managed Db2 database to an RDS for Db2 database. This approach works as long as network connectivity exists between the two databases.

> **Important**
>
> The INGEST utility doesn't support large objects (LOBs). Use the IMPORT command instead.

To use the RESTARTABLE feature of the INGEST utility, run the following command on the RDS for Db2 database.

```
db2 "call sysproc.sysinstallobjects('INGEST','C',NULL,NULL)"
```

Amazon RDS for Db2 federation

You can use your Amazon RDS for Db2 database as a federated database. After setting up federation for RDS for Db2, you will be able to access and query data across multiple databases from your RDS for Db2 database. Federation saves you from needing to migrate data to your RDS for Db2 database or consolidate data into a single database.

By using your RDS for Db2 database as a federated database, you can continue to access to all RDS for Db2 features and can take advantage of various AWS services, all while keeping your data in different databases. You can set up both homogeneous federation which connects different databases of the same type, or heterogeneous federation which connects different databases of different types.

You first connect your Db2 database in RDS for Db2 to remote databases. Then you can run queries against all your connected databases. For example, you can run a SQL JOIN statement that joins tables in your RDS for Db2 database with tables in a remote Db2 on z/OS database.

Topics

- Homogeneous federation
- Heterogeneous federation

Homogeneous federation

You can set up homogeneous federation between your RDS for Db2 database and the following Db2 family of products:

- Db2 for Linux, UNIX, Windows (LUW)
- Db2 iSeries
- Db2 for z/OS

RDS for Db2 homogeneous federation doesn't support the following actions:

- Running `CATALOG` commands to set up a node directory and a remote database on an RDS for Db2 host database
- Settting up Workload Balancing (WLB) when federating to Db2 on z/OS
- Configuring the IBM data server driver configuration file (`db2dsdriver.cfg`)

RDS for Db2 homogeneous federation has the following requirements:

- You must create the DRDA wrapper in UNFENCED mode. If you don't, then federation won't work in RDS for Db2.
- You must allow incoming and outgoing traffic from your RDS for Db2 host database to your remote host databases. For more information, see Provide access to your DB instance in your VPC by creating a security group.

Topics

- Step 1: Create a DRDA wrapper and a federated server
- Step 2: Create a user mapping
- Step 3: Check the connection

Step 1: Create a DRDA wrapper and a federated server

For homogeneous federation, create a DRDA wrapper and a federated server. The connection to the remote host uses HOST, PORT, and DBNAME.

Choose one of the following methods based on the type of your remote Db2 database:

- **Db2 for Linux, UNIX, and Windows (LUX) database** – Run the following SQL commands. In the following example, replace *server_name* with the name of the server that you will use for federation. Replace *db2_version* with the version of your remote Db2 database. Replace *username* and *password* with your credentials for the remote Db2 database you want to connect to. Replace *db_name*, *dns_name*, and *port* with the appropriate values for the remote Db2 database you want to connect to.

```
create wrapper drda options(DB2_FENCED 'N');
create server server_name type DB2/LUW wrapper drda version 'db2_version'
 authorization "master_username" password "master_password" options (add DBNAME
 'db_name',add HOST 'dns_name',add PORT 'port');
```

Example

```
create wrapper drda options(DB2_FENCED 'N');
```

```
create server SERVER1 type DB2/LUW wrapper drda version '11.5' authorization
 "sysuser" password "******" options (add DBNAME 'TESTDB2',add HOST
 'ip-123-45-67-899.us-west-1.compute.internal',add PORT '25010');
```

- **Db2 iSeries** – Run the following SQL commands. In the following example, replace *wrapper_name* and *library_name* with a name for your DRDA wrapper and the wrapper library file. Replace *server_name* with the name of the server that you will use for federation. Replace *db2_version* with the version of your remote Db2 database. Replace *username* and *password* with your credentials for the remote Db2 database you want to connect to. Replace *dns_name*, *port*, and *db_name* with the appropriate values for the remote Db2 database you want to connect to.

```
create wrapper wrapper_name library 'library name' options(DB2_FENCED 'N');
create server server_name type db2/mvs version db2_version wrapper wrapper_name
 authorization "sername" password "password" options (HOST 'dns_name', PORT 'port',
 DBNAME 'db_name');
```

Example

```
create wrapper WRAPPER1 library 'libdb2drda.so' options(DB2_FENCED 'N');
create server SERVER1 type db2/mvs version 11 wrapper WRAPPER1 authorization
 "sysuser" password "******" options (HOST 'test1.123.com', PORT '446', DBNAME
 'STLEC1');
```

- **Db2 for z/OS** – Run the following SQL commands. In the following example, replace *wrapper_name* and *library_name* with a name for your DRDA wrapper and the wrapper library file. Replace *server_name* with the name of the server that you will use for federation. Replace *db2_version* with the version of your remote Db2 database. Replace *username* and *password* with your credentials for the remote Db2 database you want to connect to. Replace *dns_name*, *port*, and *db_name* with the appropriate values for the remote Db2 database you want to connect to.

```
create wrapper wrapper_name library 'library_name' options(DB2_FENCED 'N');
create server server_name type db2/mvs version db2_version wrapper wrapper_name
 authorization "username" password "password" options (HOST 'dns_name', PORT 'port',
 DBNAME 'db_name');
```

Example

```
create wrapper WRAPPER1 library 'libdb2drda.so' OPTIONS(DB2_FENCED 'N');
create server SERVER1 type db2/mvs version 11 wrapper WRAPPER1 authorization
 "sysuser" password "******" options (HOST 'test1.123.com', PORT '446', DBNAME
 'STLEC1');
```

Step 2: Create a user mapping

Create a user mapping to associate your federated server with your data source server by running the following SQL command. In the following example, replace *server_name* with the name of the remote server than you want to perform operations on. This is the server that you created in step 1. Replace *username* and *password* with your credentials for this remote server.

```
create user mapping for user server server_name options (REMOTE_AUTHID 'username',
 REMOTE_PASSWORD 'password');
```

For more information, see User mappings in the IBM Db2 documentation.

Step 3: Check the connection

Confirm that setting up your federation was successful by checking the connection. Open a session to send native SQL commands to your remote data source using the SET PASSTHRU command, and then create a table on the remote data server.

1. Open and close a session to submit SQL to a data source. In the following example, replace *server_name* with the name of the server that you created for federation in step 1.

   ```
   set passthru server_name;
   ```

2. Create a new table. In the following example, replace *column_name*, *data_type*, and *value* with the appropriate items for your table.

   ```
   create table table_name
    ( column_name data_type(value), column_name data_type(value);
   ```

   For more information, see CREATE TABLE statement in the IBM Db2 documentation.

3. Create an index, insert values for rows into the table, and reset the connection. Resetting the connection drops the connection but retains the back-end processes. In the following

example, replace *index_name*, *table_name*, *column_name*, and *columnx_value* with your information.

```
create index index_name on table_name(column_name);
insert into table_name values(column1_value,column2_value,column3_value);
insert into table_name values(column1_value,column2_value,column3_value);
set passthru reset;

connect reset;
```

4. Connect to your remote Db2 database, create a nickname for your remote server, and perform operations. When you are done accessing data in the remote Db2 database, reset and then terminate the connection. In the following example, replace *database_name* with the name of your remote Db2 database. Replace *nickname* with a name. Replace *server_name* and *table_name* with the name of the remote server and table on that server that you want to perform operations on. Replace *username* with the information for your remote server. Replace *sql_command* with the operation to perform on the remote server.

```
connect to database_name;
create nickname nickname for server_name."username"."table_name";
select sql_command from nickname;
connect reset;
terminate;
```

Example

The following example creates a pass-through session to allow operations on the federated server `testdb10`.

Next, it creates the table `t1` with three columns with different data types.

Then, the example creates the index `i1_t1` on three columns in table `t1`. Afterwards, it inserts two rows with values for these three columns, and then disconnects.

Last, the example connects to the remote Db2 database `testdb2` and creates a nickname for the table `t1` in the federated server `testdb10`. It creates the nickname with the username TESTUSER for that data source. An SQL command outputs all data from the table `t1`. The example disconnects and ends the session.

```
set passthru testdbl0;
```

```
create table t1 ( c1 decimal(13,0), c2 char(200), c3 int);

create index i1_t1 on t1(c3);
insert into t1 values(1,'Test',1);
insert into t1 values(2,'Test 2',2);
connect reset;

connect to testdb2;
create nickname remote_t1 for testdb10."TESTUSER"."T1";
select * from remote_t1;
connect reset;
terminate;
```

Heterogeneous federation

You can set up heterogeneous federation between your RDS for Db2 database and other data sources such as Oracle and Microsoft SQL Server. For a complete list of data sources that Db2 LUW supports, see Data Source Support Matrix of Federation Bundled in Db2 LUW V11.5 on the IBM Support site.

RDS for Db2 heterogeneous federation doesn't support the following items:

- Native wrappers for the other data sources
- JDBC wrappers for the other data sources
- Federation to Sybase, Informix, and Teradata data sources because these data sources require client software installation on RDS for Db2

RDS for Db2 heterogeneous federation has the following requirements:

- RDS for Db2 only supports the ODBC wrapper method.
- If you create an explicit definition of a wrapper, then you must set the option `DB2_FENCED` to `'N'`. For a list of valid wrapper options for ODBC, see ODBC options in the IBM Db2 documentation.
- You must allow incoming and outgoing traffic from your RDS for Db2 host database to your remote host database. For more information, see Provide access to your DB instance in your VPC by creating a security group.

For information about federation to Oracle, see How to query Oracle by using Db2 Federation and the ODBC driver? on the IBM Support site.

For more information about data sources that support federation, see Data Source Support Matrix of Federation Bundled in Db2 LUW V11.5 on the IBM Support site.

Topics

- Step 1: Create an ODBC wrapper
- Step 2: Create a federated server
- Step 3: Create a user mapping
- Step 4: Check the connection

Step 1: Create an ODBC wrapper

Create a wrapper by running the following command:

```
db2 "create wrapper odbc options( module '/home/rdsdb/sqllib/federation/odbc/lib/
libodbc.so')"
```

Step 2: Create a federated server

Create a federated server by running the following command. In the following example, replace *server_name* with the name of the server that you will use for federation. Replace *wrapper_type* with the appropriate wrapper. Replace *db_version* with the version of your remote database. Replace *dns_name*, *port*, and *service_name* with the appropriate values for the remote database that you want to connect to.

```
db2 "create server server_name type wrapper_type version db_version options (HOST
 'dns_name', PORT 'port', SERVICE_NAME 'service_name')“
```

For information about wrapper types, see Data Source Support Matrix of Federation Bundled in Db2 LUW V11.5 on the IBM Support site.

Example

The following example creates a federated server for a remote Oracle database.

```
db2 "create server server1 type oracle_odbc version 12.1 options (HOST
 'test1.amazon.com', PORT '1521', SERVICE_NAME 'pdborcl.amazon.com')“
```

Step 3: Create a user mapping

Create a user mapping to associate your federated server with your data source server by running the following SQL command. In the following example, replace *server_name* with the name of the remote server than you want to perform operations on. This is the server that you created in step 2. Replace *username* and *password* with your credentials for this remote server.

```
create user mapping for user server server_name options (REMOTE_AUTHID 'username',
 REMOTE_PASSWORD 'password');
```

For more information, see User mappings in the IBM Db2 documentation.

Step 4: Check the connection

Confirm that setting up your federation was successful by checking the connection. Open a session to send native SQL commands to your remote data source using the SET PASSTHRU command, and then create a table on the remote data server.

1. Open and close a session to submit SQL to a data source. In the following example, replace *server_name* with the name of the server that you created for federation in step 2.

   ```
   set passthru server_name;
   ```

2. Create a new table. In the following example, replace *column_name*, *data_type*, and *value* with the appropriate items for your table.

   ```
   create table table_name
    ( column_name data_type(value), column_name data_type(value);
   ```

   For more information, see CREATE TABLE statement in the IBM Db2 documentation.

3. Create an index, insert values for rows into the table, and reset the connection. Resetting the connection drops the connection but retains the back-end processes. In the following example, replace *index_name*, *table_name*, *column_name*, and *columnx_value* with your information.

   ```
   create index index_name on table_name(column_name);
   ```

```
insert into table_name values(column1_value,column2_value,column3_value);
insert into table_name values(column1_value,column2_value,column3_value);
set passthru reset;

connect reset;
```

4. Connect to your remote Db2 database, create a nickname for your remote server, and perform operations. When you are done accessing data in the remote Db2 database, reset and then terminate the connection. In the following example, replace *database_name* with the name of your remote Db2 database. Replace *nickname* with a name. Replace *server_name* and *table_name* with the name of the remote server and table on that server that you want to perform operations on. Replace *username* with the information for your remote server. Replace *sql_command* with the operation to perform on the remote server.

```
connect to database_name;
create nickname nickname for server_name."username"."table_name";
select sql_command from nickname;
connect reset;
terminate;
```

Example

The following example creates a pass-through session to allow operations on the federated server `testdb10`.

Next, it creates the table `t1` with three columns with different data types.

Then, the example creates the index `i1_t1` on three columns in table `t1`. Afterwards, it inserts two rows with values for these three columns, and then disconnects.

Last, the example connects to the remote Db2 database `testdb2` and creates a nickname for the table `t1` in the federated server `testdb10`. It creates the nickname with the username TESTUSER for that data source. An SQL command outputs all data from the table `t1`. The example disconnects and ends the session.

```
set passthru testdbl0;

create table t1 ( c1 decimal(13,0), c2 char(200), c3 int);

create index i1_t1 on t1(c3);
```

```
insert into t1 values(1,'Test',1);
insert into t1 values(2,'Test 2',2);
connect reset;

connect to testdb2;
create nickname remote_t1 for testdbl0."TESTUSER"."T1";
select * from remote_t1;
connect reset;
terminate;
```

Working with replicas for Amazon RDS for Db2

RDS for Db2 supports creating replica databases to provide read scaling and disaster recovery capabilities. You can create replicas in two modes: read-only replicas for offloading read workloads, and standby replicas for cross-region disaster recovery. RDS for Db2 uses IBM Db2 High Availability Disaster Recovery (HADR) technology for replication. For more information, see High availability disaster recovery (HADR) in the IBM Db2 documentation.

A *Db2 replica* database is a physical copy of your primary database. A Db2 replica in read-only mode is called a *read replica*. A Db2 replica in standby mode is called a *standby replica*. Db2 doesn't permit writes in a replica, but you can promote a replica to make it writable. The promoted replica has the replicated data to the point when the request was made to promote it. For more information, see Promoting a read replica to be a standalone DB instance.

For a summary of the features and behaviors of RDS for Db2 replicas, see Differences between read replicas for DB engines.

Read-only and standby replicas

When creating or modifying a Db2 replica, you can place it in either of the following modes:

Read-only

This is the default. HADR transmits and applies changes from the source database to all read replica databases. For read-only replicas, the Db2 environment variable `DB2_HADR_ROS` is set to `ON`. The isolation level for read queries on the replica database is `Uncommitted Read`. For more information, see Isolation level on the active standby database in the IBM Db2 documentation.

For general information about read replicas that applies to all DB engines, see Working with DB instance read replicas. For more information about Db2 HADR, see High availability disaster recovery (HADR) in the IBM Db2 documentation.

Standby

For standby replicas, the Db2 environment variable `DB2_HADR_ROS` is set to `OFF` so that the replica databases don't accept user connections. The primary use for standby replicas is cross-Region disaster recovery.

A standby replica can't serve a read-only workload. The standby replica doesn't have any archive logs.

You can create up to three replicas from one source DB instance. You can create a combination of read-only and standby DB replicas for the same source DB instance. After you create a replica, you can change the replica mode. for more information, see Modifying the RDS for Db2 replica mode.

Before creating replicas, make sure that you meet all requirements. For more information, see Requirements and considerations for RDS for Db2 replicas.

Database activations

Db2 HADR is configured at the database level. After you create replicas, HADR is set for all Db2 databases, including `rdsadmin`, which RDS fully manages. Before you create Db2 replicas, you must explicitly activate all databases. Otherwise, creation of replicas fails and Amazon RDS emits an event. After a DB instance has one or more replicas, you can't activate or deactivate any databases on the DB instance by using the `rdsadmin.activate_database` or `rdsadmin.deactivate_database` stored procedures. For more information, see Stored procedures for databases for RDS for Db2.

HADR configurations

You can see all HADR configurations for a database by connecting to the database and then running `db2 get db cfg`.

Archive log retention

Amazon RDS purges logs from a primary DB instance after the following conditions have been met:

- The logs are at least two hours old.
- The setting for archive log retention hours has passed.
- The archive logs were successfully replicated to all replica DB instances. This condition applies both to DB instances in the same AWS Region and to cross-Region DB instances.

For information about setting archive log retention hours, see rdsadmin.set_archive_log_retention.

Amazon RDS checks and cleans up each database individually. If a database loses the HADR connection or if information about the connection isn't available, then Amazon RDS skips the database and doesn't purge the archive logs.

Outages during Db2 replication

When you create a replica, Amazon RDS takes a DB snapshot of your source DB instance and begins replication. When the DB snapshot operation begins, the source DB instance experiences a very brief I/O suspension. The I/O suspension typically lasts about one second. However, if the source DB instance is a Multi-AZ deployment, then the source DB instance doesn't experience any I/O suspension. This is because with Multi-AZ deployments, the snapshot is taken from the secondary DB instance.

The DB snapshot becomes the Db2 replica. Amazon RDS sets the necessary parameters and permissions for the source database and replica without any service interruption. Similarly, if you delete a replica, no outage occurs.

Requirements and considerations for RDS for Db2 replicas

Db2 replica requirements fall into several categories: licensing and versioning, backup and restore considerations, replication behavior, and general operational considerations. Before creating a Db2 replica, familiarize yourself with the following requirements and considerations.

Version and licensing requirements for RDS for Db2 replicas

Before you create an RDS for Db2 replica, review the following information about versions and licensing models:

- **Supported versions** – All Db2 11.5 versions support replica DB instances.

  Source and replica DB instances must use the same major version. Db2 replicas support minor version upgrades but not major version upgrades. For information about upgrading DB instances, see Upgrading a DB instance engine version.

  > **Note**
  >
  > When upgrading a source DB instance, all replicas are automatically upgraded to maintain version compatibility.

- **Valid licensing models and replica modes** – Both Db2 Advanced Edition (AE) and Standard Edition (SE) can create replicas in read-only or standby mode for both the Bring Your Own License (BYOL) model and the Db2 license through AWS Marketplace model.
- **Custom parameter group** – You must specify a custom parameter group for the replica.

For replicas that use the BYOL model, this custom parameter group must include your IBM Site ID and IBM Customer ID. For more information, see IBM IDs for bring your own license (BYOL) for Db2. You can specify this custom parameter group for the replica by using the AWS Management Console, the AWS CLI , or the RDS API.

- **vCPU count** varies by replica mode and licensing model:
  - **Standby replicas** always use two vCPUs regardless of DB instance size.
    - **BYOL model** – AWS License Manager configurations show that RDS for Db2 DB instances use two vCPUs.
    - **Db2 license through AWS Marketplace model** – Bills reflect license costs for two vCPUs.
  - **Read-only replicas** use the same vCPU count as the DB instance size.
    - **BYOL model** – AWS License Manager configurations show that RDS for Db2 DB instances use the same number of vCPUs that match the DB instance size.
    - **Db2 license through AWS Marketplace model** – Bills reflect license costs for the same number of vCPUs that match the DB instance size.

Backup and restore considerations for RDS for Db2 replicas

Replica backups have different behavior than primary database backups. Consider the following backup and restore requirements:

- To create snapshots of RDS for Db2 replicas or turn on automatic backups, make sure to set the backup retention period manually. Automatic backups aren't turned on by default.
- When you restore a replica backup, you restore to the database time, not the time that the backup was taken. The *database time* refers to the latest applied transaction time of the data in the backup. The difference is significant because a replica can lag behind the primary database for minutes or hours. When there are multiple databases, RDS for Db2 uses the earliest database time.

  To find the difference, run the AWS CLI describe-db-snapshots command or call the RDS API DescribeDBSnapshots operation. Compare the `SnapshotDatabaseTime` value to the `OriginalSnapshotCreateTime` value. The `SnapshotDatabaseTime` value is the database time of the replica backup. The `OriginalSnapshotCreateTime` value is the latest applied transaction on the primary database.

For more information about backups and restoring backups, see Working with RDS for Db2 replica backups.

Replication considerations for RDS for Db2 replicas

Db2 replicas use HADR technology with specific limitations and behaviors. Review the following replication considerations:

- Replication uses Db2 HADR for all databases on the RDS for Db2 DB instance.
- Replication doesn't support the LOAD command. If you run the LOAD command from the source DB instance, you will receive inconsistent data.
- RDS for Db2 doesn't replicate the following items:
  - Storage access. Be aware of data, such as external tables, that rely on storage access.
  - Non-inline LOBs that are not logged.
  - Binaries of external stored procedures (in C or Java).
- For standby replicas, RDS for Db2 replicates the following items:
  - Local users, except master users
  - Database configuration parameters
- For read-only replicas, RDS for Db2 replicates the following items:
  - Local users, except master users
  - SID group mappings

Miscellaneous considerations for RDS for Db2 replicas

Additional operational considerations apply to Db2 replicas. Review the following items:

- RDS for Db2 replicates database configurations to the replicas. When RDS for Db2 promotes a replica, it deactivates and activates each database.
- RDS for Db2 replicates the local users, but not the master user, and SID group mappings to the replicas. You can modify the master user on the replica. For more information, see Modifying an Amazon RDS DB instance.
- All databases must be in an active state. For information about activating databases, see Stored procedures for databases for RDS for Db2.

- All stored procedures for creating, dropping, restoring, or rolling forward databases must be completed before creating a replica. For information about these stored procedures, see Stored procedures for databases for RDS for Db2.
- When the replica is created, Amazon RDS sets the database-level parameter `blocknonlogged` for all databases on the source DB instance to `YES`. When the source replica becomes a standalone instance again, Amazon RDS sets the value back to `NO`. For more information, see blocknonlogged - Block creation of tables that allow non-logged activity configuration parameter in the IBM Db2 documentation.
- When the replica is created, Amazon RDS sets the database-level parameter `logindexbuild` for all databases on the source DB instance to `YES`. When the source replica becomes a standalone instance again, Amazon RDS sets the value back to `NO`. For more information, see logindexbuild - Log index pages created configuration parameter in the IBM Db2 documentation.

Preparing to create an RDS for Db2 replica

Before creating an RDS for Db2 replica, you must complete the following tasks for successful replication. These tasks help prevent common issues and ensure optimal performance.

Task 1: Enable automatic backups

Before a DB instance can serve as a source DB instance, you must enable automatic backups on the source DB instance. This is a prerequisite for all replica creation operations. To learn how to perform this procedure, see Enabling automated backups.

For information about backups specific to Db2 replicas, see Working with RDS for Db2 replica backups.

Task 2: Plan compute and storage resources

Ensure that the source DB instance and its replicas are sized properly, in terms of compute and storage, to suit their operational load. If a replica reaches compute, network, or storage resource capacity, the replica stops receiving or applying changes from its source. For information about monitoring replica performance and resource utilization, see Monitoring read replication.

RDS for Db2 doesn't intervene to mitigate high replica lag between a source DB instance and its replicas. If you experience high replica lag, see Monitoring Db2 replication lag for troubleshooting guidance.

You can modify the storage and CPU resources of a replica independently from its source and other replicas. For more information, see Modifying an Amazon RDS DB instance.

Task 3: Prepare databases

Before creating a replica, confirm that your databases are ready based on the following points:

- The DB instance contains all databases that you want present on the DB instance. After replica creation, you can't create, drop, or native restore a database on the DB instance. Any calls to the rdsadmin.create_database, rdsadmin.drop_database, or rdsadmin.restore_database stored procedures fail.
- All databases on the DB instance are in an active state. If any database is in an inactive state, replica creation will fail. For information about activating databases, see Stored procedures for databases for RDS for Db2.

Next steps

After completing all the preparation tasks, you are ready to create a Db2 replica.

- To create a read-only replica, see Creating a read replica.
- To create a standby replica, see Creating a standby Db2 replica.

Creating an RDS for Db2 replica in standby mode

By default, Db2 replicas are created in read-only mode. You can create a replica in standby mode for disaster recovery purposes. Standby replicas don't accept user connections but provide faster failover capabilities for cross-Region scenarios.

Before creating a standby replica, make sure that you have completed the preparation tasks. For more information, see Preparing to create an RDS for Db2 replica. After creating a standby replica, you can change the replica mode. For more information, see Modifying the RDS for Db2 replica mode.

You can create a standby replica using the AWS Management Console, the AWS CLI, or the RDS API. For information about creating a read-only replica, see Creating a read replica.

Console

To create a standby replica from a source RDS for Db2 DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the RDS for Db2 DB instance that you want to use as the source for a standby replica.
4. For **Actions**, choose **Create replica**.
5. For **Replica mode**, choose **Standby**.
6. Choose the settings that you want to use. For **DB instance identifier**, enter a name for the standby replica. Adjust other settings as needed.
7. For **Regions**, choose the AWS Region where the standby replica will be launched.
8. Choose your instance size and storage type. We recommend that you use the same DB instance class and storage type as the source DB instance for the standby replica.
9. For **Multi-AZ deployment**, choose **Create a standby instance** to create a standby of your replica in another Availability Zone for failover support for the standby replica.
10. Choose the other settings that you want to use.
11. Choose **Create replica**.

In the **Databases** page, the standby replica has the role **Replica**.

AWS CLI

To create a Db2 replica in standby mode, use the AWS CLI command create-db-instance-read-replica with `--replica-mode` set to `mounted`.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance-read-replica \
    --db-instance-identifier my_standby_replica \
    --source-db-instance-identifier my_db_instance \
    --replica-mode mounted
```

For Windows:

```
aws rds create-db-instance-read-replica ^
    --db-instance-identifier my_standby_replica ^
    --source-db-instance-identifier my_db_instance ^
    --replica-mode mounted
```

RDS API

To create a Db2 replica in standby mode, specify `ReplicaMode=mounted` in the RDS API operation CreateDBInstanceReadReplica.

Modifying the RDS for Db2 replica mode

You can change the replica mode of an existing Db2 replica between read-only and standby modes. This flexibility allows you to adapt your replica configuration based on changing requirements for read workloads or disaster recovery needs.

You might want to change replica modes in the following scenarios:

- **Read-only to standby** – When you no longer need read capacity but want to maintain disaster recovery capabilities
- **Standby to read-only** – When you need to add read capacity for reporting or analytics workloads

Before changing replica modes, make sure the following conditions are met:

- The replica is in an available state.
- No active maintenance operations are running on the replica.
- You have the necessary permissions to modify the DB instance.

The change operation can take a few minutes. During the operation, the DB instance status changes to **modifying**. For more information about status changes, see Viewing Amazon RDSDB instance status. When you change from read-only to standby mode, the replica disconnects all active connections.

> **Important**
>
> Because changing replica modes temporarily interrupts service, plan the change during a maintenance window to minimize impact on your applications.

You can modify the replica mode using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To change the replica mode of a Db2 replica

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the replica database that you want to modify.
4. Choose **Modify**.
5. For **Replica mode**, choose the desired mode:
   - **Read-only** – For read workloads
   - **Standby** – For disaster recovery
6. Choose the other settings that you want to change.
7. Choose **Continue**.
8. For **Scheduling of modifications**, choose **Apply immediately**.
9. Choose **Modify DB instance**.
10. After the modification completes, verify the replica mode change in the **Databases** page. The replica status should show as **Available** when the change is complete.

AWS CLI

To change a Db2 replica from read-only mode to standby mode, set `--replica-mode` to `mounted` in the AWS CLI command modify-db-instance. To change a Db2 replica from standby mode to read-only mode, set `--replica-mode` to `open-read-only`.

The following example changes a replica from read-only to standby mode:

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my_db2_replica \
    --replica-mode mounted
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my_db2_replica ^
    --replica-mode mounted
```

The following example changes a replica from standby to read-only mode:

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my_db2_replica \
    --replica-mode open-read-only
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my_db2_replica ^
    --replica-mode open-read-only
```

RDS API

To change a Db2 replica from read-only mode to standby mode, set `ReplicaMode=mounted` in ModifyDBInstance. To change a Db2 replica from standby mode to read-only mode, set `ReplicaMode=open-read-only`.

The following is an example API call to change the replica mode from read-only to standby:

```
{
    "DBInstanceIdentifier": "my_db2_replica",
    "ReplicaMode": "mounted",
```

```
    "ApplyImmediately": true
}
```

The following is an example API call to change the replica mode from standby to read-only:

```
{
    "DBInstanceIdentifier": "my_db2_replica",
    "ReplicaMode": "open-read-only",
    "ApplyImmediately": true
}
```

For information about the differences between replica modes, see Working with replicas for Amazon RDS for Db2. For troubleshooting replica issues, see Troubleshooting RDS for Db2 replication issues.

Working with RDS for Db2 replica backups

You can create and restore backups of an RDS for Db2 replica just like a primary database. However, there are important differences in how replica backups work, particularly regarding restore timing and backup retention settings.

RDS for Db2 supports both automatic backups and manual snapshots for replicas. RDS for Db2 doesn't support point-in-time restore. For information about RDS backups, see Backing up, restoring, and exporting data.

Key differences for replica backups

Replica backups differ from primary database backups in several important ways:

- Automatic backups aren't enabled by default for replicas.
- Restore operations use database time rather than backup creation time.
- Replica lag can affect the actual data restored. For information about monitoring replica lag, see Monitoring Db2 replication lag.

Enabling automatic backups for RDS for Db2 replicas

Unlike primary databases, RDS for Db2 replicas don't have automated backups enabled by default. You must manually configure the backup retention period to enable automatic backups. Enable automated backups by setting the backup retention period to a positive nonzero value.

Console

To enable automatic backups immediately

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**.
4. For **Backup retention period**, choose a positive nonzero value, for example three days.
5. Choose **Continue**.
6. Choose **Apply immediately**.
7. Choose **Modify DB instance** to save your changes and enable automated backups.

AWS CLI

To enable automated backups, use the AWS CLI modify-db-instance command.

Include the following parameters:

- `--db-instance-identifier`
- `--backup-retention-period`
- `--apply-immediately` or `--no-apply-immediately`

The following example enables automated backups by setting the backup retention period to three days. The changes are applied immediately.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my_db_instance  \
    --backup-retention-period 3 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my_db_instance  ^
    --backup-retention-period 3 ^
    --apply-immediately
```

RDS API

To enable automated backups, use the RDS API ModifyDBInstance operation with the following required parameters:

- `DBInstanceIdentifier`
- `BackupRetentionPeriod`

Restoring an RDS for Db2 replica backup

You can restore an RDS for Db2 replica backup the same way that you can restore a backup of the primary database. For more information, see Restoring to a DB instance.

The most important consideration when restoring replica backups is understanding the difference between database time and backup creation time, especially when replica lag is present.

You can monitor replication lag and ensure that your backups contain the expected data. For information about the ReplicaLag metric, see Amazon CloudWatch metrics for Amazon RDS.

Understanding timing differences

When you restore a replica backup, you must determine the point in time to which you are restoring. The *database time* refers to the latest applied transaction time of the data in the backup. When you restore a replica backup, you restore to the database time, not the time when the backup completed. The difference is significant because a replica can lag behind the primary database by minutes or hours. Thus, the database time of a replica backup might be much earlier than the snapshot creation time.

To find the difference between database time and creation time, run the AWS CLI describe-db-snapshots command or call the RDS API DescribeDBSnapshots operation. Compare the `SnapshotDatabaseTime` value and the `OriginalSnapshotCreateTime` value. The `SnapshotDatabaseTime` value is the earliest database time among all the databases of the replica backup. The `OriginalSnapshotCreateTime` value is the latest applied transaction on

the primary database. Note that replication lags could be different for multiple databases, and the database time could be in between these two times.

The following AWS CLI example shows the difference between the two times:

For Linux, macOS, or Unix:

```
aws rds describe-db-snapshots \
    --db-instance-identifier my_db2_replica \
    --db-snapshot-identifier my_replica_snapshot
```

For Windows:

```
aws rds describe-db-snapshots ^
    --db-instance-identifier my_db2_replica ^
    --db-snapshot-identifier my_replica_snapshot
```

This command produces output similar to the following example.

```
{
    "DBSnapshots": [
        {
            "DBSnapshotIdentifier": "my_replica_snapshot",
            "DBInstanceIdentifier": "my_db2_replica",
            "SnapshotDatabaseTime": "2022-07-26T17:49:44Z",
            ...
            "OriginalSnapshotCreateTime": "2021-07-26T19:49:44Z"
        }
    ]
}
```

Troubleshooting RDS for Db2 replication issues

This topic describes common RDS for Db2 replication issues and provides troubleshooting guidance for both read-only and standby replicas. In addition to reviewing the following troubleshooting information, make sure that you followed the requirements and considerations, and completed the preparation steps before creating Db2 replicas.

Replica creation failures

Replica creation can fail for several reasons:

- **Inactive databases** – All databases on the source DB instance must be active before creating replicas.

  For information about activating databases, see Stored procedures for databases for RDS for Db2.
- **Missing automatic backups** – The source DB instance must have automatic backups enabled.

  For information about enabling backups, see Enabling automatic backups for RDS for Db2 replicas.
- **Parameter group issues** – Custom parameter groups are required for replicas. For BYOL licensing, the parameter group must include the IBM Site ID and IBM Customer ID.

  For more information, see IBM IDs for bring your own license (BYOL) for Db2.

Monitoring Db2 replication lag

To monitor replication lag in Amazon CloudWatch, view the Amazon RDS `ReplicaLag` metric. For more information about replication lag time, see Monitoring read replication and Amazon CloudWatch metrics for Amazon RDS. For information about setting up CloudWatch alarms for replica lag, see Monitoring Amazon RDS metrics with Amazon CloudWatch.

For a read-only replica, if the lag time is too long, query the `MON_GET_HADR` table for the status of the replica DB instance.

For a standby replica, if the lag time is too long, query the `MON_GET_HADR` table for the status of the source DB instance. Don't query the replica DB instance because replica DB instances don't accept user connections.

Common causes of high replication lag include the following reasons:

- Insufficient compute resources on the replica
- Network connectivity issues between the source and the replica
- High write activity on the source database
- Storage performance limitations on the replica

If high replication lag persists, consider scaling your replica resources. For more information, see Modifying an Amazon RDS DB instance.

Db2 replication errors

Db2 replication can be in an error state for a number of reasons. Perform the following actions:

- Monitor events and the DB instance state to make sure that the DB instance is replicating.

  For more information, see Working with Amazon RDS event notification.
- Check the diagnostic logs for the Db2 replica in the Amazon RDS console. In the logs, look for errors in HADR messages. Compare the log sequence number to the primary sequence number.

  For information about accessing and interpreting Db2 diagnostic logs, see Amazon RDS for Db2 database log files. For information about Db2 HADR configuration and troubleshooting, see Working with replicas for Amazon RDS for Db2.

If replication errors persist, you might need to recreate the replica.

Connection issues

If you can't connect to your replica, review the following information about the replica modes:

- **Standby replicas** – They don't accept user connections by design. Use read-only replicas for read workloads.
- **Read-only replicas** – Check your security group settings, network ACLs, and parameter group configurations.

  For more information, see Control traffic to your AWS resources using security groups in the *Amazon VPC User Guide*, Control subnet traffic with network access control lists in the *Amazon VPC User Guide*, and Parameter groups for Amazon RDS.

Performance issues

If replica performance is poor, review the following suggestions:

- Ensure the replica has adequate compute and storage resources.
- Monitor the `ReplicaLag` metric in Amazon CloudWatch.
- Consider scaling up the replica DB instance class.

For information about modifying resources or instance classes, see Modifying an Amazon RDS DB instance.

For information monitoring replication lag, see Monitoring replication lag and Amazon CloudWatch metrics for Amazon RDS. For information about setting up CloudWatch alarms for replica lag, see Monitoring Amazon RDS metrics with Amazon CloudWatch.

Options for Amazon RDS for Db2 DB instances

The following shows the options, or additional features, that are available for Amazon RDS instances running the Db2 DB engine. To enable these options, you can add them to a custom option group, and then associate the option group with your DB instance. For more information about working with option groups, see Working with option groups.

Amazon RDS supports the following options for Db2:

Option	Option ID
Db2 audit logging	DB2_AUDIT

Db2 audit logging

With Db2 audit logging, Amazon RDS records database activity, including users logging on to the database and queries run against the database. RDS uploads the completed audit logs to your Amazon S3 bucket, using the AWS Identity and Access Management (IAM) role that you provide.

Topics

- Setting up Db2 audit logging
- Managing Db2 audit logging
- Viewing audit logs
- Troubleshooting Db2 audit logging

Setting up Db2 audit logging

To enable audit logging for an Amazon RDS for Db2 database, you enable the `DB2_AUDIT` option on the RDS for Db2 DB instance. Then, configure an audit policy to enable the feature for the specific database. To enable the option on the RDS for Db2 DB instance, you configure the option settings for the `DB2_AUDIT` option. You do so by providing the Amazon Resource Names (ARNs) for your Amazon S3 bucket and the IAM role with permissions to access your bucket.

To set up Db2 audit logging for an RDS for Db2 database, complete the following steps.

Topics

- Step 1: Create an Amazon S3 bucket
- Step 2: Create an IAM policy
- Step 3: Create an IAM role and attach your IAM policy
- Step 4: Configure an option group for Db2 audit logging
- Step 5: Configure the audit policy
- Step 6: Check the audit configuration

Step 1: Create an Amazon S3 bucket

If you haven't already done so, create an Amazon S3 bucket where Amazon RDS can upload your RDS for Db2 database's audit log files. The following restrictions apply to the S3 bucket that you use as a target for audit files:

- It must be in the same AWS Region as your RDS for Db2 DB instance.
- It must not be open to the public.
- The bucket owner must also be the IAM role owner.

To learn how to create an Amazon S3 bucket, see Creating a bucket in the *Amazon S3 User Guide*.

After you enable audit logging, Amazon RDS automatically sends the logs from your DB instance to the following locations:

- DB instance level logs – *bucket_name*/db2-audit-logs/*dbi_resource_id*/*date_time_utc*/
- Database level logs – *bucket_name*/db2-audit-logs/*dbi_resource_id*/*date_time_utc*/*db_name*/

Take note of the Amazon Resource Name (ARN) for your bucket. This information is needed to complete subsequent steps.

Step 2: Create an IAM policy

Create an IAM policy with the permissions required to transfer audit log files from your DB instance to your Amazon S3 bucket. This step assumes that you have an S3 bucket.

Before you create the policy, gather the following information:

- The ARN for your bucket.
- The ARN for your AWS Key Management Service (AWS KMS) key, if your bucket uses SSE-KMS encryption.

Create an IAM policy that includes the following permissions:

```
"s3:ListBucket",
 "s3:GetBucketAcl",
 "s3:GetBucketLocation",
 "s3:PutObject",
 "s3:ListMultipartUploadParts",
 "s3:AbortMultipartUpload",
 "s3:ListAllMyBuckets"
```

> **Note**
>
> Amazon RDS needs the `s3:ListAllMyBuckets` action internally to verify that the same AWS account owns both the S3 bucket and the RDS for Db2 DB instance.

If your bucket uses SSE-KMS encryption, also include the following permissions for your IAM role and AWS KMS key.

Include the following permissions to the policy for your IAM role.

```
"kms:GenerateDataKey",
 "kms:Decrypt"
```

Include the following permissions to the key policy for your AWS KMS key. Replace *111122223333* with your account number and *AR0A123456789EXAMPLE* with your IAM role name.

```
{
  "Sid": "Allow RDS role to use the key",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:sts::111122223333:assumed-role/AR0A123456789EXAMPLE/RDS-Db2Audit",
      "arn:aws:iam::111122223333:role/AR0A123456789EXAMPLE"
    ]
  },
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

You can create an IAM policy by using the AWS Management Console or the AWS Command Line Interface (AWS CLI).

Console

To create an IAM policy to allow Amazon RDS to access your Amazon S3 bucket

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**, and then choose **JSON**.
4. In **Add actions**, filter by **S3**. Add access **ListBucket**, **GetBucketAcl**, and **GetBucketLocation**.
5. For **Add a resource**, choose **Add**. For **Resource type**, choose **bucket**, and enter the name of your bucket. Then, choose **Add resource**.
6. Choose **Add new statement**.
7. In **Add actions**, filter by **S3**. Add access **PutObject**, **ListMultipartUploadParts**, and **AbortMultipartUpload**.
8. For **Add a resource**, choose **Add**. For **Resource type**, choose **object**, and enter *`your bucket name/*`*. Then, choose **Add resource**.
9. Choose **Add new statement**.
10. In **Add actions**, filter by **S3**. Add access **ListAllMyBuckets**.
11. For **Add a resource**, choose **Add**. For **Resource type**, choose **All Resources**. Then, choose **Add resource**.
12. If you're using your own KMS keys to encrypt the data:
    1. Choose **Add new statement**.
    2. In **Add actions**, filter by KMS. Add access **GenerateDataKey** and **Decrypt**.
    3. For **Add a resource**, choose **Add**. For **Resource type**, choose **All Resources**. Then, choose **Add resource**.
13. Choose **Next**.
14. For **Policy name**, enter a name for this policy.
15. (Optional) For **Description**, enter a description for this policy.
16. Choose **Create policy**.

AWS CLI

To create an IAM policy to allow Amazon RDS to access your Amazon S3 bucket

1. Run the create-policy command. In the following example, replace *iam_policy_name* and *amzn-s3-demo-bucket* with a name for your IAM policy and the name of your target Amazon S3 bucket.

   For Linux, macOS, or Unix:

```
aws iam create-policy \
    --policy-name iam_policy_name \
    --policy-document '{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "Statement1",
                "Effect": "Allow",
                "Action": [
                    "s3:ListBucket",
                    "s3:GetBucketAcl",
                    "s3:GetBucketLocation"
                ],
                "Resource": [
                    "arn:aws:s3:::amzn-s3-demo-bucket"
                ]
            },
            {
                "Sid": "Statement2",
                "Effect": "Allow",
                "Action": [
                    "s3:PutObject",
                    "s3:ListMultipartUploadParts",
                    "s3:AbortMultipartUpload"
                ],
                "Resource": [
                    "arn:aws:s3:::amzn-s3-demo-bucket/*"
                ]
            },
            {
                "Sid": "Statement3",
                "Effect": "Allow",
                "Action": [
```

```
                    "s3:ListAllMyBuckets"
                ],
                "Resource": [
                    "*"
                ]
            },
            {
                "Sid": "Statement4",
                "Effect": "Allow",
                "Action": [
                    "kms:GenerateDataKey",
                    "kms:Decrypt"
                ],
                "Resource": [
                    "*"
                ]
            }
        ]
  }'
```

For Windows:

```
aws iam create-policy ^
    --policy-name iam_policy_name ^
    --policy-document '{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "Statement1",
                "Effect": "Allow",
                "Action": [
                    "s3:ListBucket",
                    "s3:GetBucketAcl",
                    "s3:GetBucketLocation"
                ],
                "Resource": [
                    "arn:aws:s3:::amzn-s3-demo-bucket"
                ]
            },
            {
                "Sid": "Statement2",
                "Effect": "Allow",
                "Action": [
```

```
                    "s3:PutObject",
                    "s3:ListMultipartUploadParts",
                    "s3:AbortMultipartUpload"
                ],
                "Resource": [
                    "arn:aws:s3:::amzn-s3-demo-bucket/*"
                ]
            },
            {
                "Sid": "Statement3",
                "Effect": "Allow",
                "Action": [
                    "s3:ListAllMyBuckets"
                ],
                "Resource": [
                    "*"
                ]
            },
            {
                "Sid": "Statement4",
                "Effect": "Allow",
                "Action": [
                    "kms:GenerateDataKey",
                    "kms:Decrypt"
                ],
                "Resource": [
                    "*"
                ]
            }
        ]
    }'
```

2. After the policy is created, note the ARN of the policy. You need the ARN for Step 3: Create an IAM role and attach your IAM policy.

For information about creating an IAM policy, see Creating IAM policies in the IAM User Guide.

Step 3: Create an IAM role and attach your IAM policy

This step assumes that you created the IAM policy in Step 2: Create an IAM policy. In this step, you create an IAM role for your RDS for Db2 DB instance and then attach your IAM policy to the role.

You can create an IAM role for your DB instance by using the console or the AWS CLI.

Console

To create an IAM role and attach your IAM policy to it

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. For **Trusted entity type**, select **AWS service**.
5. For **Service or use case**, select **RDS**, and then select **RDS – Add Role to Database**.
6. Choose **Next**.
7. For **Permissions policies**, search for and select the name of the IAM policy that you created.
8. Choose **Next**.
9. For **Role name**, enter a role name.
10. (Optional) For **Description**, enter a description for the new role.
11. Choose **Create role**.

AWS CLI

To create an IAM role and attach your IAM policy to it

1. Run the create-role command. In the following example, replace *iam_role_name* with a name for your IAM role.

   For Linux, macOS, or Unix:

```
aws iam create-role \
    --role-name iam_role_name \
    --assume-role-policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "rds.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
```

```
      ]
    }'
```

For Windows:

```
aws iam create-role ^
    --role-name iam_role_name ^
    --assume-role-policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "rds.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }'
```

2. After the role is created, note the ARN of this role. You need this ARN for the next step, Step 4: Configure an option group for Db2 audit logging.
3. Run the attach-role-policy command. In the following example, replace *iam_policy_arn* with the ARN of the IAM policy that you created in Step 2: Create an IAM policy. Replace *iam_role_name* with the name of the IAM role that you just created.

   For Linux, macOS, or Unix:

   ```
   aws iam attach-role-policy \
      --policy-arn iam_policy_arn \
      --role-name iam_role_name
   ```

   For Windows:

   ```
   aws iam attach-role-policy ^
      --policy-arn iam_policy_arn ^
      --role-name iam_role_name
   ```

For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.

Step 4: Configure an option group for Db2 audit logging

The process for adding the Db2 audit logging option to an RDS for Db2 DB instance is as follows:

1. Create a new option group, or copy or modify an existing option group.
2. Add and configure all required options.
3. Associate the option group with the DB instance.

After you add the Db2 audit logging option, you don't need to restart your DB instance. As soon as the option group is active, you can create audits and store audit logs in your S3 bucket.

To add and configure Db2 audit logging on a DB instance's option group

1. Choose one of the following:
   - Use an existing option group.
   - Create a custom DB option group, and use that option group. For more information, see Creating an option group.
2. Add the **DB2_AUDIT** option to the option group, and configure the option settings. For more information about adding options, see Adding an option to an option group.
   - For **IAM_ROLE_ARN**, enter the ARN of the IAM role that you created in the section called "Create an IAM role and attach your IAM policy".
   - For **S3_BUCKET_ARN**, enter the ARN of the S3 bucket to use for your Db2 audit logs. The bucket must be in the same Region as your RDS for Db2 DB instance. The policy associated with the IAM role you entered must allow the required operations on this resource.
3. Apply the option group to a new or existing DB instance. Choose one of the following:
   - If you are creating a new DB instance, apply the option group when you launch the instance.
   - On an existing DB instance, apply the option group by modifying the instance and then attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

Step 5: Configure the audit policy

To configure the audit policy for your RDS for Db2 database, connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. Then, call the

`rdsadmin.configure_db_audit` stored procedure with the DB name of your database and the applicable parameter values.

The following example connects to the database and configures an audit policy for `testdb` with the categories AUDIT, CHECKING, OBJMAINT, SECMAINT, SYSADMIN, and VALIDATE. The status value `BOTH` logs success and failures, and the `ERROR TYPE` is `NORMAL` by default. For more information about how to use this stored procedure, see the section called "rdsadmin.configure_db_audit".

```
db2 "connect to rdsadmin user master_user using master_password"
db2 "call rdsadmin.configure_db_audit('testdb', 'ALL', 'BOTH', ?)"
```

Step 6: Check the audit configuration

To make sure that your audit policy is set up correctly, check the status of your audit configuration.

To check the configuration, connect to the `rdsadmin` database using the master username and master password for your RDS for Db2 DB instance. Then, run the following SQL statement with the DB name of your database. In the following example, the DB name is *testdb*.

```
db2 "select task_id, task_type, database_name, lifecycle,
    varchar(bson_to_json(task_input_params), 500) as task_params,
    cast(task_output as varchar(500)) as task_output
    from table(rdsadmin.get_task_status(null,'testdb','CONFIGURE_DB_AUDIT'))"

Sample Output

TASK_ID              TASK_TYPE            DATABASE_NAME       LIFECYCLE
-------------------- -------------------- --------------- -------------
                   2  CONFIGURE_DB_AUDIT            DB2DB       SUCCESS

... continued ...
TASK_PARAMS
------------------------------------------------------------
{ "AUDIT_CATEGORY" : "ALL", "CATEGORY_SETTING" : "BOTH" }

... continued ...
                                        TASK_OUTPUT
-------------------------------------------------
2023-12-22T20:27:03.029Z Task execution has started.
```

```
2023-12-22T20:27:04.285Z Task execution has completed successfully.
```

Managing Db2 audit logging

After you set up Db2 audit logging, you can modify the audit policy for a specific database, or disable audit logging at the database level or for the entire DB instance. You can also change the Amazon S3 bucket where your log files are uploaded to.

Topics

- Modifying a Db2 audit policy
- Modifying the location of your log files
- Disabling Db2 audit logging

Modifying a Db2 audit policy

To modify the audit policy for a specific RDS for Db2 database, run the `rdsadmin.configure_db_audit` stored procedure. With this stored procedure, you can change the categories, category settings, and error type configuration of the audit policy. For more information, see the section called "rdsadmin.configure_db_audit".

Modifying the location of your log files

To change the Amazon S3 bucket where your log files are uploaded to, do one of the following:

- Modify the current option group attached to your RDS for Db2 DB instance – Update the `S3_BUCKET_ARN` setting for the `DB2_AUDIT` option to point to the new bucket. Also, make sure to update the IAM policy attached to the IAM role specified by the `IAM_ROLE_ARN` setting in the attached option group. This IAM policy must provide your new bucket with the required access permissions. For information about the permissions required in the IAM policy, see Create an IAM policy.
- Attach your RDS for Db2 DB instance to a different option group – Modify your DB instance to change the option group that's attached to it. Make sure that the new option group is configured with the correct `S3_BUCKET_ARN` and `IAM_ROLE_ARN` settings. For information about how to configure these settings for the `DB2_AUDIT` option, see Configure an option group.

When you modify the option group, make sure that you apply the changes immediately. For more information, see the section called "Modifying a DB instance".

Disabling Db2 audit logging

To disable Db2 audit logging, do one of the following:

- Disable audit logging for the RDS for Db2 DB instance – Modify your DB instance and remove the option group with the `DB2_AUDIT` option from it. For more information, see the section called "Modifying a DB instance".
- Disable audit logging for a specific database – Stop audit logging and remove the audit policy by calling `rdsadmin.disable_db_audit` with the DB name of your database. For more information, see the section called "rdsadmin.disable_db_audit".

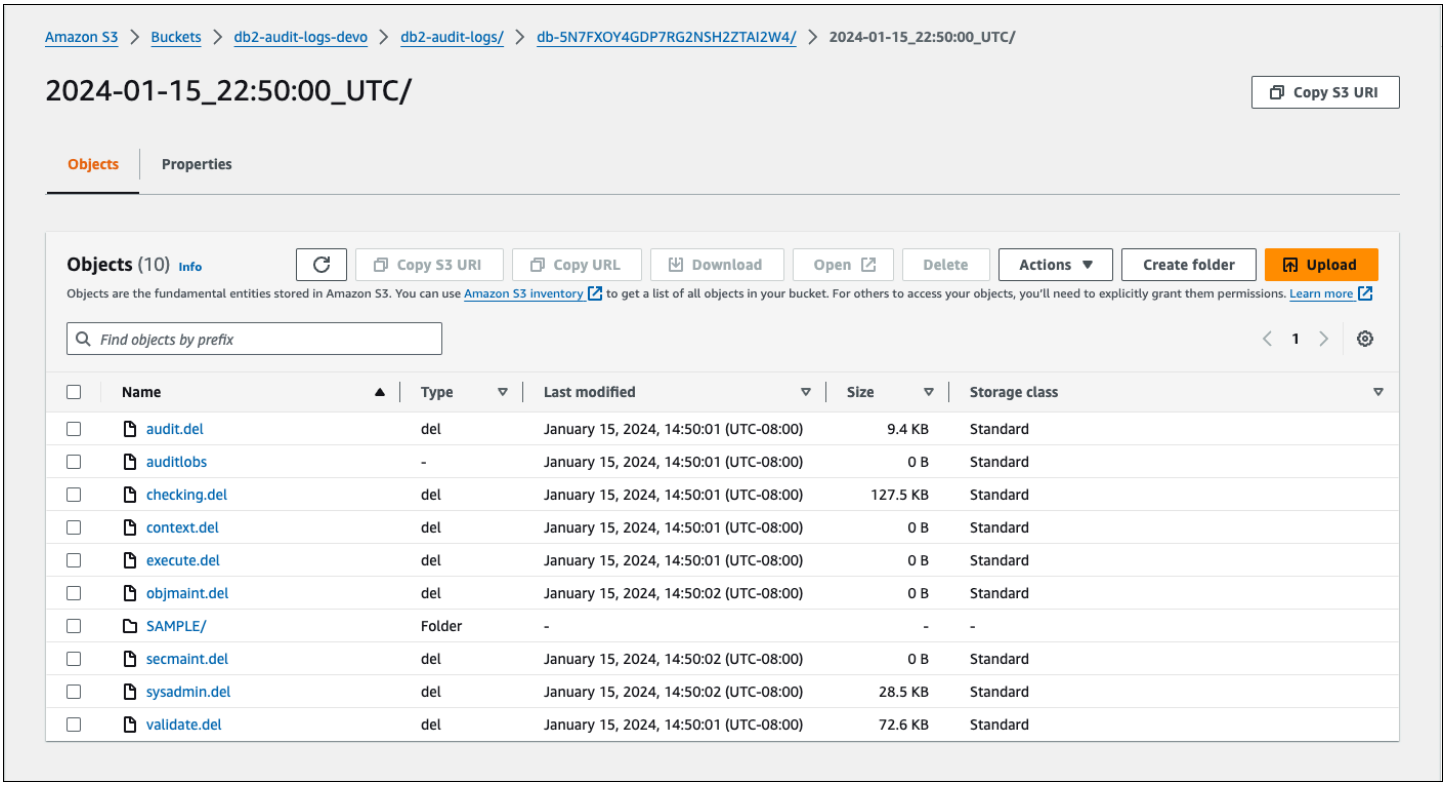
```
db2 "call rdsadmin.disable_db_audit(
    'db_name',
    ?)"
```

Viewing audit logs

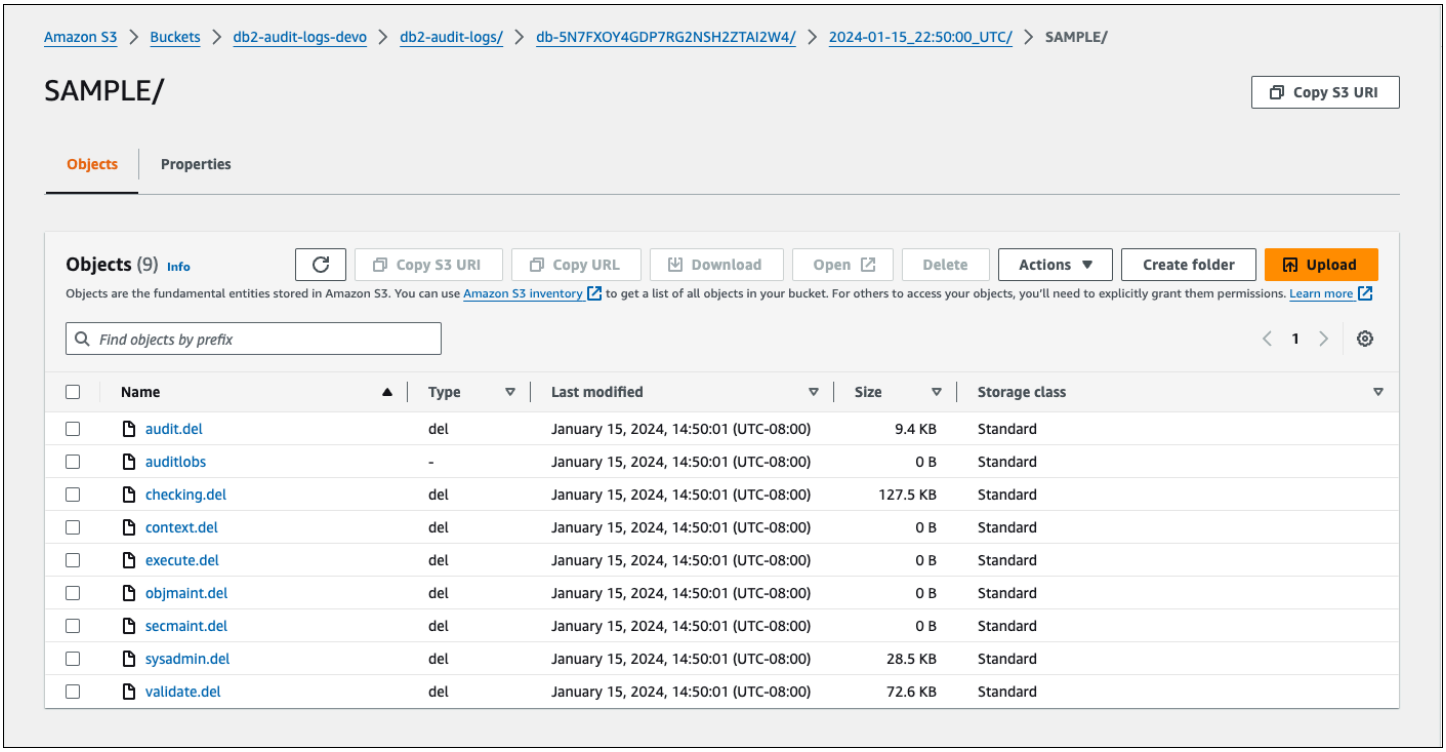
After you enable Db2 audit logging, wait for at least one hour before viewing the audit data in your Amazon S3 bucket. Amazon RDS automatically sends the logs from your RDS for Db2 DB instance to the following locations:

- DB instance level logs – *bucket_name*`/db2-audit-logs/`*dbi_resource_id*`/`*date_time_utc*`/`
- Database level logs – *bucket_name*`/db2-audit-logs/`*dbi_resource_id*`/`*date_time_utc*`/`*db_name*`/`

The following example screenshot of the Amazon S3 console shows a list of folders for RDS for Db2 DB instance level log files.



The following example screenshot of the Amazon S3 console shows database level log files for the RDS for Db2 DB instance.



Troubleshooting Db2 audit logging

Use the following information to troubleshoot common issues with Db2 audit logging.

Can't configure the audit policy

If calling the stored procedure `rdsadmin.configure_db_audit` returns an error, it could be that the option group with the `DB2_AUDIT` option isn't associated with the RDS for Db2 DB instance. Modify the DB instance to add the option group, and then try calling the stored procedure again. For more information, see Modifying an Amazon RDS DB instance.

No data in the Amazon S3 bucket

If logging data is missing from the Amazon S3 bucket, check the following:

- The Amazon S3 bucket is in the same Region as your RDS for Db2 DB instance.
- The role you specified in the `IAM_ROLE_ARN` option setting is configured with the required permissions to upload logs to your Amazon S3 bucket. For more information, see Create an IAM policy.
- The ARNs for the `IAM_ROLE_ARN` and `S3_BUCKET_ARN` option settings are correct in the option group associated with your RDS for Db2 DB instance. For more information, see Configure an option group.

You can check the task status of your audit logging configuration by connecting to the database and running a SQL statement. For more information, see Check the audit configuration.

You can also check events to find out more about why logs might be missing. For information about how to view events, see the section called “Viewing logs, events, and streams in the Amazon RDS console”.

External stored procedures for Amazon RDS for Db2

You can create external routines and register them with your Amazon RDS for Db2 databases as external stored procedures. Currently, RDS for Db2 only supports Java-based routines for external stored procedures.

Java-based external stored procedures

Java-based external stored procedures are external Java routines that you register with your RDS for Db2 database as external stored procedures.

Topics

- Limitations for Java-based external stored procedures
- Configuring Java-based external stored procedures

Limitations for Java-based external stored procedures

Before you develop your external routine, consider the following limitations and restrictions.

To create your external routine, make sure to use the Java Development Kit (JDK) provided by Db2. For more information, see Java software support for Db2 database products.

Your Java program can create files only in the `/tmp` directory, and Amazon RDS doesn't support enabling executable or Set User ID (SUID) permissions on these files. Your Java program also can't use socket system calls or the following system calls:

- _sysctl
- acct
- afs_syscall
- bpf
- capset
- chown
- chroot
- create_module
- delete_module

- fanotify_init
- fanotify_mark
- finit_module
- fsconfig
- fsopen
- fspick
- get_kernel_syms
- getpmsg
- init_module
- mount
- move_mount
- nfsservctl
- open_by_handle_at
- open_tree
- pivot_root
- putpmsg
- query_module
- quotactl
- reboot
- security
- setdomainname
- setfsuid
- sethostname
- sysfs
- tuxcall
- umount2
- uselib
- ustat
- vhangup
- vserver

For additional restrictions on external routines for Db2, see Restrictions on external routines in the IBM Db2 documentation.

Configuring Java-based external stored procedures

To configure an external stored procedure, create a .jar file with your external routine, install it on your RDS for Db2 database, and then register it as an external stored procedure.

Topics

- Step 1: Enable external stored procedures
- Step 2: Install the .jar file with your external routine
- Step 3: Register the external stored procedure
- Step 4: Validate the external stored procedure

Step 1: Enable external stored procedures

To enable external stored procedures, in a custom parameter group associated with your DB instance, set the parameter `db2_alternate_authz_behaviour` to one of the following values:

- `EXTERNAL_ROUTINE_DBADM` – Implicitly grants any user, group, or role with `DBADM` authority the `CREATE_EXTERNAL_ROUTINE` permission.
- `EXTERNAL_ROUTINE_DBAUTH` – Allows a user with `DBADM` authority to grant `CREATE_EXTERNAL_ROUTINE` permission to any user, group, or role. In this case, no user, group, or role is implicitly granted this permission, not even a user with `DBADM` authority.

For more information about this setting, see GRANT (database authorities) statement in the IBM Db2 documentation.

You can create and modify a custom parameter group by using the AWS Management Console, the AWS CLI, or the Amazon RDS API.

Console

To configure the db2_alternate_authz_behaviour parameter in a custom parameter group

1. If you want to use a different custom DB parameter group than the one your DB instance is using, create a new DB parameter group. If you're using the bring your own license (BYOL) model, make sure that the new custom parameter group includes the IBM IDs. For information

about these IDs, see the section called "IBM IDs for bring your own license (BYOL) for Db2". For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.

2. Set the value for the `db2_alternate_authz_behaviour` parameter in your custom parameter group. For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.

AWS CLI

To configure the db2_alternate_authz_behaviour parameter in a custom parameter group

1. If you want to use a different custom DB parameter group than the one your DB instance is using, create a custom parameter group by running the create-db-parameter-group command. If you're using the bring your own license (BYOL) model, make sure that the new custom parameter group includes the IBM IDs. For information about these IDs, see the section called "IBM IDs for bring your own license (BYOL) for Db2".

   Include the following required options:

   - `--db-parameter-group-name` – A name for the parameter group that you are creating.
   - `--db-parameter-group-family` – The Db2 engine edition and major version. Valid values are `db2-se-11.5` and `db2-ae-11.5`.
   - `--description` – A description for this parameter group.

   For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.

   The following example shows you how to create a custom parameter group named `MY_EXT_SP_PARAM_GROUP` for the parameter group family `db2-se-11.5`.

   For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
--region us-east-1 \
--db-parameter-group-name MY_EXT_SP_PARAM_GROUP \
--db-parameter-group-family db2-se-11.5 \
--description "test db2 external routines"
```

For Windows:

```
aws rds create-db-parameter-group ^
--region us-east-1 ^
--db-parameter-group-name MY_EXT_SP_PARAM_GROUP ^
--db-parameter-group-family db2-se-11.5 ^
--description "test db2 external routines"
```

2. Modify the `db2_alternate_authz_behaviour` parameter in your custom parameter group by running the [modify-db-parameter-group] command.

   Include the following required options:

   - `--db-parameter-group-name` – The name of the parameter group that you created.
   - `--parameters` – An array of parameter names, values, and the application methods for the parameter update.

   For more information about modifying a parameter group, see [Modifying parameters in a DB parameter group in Amazon RDS].

   The following example shows you how to modify the parameter group `MY_EXT_SP_PARAM_GROUP` by setting the value of `db2_alternate_authz_behaviour` to `EXTERNAL_ROUTINE_DBADM`.

   For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name MY_EXT_SP_PARAM_GROUP \
    --parameters
 "ParameterName='db2_alternate_authz_behaviour',ParameterValue='EXTERNAL_ROUTINE_DBADM',App
```

   For Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name MY_EXT_SP_PARAM_GROUP ^
    --parameters
 "ParameterName='db2_alternate_authz_behaviour',ParameterValue='EXTERNAL_ROUTINE_DBADM',App
```

RDS API

To configure the db2_alternate_authz_behaviour parameter in a custom parameter group

1. If you want to use a different custom DB parameter group than the one your DB instance is using, create a new DB parameter group by using the Amazon RDS API CreateDBParameterGroup operation. If you're using the bring your own license (BYOL) model, make sure that the new custom parameter group includes the IBM Db2 IDs. For information about these IDs, see the section called “IBM IDs for bring your own license (BYOL) for Db2”.

   Include the following required parameters:

   - `DBParameterGroupName`
   - `DBParameterGroupFamily`
   - `Description`

   For more information about creating a DB parameter group, see Creating a DB parameter group in Amazon RDS.
2. Modify the `db2_alternate_authz_behaviour` parameter in your custom parameter group that you created by using the RDS API ModifyDBParameterGroup operation.

   Include the following required parameters:

   - `DBParameterGroupName`
   - `Parameters`

   For more information about modifying a parameter group, see Modifying parameters in a DB parameter group in Amazon RDS.

Step 2: Install the .jar file with your external routine

After you create your Java routine, create the .jar file and then run `db2 "call sqlj.install_jar('file:`*`file_path`*`',`*`jar_ID`*`)"` to install it on your RDS for Db2 database.

The following example shows you how to create a Java routine and install it on an RDS for Db2 database. The example includes sample code for a simple routine that you can use to test the process. This example makes the following assumptions:

- The Java code is compiled on a server where Db2 is installed. This is a best practice because not compiling with the IBM-provided JDK can result in unexplained errors.
- The server has the RDS for Db2 database cataloged locally.

If you'd like to try out the process with the following sample code, copy it and then save it to a file named `MYJAVASP.java`.

```
import java.sql.*;
public class MYJAVASP
{
public static void my_JAVASP (String inparam) throws SQLException, Exception
{
try
{
// Obtain the calling context's connection details.
Connection myConn = DriverManager.getConnection("jdbc:default:connection");
String myQuery = "INSERT INTO TEST.TEST_TABLE VALUES (?, CURRENT DATE)";
PreparedStatement myStmt = myConn.prepareStatement(myQuery);
myStmt.setString(1, inparam);
myStmt.executeUpdate();
}
catch (SQLException sql_ex)
{
throw sql_ex;
}
catch (Exception ex)
{
throw ex;
}
}
```

The following command compiles the Java routine.

```
~/sqllib/java/jdk64/bin/javac MYJAVASP.java
```

The following command creates the .jar file.

```
~/sqllib/java/jdk64/bin/jar cvf MYJAVASP.jar MYJAVASP.class
```

The following commands connect to the database named MY_DB2_DATABASE and install the .jar file.

```
db2 "connect to MY_DB2_DATABASE user master_username using master_password"

db2 "call sqlj.install_jar('file:/tmp/MYJAVASP.jar','MYJAVASP')"
db2 "call sqlj.refresh_classes()"
```

Step 3: Register the external stored procedure

After you install the .jar file on your RDS for Db2 database, register it as a stored procedure by running the db2 CREATE PROCEDURE or db2 REPLACE PROCEDURE command.

The following example shows you how to connect to the database and register the Java routine created in the previous step as a stored procedure.

```
db2 "connect to MY_DB2_DATABASE user master_username using master_password"

create procedure TESTSP.MYJAVASP (in input char(6))
specific myjavasp
dynamic result sets 0
deterministic
language java
parameter style java
no dbinfo
fenced
threadsafe
modifies sql data
program type sub
external name 'MYJAVASP!my_JAVASP';
```

Step 4: Validate the external stored procedure

Use the following steps to test the sample external stored procedure that was registered in the previous step.

To validate the external stored procedure

1. Create a table like TEST.TEST_TABLE in the following example.

   ```
   db2 "create table TEST.TEST_TABLE(C1 char(6), C2 date)"
   ```

2. Call the new external stored procedure. The call returns a status of `0`.

```
db2 "call TESTSP.MYJAVASP('test')"
Return Status = 0
```

3. Query the table you created in step 1 to verify the results of the stored procedure call.

```
db2 "SELECT * from TEST.TEST_TABLE"
```

The query produces output similar to the following example:

```
C1     C2
------ ----------
test   02/05/2024
```

Known issues and limitations for Amazon RDS for Db2

The following items are known issues and limitations for working with Amazon RDS for Db2:

Topics

- Authentication limitation
- Non-fenced routines
- Non-automatic storage tablespaces during migration
- Setting the db2_compatibility_vector parameter
- Migrating databases that contain INVALID packages

Authentication limitation

Amazon RDS sets `db2auth` to `JCC_ENFORCE_SECMEC` by default. However, if you don't want to enforce userid and password encryption over the wire, you can override this setting by changing the `db2auth` parameter to `CLEAR_TEXT` in the parameter group. For more information, see Modifying parameters in a DB parameter group in Amazon RDS.

Non-fenced routines

RDS for Db2 doesn't support the creation of non-fenced routines and the migration of these routines by backing up and restoring data. To check if your database contains any non-fenced routines, run the following SQL command:

```
SELECT 'COUNT:' || count(*) FROM SYSCAT.ROUTINES where fenced='N' and routineschema not
 in ('SQLJ','SYSCAT','SYSFUN','SYSIBM','SYSIBMADM','SYSPROC','SYSTOOLS')
```

Non-automatic storage tablespaces during migration

RDS for Db2 doesn't support the creation of new non-automatic storage tablespaces. When you use native restore for a one-time migration of your database, RDS for Db2 automatically converts your non-automatic storage tablespaces to automatic ones, and then restores your database to RDS for Db2. For information about one-time migrations, see Migrating from Linux to Linux for Amazon RDS for Db2 and Migrating from AIX or Windows to Linux for Amazon RDS for Db2.

Setting the db2_compatibility_vector parameter

With Amazon RDS, you can create an initial database when you create the DB instance and then modify parameters in an associated parameter group. However, for Db2, if you want to set the `db2_compatibility_vector` parameter in a parameter group, you must first modify the parameter in a custom parameter group, create the DB instance without a database, and then create a database using the `rdsadmin.create_database` stored procedure.

To set the `db2_compatibility_vector` parameter

1. Create a custom parameter group. (You can't modify parameters in default parameter groups.)
2. Modify the parameter.
3. Create a DB instance.
4. Create a database using the `rdsadmin.create_database` stored procedure.
5. Associate the parameter group with the DB instance that contains the database.

Migrating databases that contain INVALID packages

If you migrate Db2 databases that contain `INVALID` packages to RDS for Db2 by using the `RESTORE` command, you could encounter issues when you start to use the databases. `INVALID` packages can cause issues because of the authorization setup for the DB instance user `rdsdb` and the removal of authorization from `PUBLIC`. `INVALID` packages cause the following commands to fail:

- `db2updv115`
- `db2 "call SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS()"`

Before migrating your database with the `RESTORE` command, ensure that your database doesn't contain `INVALID` packages by running the following command:

```
db2 "SELECT 'COUNT:' || count(*) FROM SYSCAT.INVALIDOBJECTS"
```

If the command returns a count greater than zero, then call the following command:

```
db2 "call SYSPROC.ADMIN_REVALIDATE_DB_OBJECTS()"
```

Afterwards, call the previous command to confirm that your database no longer contains `INVALID` packages.

```
db2 "SELECT 'COUNT:' || count(*) FROM SYSCAT.INVALIDOBJECTS"
```

Now you are ready to make a backup of your database and restore it to your RDS for Db2 DB instance.

Amazon RDS for Db2 stored procedure reference

You can manage your Amazon RDS for Db2 DB instances running the Db2 engine by calling built-in stored procedures.

Stored procedure	Category	Description
the section called “rdsadmin.activate_database”	Databases	Use the `rdsadmin.activate_database` stored procedure to activate a database on a standalone RDS for Db2 DB instance.
the section called “rdsadmin.add_groups”	Granting and revoking privileges	Use the `rdsadmin.add_groups` stored procedure to add one or more groups to a user for a database on an RDS for Db2 DB instance.
the section called “rdsadmin.add_user”	Granting and revoking privileges	Use the `rdsadmin.add_user` stored procedure to add a user to an authorization list for a database on an RDS for Db2 DB instance.
the section called “rdsadmin.alter_bufferpool”	Buffer pools	Use the `rdsadmin.alter_bufferpool` stored procedure to modify a buffer pool for a database on an RDS for Db2 DB instance.
the section called “rdsadmin.alter_tablespace”	Tablespaces	Use the `rdsadmin.alter_tablespace` stored procedure to modify a tablespace for a database on an RDS for Db2 DB instance.
the section called “rdsadmin.backup_database”	Database	Use the `rdsadmin.backup_database` stored procedure to back up a database on an RDS for Db2 DB instance to an Amazon S3 bucket. Then you can restore the backup from Amazon S3 to an RDS for Db2 DB instance or to another location such as a local server.
the section called “rdsadmin.catalog_storage_access”	Storage access	Use the `rdsadmin.catalog_storage_access` stored procedure to catalog a storage alias for accessing an Amazon S3 bucket with

Stored procedure	Category	Description
		Db2 data files for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.change_password"	Granting and revoking privileges	Use the `rdsadmin.change_password` stored procedure to change a user's password for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.complete_rollforward"	Databases	Use the `rdsadmin.complete_rollforward` stored procedure to bring a database on an RDS for Db2 DB instance online from a `ROLL-FORWARD PENDING` state. A `ROLL-FORWARD PENDING` state occurs when you called the section called "rdsadmin.rollforward_database" but set the `complete_rollforward` parameter to `FALSE`.
the section called "rdsadmin.configure_db_audit"	Audit policies	Use the `rdsadmin.configure_db_audit` stored procedure to modify an audit policy for a database on an RDS for Db2 DB instance. If no audit policy exists, running this stored procedure creates an audit policy.
the section called "rdsadmin.create_bufferpool"	Buffer pools	Use the `rdsadmin.create_bufferpool` stored procedure to create a buffer pool for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.create_database"	Databases	Use the `rdsadmin.create_database` stored procedure to create a database on an RDS for Db2 DB instance.
the section called "rdsadmin.create_role"	Granting and revoking privileges	Use the `rdsadmin.create_role` stored procedure to create a role to attach to a database on an RDS for Db2 DB instance.

Stored procedure	Category	Description
the section called "rdsadmin.create_tablespace"	Tablespaces	Use the `rdsadmin.create_tablespace` stored procedure to create a tablespace for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.db2pd_command"	Databases	Use the `rdsadmin.db2pd_command` stored procedure collect information about a database on an RDS for Db2 DB instance. This information can help with monitoring and troubleshooting databases in RDS for Db2.
the section called "rdsadmin.db2support_command"	Databases	Use the `rdsadmin.db2support_command` stored procedure to collect diagnostic information about a database on an RDS for Db2 DB instance and upload it to an Amazon S3 bucket.
the section called "rdsadmin.dbadm_grant"	Granting and revoking privileges	Use the `rdsadmin.dbadm_grant` stored procedure to grant one or more authorization types (`DBADM`, `ACCESSCTRL` , or `DATAACCESS`) to one or more roles, users, or groups for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.dbadm_revoke"	Granting and revoking privileges	Use the `rdsadmin.dbadm_revoke` stored procedure to revoke one or more authorization types (`DBADM`, `ACCESSCTRL` , or `DATAACCESS`) from one or more roles, users, or groups for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.deactivate_database"	Databases	Use the `rdsadmin.deactivate_database` stored procedure to deactivate a database on an RDS for Db2 DB instance. You can deactivate databases to conserve memory resources.

Stored procedure	Category	Description
the section called “rdsadmin.disable_db_audit”	Audit policies	Use the `rdsadmin.disable_db_audit` stored procedure to stop audit logging and remove an audit policy from a database on an RDS for Db2 DB instance.
the section called “rdsadmin.drop_bufferpool”	Buffer pools	Use the `rdsadmin.drop_bufferpool` stored procedure to drop a buffer pool from a database on an RDS for Db2 DB instance.
the section called “rdsadmin.drop_database”	Databases	Use the `rdsadmin.drop_database` stored procedure to drop a database from an RDS for Db2 DB instance.
the section called “rdsadmin.drop_role”	Granting and revoking privileges	Use the `rdsadmin.drop_role` stored procedure to delete a role from a database on an RDS for Db2 DB instance.
the section called “rdsadmin.drop_tablespace”	Tablespaces	Use the `rdsadmin.drop_tablespace` stored procedure to drop a tablespace from a database on an RDS for Db2 DB instance.
the section called “rdsadmin.fgac_command”	Databases	Use the `rdsadmin.fgac_command` stored procedure to control access at the row or column level to table data in your database on an RDS for Db2 DB instance.
the section called “rdsadmin.force_application”	Databases	Use the `rdsadmin.force_application` stored procedure to force applications off of a database on an RDS for Db2 DB instance to perform maintenance.

Stored procedure	Category	Description
the section called "rdsadmin.grant_role"	Granting and revoking privileges	Use the `rdsadmin.grant_role` stored procedure to assign a role to a grantee role, user, or group in a database on an RDS for Db2 DB instance. You can also use this stored procedure to give the grantee role `DBADM` authorization to assign roles.
the section called "rdsadmin.list_archive_log_information"	Databases	Use the `rdsadmin.list_archive_log_information` stored procedure to return information about archive logs for a database on an RDS for Db2 DB instance. This information includes details such as size and creation date of individual log files, and the total storage used by the archive log files.
the section called "rdsadmin.list_sid_group_mapping"	Granting and revoking privileges	Use the `rdsadmin.list_sid_group_mapping` stored procedure to return a list of all security ID (SID) and Active Directory group mappings configured on an RDS for Db2 DB instance.
the section called "rdsadmin.list_users"	Granting and revoking privileges	Use the `rdsadmin.list_users` stored procedure to return a list of users on an authorization list for a database on an RDS for Db2 DB instance.

Stored procedure	Category	Description
the section called "rdsadmin.reactivate_database"	Databases	Use the `rdsadmin.reactivate_database` stored procedure to reactivate a database on an RDS for Db2 DB instance after you make database configuration changes. For a database on a standalone DB instance, you can use either this stored procedure or the rdsadmin.activate_database stored procedure. For a database on a replica source DB instance, you must use the `rdsadmin.reactivate_database` stored procedure.
the section called "rdsadmin.remove_groups"	Granting and revoking privileges	Use the `rdsadmin.remove_groups` stored procedure to remove one or more groups from a user for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.remove_sid_group_mapping"	Granting and revoking privileges	Use the `rdsadmin.remove_sid_group_mapping` stored procedure to remove a security ID (SID) and its corresponding Active Directory group mapping from an RDS for Db2 DB instance.
the section called "rdsadmin.remove_user"	Granting and revoking privileges	Use the `rdsadmin.remove_user` stored procedure to remove a user from an authorization list for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.rename_tablespace"	Tablespaces	Use the `rdsadmin.rename_tablespace` stored procedure to rename a tablespace for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.restore_database"	Databases	Use the `rdsadmin.restore_database` stored procedure to restore a database on an RDS for Db2 DB instance from an Amazon S3 bucket.

Stored procedure	Category	Description
the section called "rdsadmin.revoke_role"	Granting and revoking privileges	Use the `rdsadmin.revoke_role` stored procedure to revoke a role from a grantee role, user, or group in a database on an RDS for Db2 DB instance.
the section called "rdsadmin.rollforward_database"	Databases	Use the `rdsadmin.rollforward_database` stored procedure to bring a database on an RDS for Db2 DB instance online, and to apply transaction logs after you restored a database on an RDS for Db2 DB instance by calling the section called "rdsadmin.restore_database".
the section called "rdsadmin.rollforward_status"	Databases	Use the `rdsadmin.rollforward_status` stored procedure to query the rollforward status of calling the the section called "rdsadmin.rollforward_database" or the section called "rdsadmin.complete_rollforward" stored procedure on an RDS for Db2 DB instance.
the section called "rdsadmin.set_archive_log_retention"	Databases	Use the `rdsadmin.set_archive_log_retention` stored procedure to configure how long to retain archive log files for a database on an RDS for Db2 DB instance. You can also use this stored procedure to disable archive log retention.
the section called "rdsadmin.set_configuration"	Databases	Use the `rdsadmin.set_configuration` stored procedure to configure certain settings for a database on an RDS for Db2 DB instance.

Stored procedure	Category	Description
the section called "rdsadmin.set_sid_group_mapping"	Granting and revoking privileges	Use the `rdsadmin.set_sid_group_mapping` stored procedure to create a mapping between a security ID (SID) and the corresponding Active Directory group on an RDS for Db2 DB instance.
the section called "rdsadmin.show_archive_log_retention"	Databases	Use the `rdsadmin.show_archive_log_retention` stored procedure to return the current archive log retention setting for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.show_configuration"	Databases	Use the `rdsadmin.show_configuration` stored procedure to return one or more settings that are modifiable for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.uncatalog_storage_access"	Storage access	Use the `rdsadmin.uncatalog_storage_access` stored procedure to remove a storage alias for accessing an Amazon S3 bucket with Db2 data files.
the section called "rdsadmin.update_db_param"	Databases	Use the `rdsadmin.update_db_param` stored procedure to update database parameters for a database on an RDS for Db2 DB instance.
the section called "rdsadmin.enable_archive_log_copy"	Databases	Use the `rdsadmin.enable_archive_log_copy` stored procedure to enables RDS Db2 database archive log copy to Amazon S3.
the section called "rdsadmin.disable_archive_log_copy"	Databases	Use the `rdsadmin.disable_archive_log_copy` stored procedure to disable RDS Db2 database archive log copy to Amazon S3.

Topics

- Considerations for Amazon RDS for Db2 stored procedures
- Stored procedures for granting and revoking privileges for RDS for Db2
- Stored procedures for audit policies for RDS for Db2
- Stored procedures for buffer pools for RDS for Db2
- Stored procedures for databases for RDS for Db2
- Stored procedures for storage access for RDS for Db2
- Stored procedures for tablespaces for RDS for Db2

Considerations for Amazon RDS for Db2 stored procedures

Before using the Amazon RDS system stored procedures for RDS for Db2 DB instances running the Db2 engine, review the following information:

- Before running the stored procedures, you must first connect to the `rdsadmin` database as the master user for your RDS for Db2 DB instance. In the following example, replace *`master_username`* and *`master_password`* with your own information.

  ```
  db2 "connect to rdsadmin user master_username using master_password"
  ```

- The stored procedures return the ERR_MESSAGE parameter, which indicates whether the stored procedure ran successfully or not and why it didn't run successfully.

  Examples

  The following example indicates that the stored procedure ran successfully.

  ```
  Parameter Name : ERR_MESSAGE
  Parameter Value : -
  Return Status = 0
  ```

  The following example indicates that the stored procedure didn't run successfully because the Amazon S3 bucket name used in the stored procedure wasn't valid.

  ```
  Parameter Name : ERR_MESSAGE
  Parameter Value : Invalid S3 bucket name
  Return Status = -1006
  ```

  For error messages returned when calling stored procedures, see the section called “Stored procedure errors”.

For information about checking the status of a stored procedure, see rdsadmin.get_task_status.

Stored procedures for granting and revoking privileges for RDS for Db2

The built-in stored procedures described in this topic manage users, roles, groups, and authorization for Amazon RDS for Db2 databases. To run these procedures, the master user must first connect to the `rdsadmin` database.

For tasks that use these stored procedures, see Granting and revoking privileges and Setting up Kerberos authentication.

Refer to the following built-in stored procedures for information about their syntax, parameters, usage notes, and examples.

Stored procedures

- rdsadmin.create_role
- rdsadmin.grant_role
- rdsadmin.revoke_role
- rdsadmin.drop_role
- rdsadmin.add_user
- rdsadmin.change_password
- rdsadmin.list_users
- rdsadmin.remove_user
- rdsadmin.add_groups
- rdsadmin.remove_groups
- rdsadmin.dbadm_grant
- rdsadmin.dbadm_revoke
- rdsadmin.set_sid_group_mapping
- rdsadmin.list_sid_group_mapping
- rdsadmin.remove_sid_group_mapping

rdsadmin.create_role

Creates a role.

Syntax

```
db2 "call rdsadmin.create_role(
```

```
    'database_name',
    'role_name')"
```

Parameters

The following parameters are required:

database_name

The name of the database the command will run on. The data type is `varchar`.

role_name

The name of the role that you want to create. The data type is `varchar`.

Usage notes

For information about checking the status of creating a role, see rdsadmin.get_task_status.

Examples

The following example creates a role called `MY_ROLE` for database `DB2DB`.

```
db2 "call rdsadmin.create_role(
    'DB2DB',
    'MY_ROLE')"
```

rdsadmin.grant_role

Assigns a role to a role, user, or group.

Syntax

```
db2 "call rdsadmin.grant_role(
    ?,
    'database_name',
    'role_name',
    'grantee',
    'admin_option')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs the unique identifier for the task. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database the command will run on. The data type is `varchar`.

role_name

The name of the role that you want to create. The data type is `varchar`.

grantee

The role, user, or group to receive authorization. The data type is `varchar`. Valid values: `ROLE`, `USER`, `GROUP`, `PUBLIC`.

Format must be value followed by name. Separate multiple values and names with commas. Example: 'USER *user1*, *user2*, GROUP *group1*, *group2*'. Replace the names with your own information.

The following input parameter is optional:

admin_option

Specifies whether the grantee `ROLE` has `DBADM` authorization to assign roles. The data type is `char`. The default is `N`.

Usage notes

For information about checking the status of assigning a role, see rdsadmin.get_task_status.

Examples

Example 1: Assigning role to role, user, and group, and granting authorization

The following example assigns a role called `ROLE_TEST` for database `TESTDB` to the role called `role1`, the user called `user1`, and the group called `group1`. `ROLE_TEST` is given admin authorization to assign roles.

```
db2 "call rdsadmin.grant_role(
    ?,
    'TESTDB',
    'ROLE_TEST',
    'ROLE role1, USER user1, GROUP group1',
    'Y')"
```

Example 2: Assigning role to PUBLIC and not granting authorization

The following example assigns a role called ROLE_TEST for database TESTDB to PUBLIC. ROLE_TEST isn't given admin authorization to assign roles.

```
db2 "call rdsadmin.grant_role(
    ?,
    'TESTDB',
    'ROLE_TEST',
    'PUBLIC')"
```

rdsadmin.revoke_role

Revokes a role from a role, user, or group.

Syntax

```
db2 "call rdsadmin.revoke_role(
    ?,
    'database_name',
    'role_name',
    'grantee')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs the unique identifier for the task. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database the command will run on. The data type is varchar.

role_name

The name of the role that you want to revoke. The data type is varchar.

grantee

The role, user, or group to lose authorization. The data type is varchar. Valid values: ROLE, USER, GROUP, PUBLIC.

Format must be value followed by name. Separate multiple values and names with commas. Example: 'USER *user1*, *user2*, GROUP *group1*, *group2*'. Replace the names with your own information.

Usage notes

For information about checking the status of revoking a role, see rdsadmin.get_task_status.

Examples

Example 1: Revoking role from role, user, and group

The following example revokes a role called ROLE_TEST for database TESTDB from the role called role1, the user called user1, and the group called group1.

```
db2 "call rdsadmin.revoke_role(
    ?,
    'TESTDB',
    'ROLE_TEST',
    'ROLE role1, USER user1, GROUP group1')"
```

Example 2: Revoking role from PUBLIC

The following example revokes a role called ROLE_TEST for database TESTDB from PUBLIC.

```
db2 "call rdsadmin.revoke_role(
    ?,
    'TESTDB',
    'ROLE_TEST',
    'PUBLIC')"
```

rdsadmin.drop_role

Drops a role.

Syntax

```
db2 "call rdsadmin.drop_role(
    ?,
    'database_name',
    'role_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs the unique identifier for the task. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database the command will run on. The data type is `varchar`.

role_name

The name of the role that you want to drop. The data type is `varchar`.

Usage notes

For information about checking the status of dropping a role, see rdsadmin.get_task_status.

Examples

The following example drops a role called `ROLE_TEST` for database `TESTDB`.

```
db2 "call rdsadmin.drop_role(
    ?,
    'TESTDB',
    'ROLE_TEST')"
```

rdsadmin.add_user

Adds a user to an authorization list.

Syntax

```
db2 "call rdsadmin.add_user(
    'username',
    'password',
    'group_name,group_name')"
```

Parameters

The following parameters are required:

username

A user's username. The data type is `varchar`.

password

A user's password. The data type is `varchar`.

The following parameter is optional:

group_name

The name of a group that you want to add the user to. The data type is `varchar`. The default is an empty string or null.

Usage notes

You can add a user to one or more groups by separating the group names with commas.

You can create a group when you create a new user, or when you add a group to an existing user. You can't create a group by itself.

> **Note**
>
> The maximum number of users that you can add by calling `rdsadmin.add_user` is 5,000.

For information about checking the status of adding a user, see rdsadmin.get_task_status.

Examples

The following example creates a user called `jorge_souza` and assigns the user to the groups called `sales` and `inside_sales`.

```
db2 "call rdsadmin.add_user(
    'jorge_souza',
    '*******',
    'sales,inside_sales')"
```

rdsadmin.change_password

Changes a user's password.

Syntax

```
db2 "call rdsadmin.change_password(
    'username',
    'new_password')"
```

Parameters

The following parameters are required:

username

A user's username. The data type is `varchar`.

new_password

A new password for the user. The data type is `varchar`.

Usage notes

For information about checking the status of changing a password, see rdsadmin.get_task_status.

Examples

The following example changes the password for `jorge_souza`.

```
db2 "call rdsadmin.change_password(
    'jorge_souza',
    '*******')"
```

rdsadmin.list_users

Lists users on an authorization list.

Syntax

```
db2 "call rdsadmin.list_users()"
```

Usage notes

For information about checking the status of listing users, see rdsadmin.get_task_status.

rdsadmin.remove_user

Removes user from authorization list.

Syntax

```
db2 "call rdsadmin.remove_user('username')"
```

Parameters

The following parameter is required:

username

A user's username. The data type is `varchar`.

Usage notes

For information about checking the status of removing a user, see rdsadmin.get_task_status.

Examples

The following example removes `jorge_souza` from being able to access databases in RDS for Db2 DB instances.

```
db2 "call rdsadmin.remove_user('jorge_souza')"
```

rdsadmin.add_groups

Adds groups to a user.

Syntax

```
db2 "call rdsadmin.add_groups(
    'username',
    'group_name,group_name')"
```

Parameters

The following parameters are required:

username

A user's username. The data type is `varchar`.

group_name

The name of a group that you want to add the user to. The data type is `varchar`. The default is an empty string.

Usage notes

You can add one or more groups to a user by separating the group names with commas. For information about checking the status of adding groups, see rdsadmin.get_task_status.

Examples

The following example adds the `direct_sales` and `b2b_sales` groups to user `jorge_souza`.

```
db2 "call rdsadmin.add_groups(
    'jorge_souza',
    'direct_sales,b2b_sales')"
```

rdsadmin.remove_groups

Removes groups from a user.

Syntax

```
db2 "call rdsadmin.remove_groups(
    'username',
    'group_name,group_name')"
```

Parameters

The following parameters are required:

username

A user's username. The data type is `varchar`.

group_name

The name of a group that you want to remove the user from. The data type is `varchar`.

Usage notes

You can remove one or more groups from a user by separating the group names with commas.

For information about checking the status of removing groups, see rdsadmin.get_task_status.

Examples

The following example removes the `direct_sales` and `b2b_sales` groups from user `jorge_souza`.

```
db2 "call rdsadmin.remove_groups(
    'jorge_souza',
    'direct_sales,b2b_sales')"
```

rdsadmin.dbadm_grant

Grants `DBADM`, `ACCESSCTRL`, or `DATAACCESS` authorization to a role, user, or group.

Syntax

```
db2 "call rdsadmin.dbadm_grant(
    ?,
```

```
    'database_name',
    'authorization',
    'grantee')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs the unique identifier for the task. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database the command will run on. The data type is `varchar`.

authorization

The type of authorization to grant. The data type is `varchar`. Valid values: `DBADM`, `ACCESSCTRL`, `DATAACCESS`.

Separate multiple types with commas.

grantee

The role, user, or group to receive authorization. The data type is `varchar`. Valid values: `ROLE`, `USER`, `GROUP`.

Format must be value followed by name. Separate multiple values and names with commas. Example: 'USER *user1*, *user2*, GROUP *group1*, *group2*'. Replace the names with your own information.

Usage notes

The role to receive access must exist.

For information about checking the status of granting database admin access, see rdsadmin.get_task_status.

Examples

Example 1: Granting database admin access to role

The following example grants database admin access to the database named TESTDB for the role ROLE_DBA.

```
db2 "call rdsadmin.dbadm_grant(
    ?,
    'TESTDB',
    'DBADM',
    'ROLE ROLE_DBA')"
```

Example 2: Granting database admin access to user and group

The following example grants database admin access to the database named TESTDB for user1 and group1.

```
db2 "call rdsadmin.dbadm_grant(
    ?,
    'TESTDB',
    'DBADM',
    'USER user1, GROUP group1')"
```

Example 3: Granting database admin access to multiple users and groups

The following example grants database admin access to the database named TESTDB for user1, user2, group1, and group2.

```
db2 "call rdsadmin.dbadm_grant(
    ?,
    'TESTDB',
    'DBADM',
    'USER user1, user2, GROUP group1, group2')"
```

rdsadmin.dbadm_revoke

Revokes DBADM, ACCESSCTRL, or DATAACCESS authorization from a role, user, or group.

Syntax

```
db2 "call rdsadmin.dbadm_revoke(
```

```
    ?,
    'database_name',
    'authorization',
    'grantee')"
```

Parameters

The following output parameter is required:

?

The unique identifier for the task. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database the command will run on. The data type is varchar.

authorization

The type of authorization to revoke. The data type is varchar. Valid values: DBADM, ACCESSCTRL, DATAACCESS.

Separate multiple types with commas.

grantee

The role, user, or group to revoke authorization from. The data type is varchar. Valid values: ROLE, USER, GROUP.

Format must be value followed by name. Separate multiple values and names with commas. Example: 'USER *user1*, *user2*, GROUP *group1*, *group2*'. Replace the names with your own information.

Usage notes

For information about checking the status of revoking database admin access, see rdsadmin.get_task_status.

Examples

Example 1: Revoking database admin access from role

The following example revokes database admin access to the database named TESTDB for the role ROLE_DBA.

```
db2 "call rdsadmin.dbadm_revoke(
    ?,
    'TESTDB',
    'DBADM',
    'ROLE ROLE_DBA')"
```

Example 2: Revoking database admin access from user and group

The following example revokes database admin access to the database named TESTDB for user1 and group1.

```
db2 "call rdsadmin.dbadm_revoke(
    ?,
    'TESTDB',
    'DBADM',
    'USER user1, GROUP group1')"
```

Example 3: Revoking database admin access from multiple users and groups

The following example revokes database admin access to the database named TESTDB for user1, user2, group1, and group2.

```
db2 "call rdsadmin.dbadm_revoke(
    ?,
    'TESTDB',
    'DBADM',
    'USER user1, user2, GROUP group1, group2')"
```

rdsadmin.set_sid_group_mapping

Creates a mapping between a security ID (SID) and the corresponding Active Directory group.

Syntax

```
db2 "call rdsadmin.set_sid_group_mapping(
    ?,
    'SID',
```

```
    'group_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameters are required:

SID

The security ID (SID). The data type is `varchar`.

group_name

The name of the Active Directory group to map to the SID. The data type is `varchar`.

Usage notes

Use this stored procedure to enable Kerberos authentication with Active Directory groups. If the `SID` or `group_name` already exists in the mapping, this stored procedure fails.

For information about how to find the SID for a group, see Step 8: Retrieve the Active Directory group SID in PowerShell.

For information about checking the status of creating a mapping, see rdsadmin.get_task_status.

Examples

The following example maps a SID to a group called `my_group`.

```
db2 "call rdsadmin.set_sid_group_mapping(
    ?,
    'S-1-5-21-9146495592-531070549-834388463-513',
    'my_group')"
```

rdsadmin.list_sid_group_mapping

Lists all security ID (SID) and Active Directory group mappings configured on the DB instance.

Syntax

```
db2 "call rdsadmin.list_sid_group_mapping()"
```

Usage notes

For information about checking the status of listing mappings, see rdsadmin.get_task_status.

rdsadmin.remove_sid_group_mapping

Removes a security ID (SID) and its corresponding Active Directory group mapping from a DB instance.

Syntax

```
db2 "call rdsadmin.remove_sid_group_mapping(
    ?,
    'SID')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

SID

The security ID (SID). The data type is `varchar`.

Usage notes

For information about how to find the SID for a group, see Step 8: Retrieve the Active Directory group SID in PowerShell.

For information about checking the status of removing mappings, see rdsadmin.get_task_status.

Examples

The following example removes a SID mapping from the group it was mapped to.

```
db2 "call rdsadmin.remove_sid_group_mapping(
    ?,
    'S-1-5-21-9146495592-531070549-834388463-513')"
```

Stored procedures for audit policies for RDS for Db2

The built-in stored procedures described in this topic manage audit policies for Amazon RDS for Db2 databases that use audit logging. For more information, see the section called “Db2 audit logging”. To run these procedures, the master user must first connect to the `rdsadmin` database.

Refer to the following built-in stored procedures for information about their syntax, parameters, usage notes, and examples.

Stored procedures

- rdsadmin.configure_db_audit
- rdsadmin.disable_db_audit

rdsadmin.configure_db_audit

Configures the audit policy for the RDS for Db2 database specified by *db_name*. If the policy you're configuring doesn't exist, calling this stored procedure creates it. If this policy does exist, calling this stored procedure modifies it with the parameter values that you provide.

Syntax

```
db2 "call rdsadmin.configure_db_audit(
    'db_name',
    'category',
    'category_setting',
    '?')"
```

Parameters

The following parameters are required.

db_name

The DB name of the RDS for Db2 database to configure the audit policy for. The data type is `varchar`.

category

The name of the category to configure this audit policy for. The data type is `varchar`. The following are valid values for this parameter:

- ALL – With ALL, Amazon RDS doesn't include the CONTEXT, EXECUTE, or ERROR categories.
- AUDIT
- CHECKING
- CONTEXT
- ERROR
- EXECUTE – You can configure this category with data or without data. With data means to also log input data values provided for any host variables and parameter markers. The default is without data. For more information, see the description of the *category_setting* parameter and the the section called "Examples".
- OBJMAINT
- SECMAINT
- SYSADMIN
- VALIDATE

For more information about these categories, see the IBM Db2 documentation.

category_setting

The setting for the specified audit category. The data type is varchar.

The following table shows the valid category setting values for each category.

Category	Valid category settings
ALL	BOTH|FAILURE|SUCCESS|NONE
AUDIT	
CHECKING	
CONTEXT	
OBJMAINT	
SECMAINT	
SYSADMIN	

Category	Valid category settings
VALIDATE	
ERROR	AUDIT|NORMAL . The default is NORMAL.
EXECUTE	BOTH,WITH|BOTH,WITHOUT|FAILURE,WITH|FAILURE,WITHOUT|SUCCESS,WITH|SUCCESS,WITHOUT|NONE

Usage notes

Before you call `rdsadmin.configure_db_audit`, make sure the RDS for Db2 DB instance with the database you're configuring the audit policy for is associated with an option group that has the `DB2_AUDIT` option. For more information, see the section called “Setting up Db2 audit logging”.

After you configure the audit policy, you can check the status of the audit configuration for the database by following the steps in Check the audit configuration.

Specifying `ALL` for the `category` parameter doesn't include the `CONTEXT`, `EXECUTE`, or `ERROR` categories. To add these categories to your audit policy, call `rdsadmin.configure_db_audit` separately with each category that you want to add. For more information, see the section called “Examples”.

Examples

The following examples create or modify the audit policy for a database named `TESTDB`. In examples 1 through 5, if the `ERROR` category wasn't previously configured, this category is set to `NORMAL` (the default). To change that setting to `AUDIT`, follow Example 6: Specifying the ERROR category.

Example 1: Specifying the ALL category

```
db2 "call rdsadmin.configure_db_audit('TESTDB', 'ALL', 'BOTH', ?)"
```

In the example, the call configures the `AUDIT`, `CHECKING`, `OBJMAINT`, `SECMAINT`, `SYSADMIN`, and `VALIDATE` categories in the audit policy. Specifying `BOTH` means that both successful and failing events will be audited for each of these categories.

Example 2: Specifying the EXECUTE category with data

```
db2 "call rdsadmin.configure_db_audit('TESTDB', 'EXECUTE', 'SUCCESS,WITH', ?)"
```

In the example, the call configures the EXECUTE category in the audit policy. Specifying SUCCESS,WITH means that logs for this category will include only successful events, and will include input data values provided for host variables and parameter markers.

Example 3: Specifying the EXECUTE category without data

```
db2 "call rdsadmin.configure_db_audit('TESTDB', 'EXECUTE', 'FAILURE,WITHOUT', ?)"
```

In the example, the call configures the EXECUTE category in the audit policy. Specifying FAILURE,WITHOUT means that logs for this category will include only failing events, and won't include input data values provided for host variables and parameter markers.

Example 4: Specifying the EXECUTE category without status events

```
db2 "call rdsadmin.configure_db_audit('TESTDB', 'EXECUTE', 'NONE', ?)"
```

In the example, the call configures the EXECUTE category in the audit policy. Specifying NONE means that no events in this category will be audited.

Example 5: Specifying the OBJMAINT category

```
db2 "call rdsadmin.configure_db_audit('TESTDB', 'OBJMAINT', 'NONE', ?)"
```

In the example, the call configures the OBJMAINT category in the audit policy. Specifying NONE means that no events in this category will be audited.

Example 6: Specifying the ERROR category

```
db2 "call rdsadmin.configure_db_audit('TESTDB', 'ERROR', 'AUDIT', ?)"
```

In the example, the call configures the ERROR category in the audit policy. Specifying AUDIT means that all errors, including errors occurring within audit logging itself, are captured in the logs. The default error type is NORMAL. With NORMAL, errors generated by the audit are ignored and only the SQLCODEs for errors associated with the operation being performed are captured.

rdsadmin.disable_db_audit

Stops audit logging for the RDS for Db2 database specified by *db_name* and removes the audit policy configured for it.

> **Note**
>
> This stored procedure only removes audit policies that were configured by calling the section called "rdsadmin.configure_db_audit".

Syntax

```
db2 "call rdsadmin.disable_db_audit('db_name', ?)"
```

Parameters

The following parameters are required.

db_name

The DB name of the RDS for Db2 database to disable audit logging for. The data type is `varchar`.

Usage notes

Calling `rdsadmin.disable_db_audit` doesn't disable audit logging for the RDS for Db2 DB instance. To disable audit logging at the DB instance level, remove the option group from the DB instance. For more information, see Disabling Db2 audit logging.

Examples

The following example disables audit logging for a database named `TESTDB`.

```
db2 "call rdsadmin.disable_db_audit('TESTDB', ?)"
```

Stored procedures for buffer pools for RDS for Db2

The built-in stored procedures described in this topic manage buffer pools for Amazon RDS for Db2 databases. To run these procedures, the master user must first connect to the `rdsadmin` database.

These stored procedures are used in a variety of tasks. This list isn't exhaustive.

- Common tasks for buffer pools
- Generating performance reports
- Copying database metadata with db2look
- Creating a repository database for IBM Db2 Data Management Console

Refer to the following built-in stored procedures for information about their syntax, parameters, usage notes, and examples.

Stored procedures

- rdsadmin.create_bufferpool
- rdsadmin.alter_bufferpool
- rdsadmin.drop_bufferpool

rdsadmin.create_bufferpool

Creates a buffer pool.

Syntax

```
db2 "call rdsadmin.create_bufferpool(
    'database_name',
    'buffer_pool_name',
    buffer_pool_size,
    'immediate',
    'automatic',
    page_size,
    number_block_pages,
    block_size)"
```

Parameters

The following parameters are required:

database_name

The name of the database to run the command on. The data type is varchar.

buffer_pool_name

The name of the buffer pool to create. The data type is varchar.

The following parameters are optional:

buffer_pool_size

The size of the buffer pool in number of pages. The data type is integer. The default is -1.

immediate

Specifies whether the command runs immediately. The data type is char. The default is Y.

automatic

Specifies whether to set the buffer pool to automatic. The data type is char. The default is Y.

page_size

The page size of the buffer pool. The data type is integer. Valid values: 4096, 8192, 16384, 32768. The default is 8192.

number_block_pages

The number of block pages in the buffer pools. The data type is integer. The default is 0.

block_size

The block size for the block pages. The data type is integer. Valid values: 2 to 256. The default is 32.

Usage notes

For information about checking the status of creating a buffer pool, see rdsadmin.get_task_status.

Examples

Example 1: Creating buffer pool with default parameters

The following example creates a buffer pool called BP8 for a database called TESTDB with default parameters, so the buffer pool uses an 8 KB page size.

```
db2 "call rdsadmin.create_bufferpool(
    'TESTDB',
    'BP8')"
```

Example 2: Creating buffer pool to run immediately with automatic allocation

The following example creates a buffer pool called BP16 for a database called TESTDB that uses a 16 KB page size with an initial page count of 1,000 and is set to automatic. Db2 runs the command immediately. If you use an initial page count of -1, then Db2 will use automatic allocation of pages.

```
db2 "call rdsadmin.create_bufferpool(
    'TESTDB',
    'BP16',
    1000,
    'Y',
    'Y',
    16384)"
```

Example 3: Creating buffer pool to run immediately using block pages

The following example creates a buffer pool called BP16 for a database called TESTDB. This buffer pool has a 16 KB page size with an initial page count of 10,000. Db2 runs the command immediately using 500 block pages with a block size of 512.

```
db2 "call rdsadmin.create_bufferpool(
    'TESTDB',
    'BP16',
    10000,
    'Y',
    'Y',
    16384,
    500,
    512)"
```

rdsadmin.alter_bufferpool

Alters a buffer pool.

Syntax

```
db2 "call rdsadmin.alter_bufferpool(
    'database_name',
    'buffer_pool_name',
    buffer_pool_size,
    'immediate',
    'automatic',
    change_number_blocks,
    number_block_pages,
    block_size)"
```

Parameters

The following parameters are required:

database_name

The name of the database to run the command on. The data type is `varchar`.

buffer_pool_name

The name of the buffer pool to alter. The data type is `varchar`.

buffer_pool_size

The size of the buffer pool in number of pages. The data type is `integer`.

The following parameters are optional:

immediate

Specifies whether the command runs immediately. The data type is `char`. The default is `Y`.

automatic

Specifies whether to set the buffer pool to automatic. The data type is `char`. The default is `N`.

change_number_blocks

Specifies whether there is a change to the number of block pages in the buffer pool. The data type is `char`. The default is `N`.

number_block_pages

The number of block pages in the buffer pools. The data type is `integer`. The default is `0`.

block_size

The block size for the block pages. The data type is `integer`. Valid values: 2 to 256. The default is 32.

Usage notes

For information about checking the status of altering a buffer pool, see rdsadmin.get_task_status.

Examples

The following example alters a buffer pool called BP16 for a database called TESTDB to non-automatic, and changes the size to 10,000 pages. Db2 runs this command immediately.

```
db2 "call rdsadmin.alter_bufferpool(
    'TESTDB',
    'BP16',
    10000,
    'Y',
    'N')"
```

rdsadmin.drop_bufferpool

Drops a buffer pool.

Syntax

```
db2 "call rdsadmin.drop_bufferpool(
    'database_name',
    'buffer_pool_name'"
```

Parameters

The following parameters are required:

database_name

The name of the database that the buffer pool belongs to. The data type is `varchar`.

buffer_pool_name

The name of the buffer pool to drop. The data type is `varchar`.

Usage notes

For information about checking the status of dropping a buffer pool, see rdsadmin.get_task_status.

Examples

The following example drops a buffer pool called BP16 for a database called TESTDB.

```
db2 "call rdsadmin.drop_bufferpool(
    'TESTDB',
    'BP16')"
```

Stored procedures for databases for RDS for Db2

The built-in stored procedures described in this topic manage databases for Amazon RDS for Db2. To run these procedures, the master user must first connect to the `rdsadmin` database.

These stored procedures are used in a variety of tasks. This list isn't exhaustive.

- Common tasks for databases
- Creating databases with EBCDIC collation
- Collecting information about databases
- Modifying database configuration parameters
- Migrating from Linux to Linux
- Migrating from Linux to Linux with near-zero downtime

Refer to the following built-in stored procedures for information about their syntax, parameters, usage notes, and examples.

Stored procedures

- rdsadmin.create_database
- rdsadmin.deactivate_database
- rdsadmin.activate_database
- rdsadmin.reactivate_database
- rdsadmin.drop_database
- rdsadmin.update_db_param
- rdsadmin.set_configuration
- rdsadmin.show_configuration
- rdsadmin.backup_database
- rdsadmin.restore_database
- rdsadmin.rollforward_database
- rdsadmin.rollforward_status
- rdsadmin.complete_rollforward
- rdsadmin.db2pd_command
- rdsadmin.force_application

- rdsadmin.set_archive_log_retention
- rdsadmin.show_archive_log_retention
- rdsadmin.list_archive_log_information
- rdsadmin.enable_archive_log_copy
- rdsadmin.disable_archive_log_copy
- rdsadmin.fgac_command
- rdsadmin.db2support_command

rdsadmin.create_database

Creates a database.

Syntax

```
db2 "call rdsadmin.create_database(
    'database_name',
    'database_page_size',
    'database_code_set',
    'database_territory',
    'database_collation',
    'database_autoconfigure_str',
    'database_non-restrictive')"
```

Parameters

> **Note**
>
> This stored procedure doesn't validate the combination of required parameters. When you call rdsadmin.get_task_status, the user-defined function could return an error because of a combination of `database_codeset`, `database_territory`, and `database_collation` that is not valid. For more information, see Choosing the code page, territory, and collation for your database in the IBM Db2 documentation.

The following parameter is required:

database_name

The name of the database to create. The data type is `varchar`.

The following parameters are optional:

database_page_size

The default page size of the database. Valid values: 4096, 8192, 16384, 32768. The data type is integer. The default is 8192.

> **Important**
>
> Amazon RDS supports write atomicity for 4 KiB, 8 KiB, and 16 KiB pages. In contrast, 32 KiB pages risk *torn writes*, or partial data being written to the disk. If you use 32 KiB pages, we recommend that you enable point-in-time recovery and automated backups. Otherwise, you run the risk of being unable to recover from torn pages. For more information, see the section called "Introduction to backups" and the section called "Point-in-time recovery".

database_code_set

The code set for the database. The data type is varchar. The default is UTF-8.

database_territory

The two-letter country code for the database. The data type is varchar. The default is US.

database_collation

The collation sequence that determines how character strings stored in the database are sorted and compared. The data type is varchar.

Valid values:

- COMPATIBILITY – An IBM Db2 Version 2 collation sequence.
- EBCDIC_819_037 – ISO Latin code page, collation; CCSID 037 (EBCDIC US English).
- EBCDIC_819_500 – ISO Latin code page, collation; CCSID 500 (EBCDIC International).
- EBCDIC_850_037 – ASCII Latin code page, collation; CCSID 037 (EBCDIC US English).
- EBCDIC_850_500 – ASCII Latin code page, collation; CCSID 500 (EBCDIC International).
- EBCDIC_932_5026 – ASCII Japanese code page, collation; CCSID 5026 (EBCDIC US English).
- EBCDIC_932_5035 – ASCII Japanese code page, collation; CCSID 5035 (EBCDIC International).

- EBCDIC_1252_037 – Windows Latin code page, collation; CCSID 037 (EBCDIC US English).
- EBCDIC_1252_500 – Windows Latin code page, collation; CCSID 500 (EBCDIC International).
- IDENTITY – Default collation. Strings are compared byte for byte.
- IDENTITY_16BIT – The Compatibility Encoding Scheme for UTF-16: 8-bit (CESU-8) collation sequence. For more information, see Unicode Technical Report #26 on the Unicode Consortium website.
- NLSCHAR – Only for use with the Thai code page (CP874).
- SYSTEM – If you use SYSTEM, the database uses the collation sequence automatically for database_codeset and database_territory.

The default is IDENTITY.

Additionally, RDS for Db2 supports the following groups of collations: language-aware-collation and locale-sensitive-collation. For more information, see Choosing a collation for a Unicode database in the IBM Db2 documentation.

database_autoconfigure_str

The AUTOCONFIGURE command syntax, for example, 'AUTOCONFIGURE APPLY DB'. The data type is varchar. The default is an empty string or null.

For more information, see AUTOCONFIGURE command in the IBM Db2 documentation.

database_non-restrictive

The granting of default authorities and privileges within the database. The data type is varchar. The default is N.

Valid values:

- N – The created database is restrictive and doesn't grant authorities or privileges.
- Y – The created database is non-restrictive and grants a set of permissions to the special group PUBLIC. For more information, see Default privileges granted on creating a database in the IBM Db2 documentation.

Usage notes

If you plan on modifying the db2_compatibility_vector parameter, modify the parameter before creating a database. For more information, see Setting the db2_compatibility_vector parameter.

Special considerations:

- The `CREATE DATABASE` command sent to the Db2 instance uses the `RESTRICTIVE` option.
- RDS for Db2 only uses `AUTOMATIC STORAGE` tablespaces.
- RDS for Db2 uses the default values for `NUMSEGS` and `DFT_EXTENT_SZ`.
- RDS for Db2 uses storage encryption and doesn't support database encryption.

For more information about these considerations, see CREATE DATABASE command in the IBM Db2 documentation.

Before calling `rdsadmin.create_database`, you must connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your RDS for Db2 DB instance information:

```
db2 connect to rdsadmin user master_username using master_password
```

For information about checking the status of creating a database, see rdsadmin.get_task_status.

For error messages returned when calling `rdsadmin.create_database`, see the section called "Stored procedure errors".

Examples

The following example creates a database called `TESTJP` with a correct combination of the *`database_code_set`*, *`database_territory`*, and *`database_collation`* parameters for Japan:

```
db2 "call rdsadmin.create_database('TESTJP', 4096, 'IBM-437', 'JP', 'SYSTEM')"
```

rdsadmin.deactivate_database

Deactivates a database.

Syntax

```
db2 "call rdsadmin.deactivate_database(
    ?,
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

database_name

The name of the database to deactivate. The data type is `varchar`.

Usage notes

You can deactivate databases to conserve memory resources or to make other database configuration changes. To bring deactivated databases back online, call the rdsadmin.activate_database stored procedure.

You can't deactivate a database on a source DB instance during replication by calling the `rdsadmin.deactivate_database` stored procedure.

For information about checking the status of deactivating a database, see rdsadmin.get_task_status.

For error messages returned when calling `rdsadmin.deactivate_database`, see the section called "Stored procedure errors".

Examples

The following example deactivates a database called `TESTDB`.

```
db2 "call rdsadmin.deactivate_database(?, 'TESTDB')"
```

rdsadmin.activate_database

Activates a database.

For information about the differences between rdsadmin.reactivate_database and `rdsadmin.activate_database`, see Usage notes.

Syntax

```
db2 "call rdsadmin.activate_database(
    ?,
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

database_name

The name of the database to activate. The data type is `varchar`.

Usage notes

All databases are activated by default when they are created. If you deactivate a database on a standalone DB instance to conserve memory resources or to make other database configuration changes, call the `rdsadmin.activate_database` stored procedure to activate the database again.

This stored procedure only activates a database that is on a standalone DB instance and that was deactivated by calling the rdsadmin.deactivate_database stored procedure. To activate a database on a replica source DB instance, you must call the rdsadmin.reactivate_database stored procedure.

For information about checking the status of activating a database, see rdsadmin.get_task_status.

For error messages returned when calling `rdsadmin.activate_database`, see the section called "Stored procedure errors".

Examples

The following example activates a database called `TESTDB`.

```
db2 "call rdsadmin.activate_database(?, 'TESTDB')"
```

rdsadmin.reactivate_database

Reactivates a database.

For information about differences between rdsadmin.activate_database and `rdsadmin.reactivate_database`, see Usage notes.

Syntax

```
db2 "call rdsadmin.reactivate_database(
    ?,
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

database_name

The name of the database to reactivate. The data type is `varchar`.

Usage notes

When you call the `rdsadmin.reactivate_database` stored procedure, the stored procedure first deactivates the database by calling the rdsadmin.deactivate_database stored procedure, and then activates the database by calling the rdsadmin.activate_database stored procedure.

After you make changes to database configurations, you might need to reactivate a database on an RDS for Db2 DB instance. To determine if you need to reactivate a database, connect to the database and run `db2 get db cfg show detail`.

For a database on a standalone DB instance, you can use the `rdsadmin.reactivate_database` store procedure to reactivate the database. Or, if you already called the rdsadmin.deactivate_database stored procedure, you can call the rdsadmin.activate_database stored procedure instead.

For a database on a replica source DB instance, you must use the `rdsadmin.reactivate_database` stored procedure to reactivate the database.

For information about checking the status of reactivating a database, see rdsadmin.get_task_status.

For error messages returned when calling `rdsadmin.reactivate_database`, see the section called "Stored procedure errors".

Examples

The following example reactivates a database called `TESTDB`.

```
db2 "call rdsadmin.reactivate_database(?, 'TESTDB')"
```

rdsadmin.drop_database

Drops a database.

Syntax

```
db2 "call rdsadmin.drop_database('database_name')"
```

Parameters

The following parameter is required:

database_name

The name of the database to drop. The data type is `varchar`.

Usage notes

You can drop a database by calling `rdsadmin.drop_database` only if the following conditions are met:

- You didn't specify the name of the database when you created your RDS for Db2 DB instance by using either the Amazon RDS console or the AWS CLI. For more information, see Creating a DB instance.
- You created the database by calling the the section called "rdsadmin.create_database" stored procedure.

- You restored the database from an offline or backed-up image by calling the the section called "rdsadmin.restore_database" stored procedure.

Before calling `rdsadmin.drop_database`, you must connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your RDS for Db2 DB instance information:

```
db2 connect to rdsadmin user master_username using master_password
```

For information about checking the status of dropping a database, see rdsadmin.get_task_status.

For error messages returned when calling `rdsadmin.drop_database`, see the section called "Stored procedure errors".

Examples

The following example drops a database called TESTDB:

```
db2 "call rdsadmin.drop_database('TESTDB')"
```

rdsadmin.update_db_param

Updates database parameters.

Syntax

```
db2 "call rdsadmin.update_db_param(
    'database_name',
    'parameter_to_modify',
    'changed_value',
    'restart_database')"
```

Parameters

The following parameters are required:

`database_name`

The name of the database to run the task for. The data type is `varchar`.

parameter_to_modify

The name of the parameter to modify. The data type is `varchar`. For more information, see Amazon RDS for Db2 parameters.

changed_value

The value to change the parameter value to. The data type is `varchar`.

The following parameter is optional:

restart_database

Specifies whether RDS restarts the database if a restart is necessary. The data type is `varchar`. To modify `logprimary` and `logfilsiz`, set this parameter to `'YES'`.

Usage notes

For information about checking the status of updating database parameters, see rdsadmin.get_task_status.

For error messages returned when calling `rdsadmin.update_db_param`, see the section called "Stored procedure errors".

Examples

Example 1: Updating a parameter

The following example updates the `archretrydelay` parameter to `100` for a database called `TESTDB`:

```
db2 "call rdsadmin.update_db_param(
    'TESTDB',
    'archretrydelay',
    '100')"
```

Example 2: Deferring validation of objects

The following example defers the validation of created objects on a database called `TESTDB` to avoid dependency checking:

```
db2 "call rdsadmin.update_db_param(
    'TESTDB',
    'auto_reval',
    'deferred_force')"
```

rdsadmin.set_configuration

Configures specific settings for the database.

Syntax

```
db2 "call rdsadmin.set_configuration(
    'name',
    'value')"
```

Parameters

The following parameters are required:

name

The name of the configuration setting. The data type is `varchar`.

value

The value for the configuration setting. The data type is `varchar`.

Usage notes

The following table shows the configuration settings that you can control with `rdsadmin.set_configuration`.

Name	Description
RESTORE_DATABASE_NUM_BUFFERS	The number of buffers to create during a restore operation. This value must be less than the total memory size of the DB instance class. If this setting isn't configured, Db2 determines the value to use during the restore operation. For more information, see RESTORE DATABASE command in the IBM Db2 documentation.

Name	Description
RESTORE_DATABASE_PARALLELISM	The number of buffer manipulators to create during a restore operation. This value must be less than double the number of vCPUs for the DB instance. If this setting isn't configured, Db2 determines the value to use during the restore operation. For more information, see RESTORE DATABASE command in the IBM Db2 documentation.
RESTORE_DATABASE_NUM_MULTI_PATHS	The number of paths (or I/O streams) to use during a restore from Amazon S3 operation. To use this configuration setting, you must have multiple backup files. This value can improve performance when restoring databases with large volumes of data because it restores multiple database backup files in parallel. We recommend that you set this value to match the number of your database backup files. For more information, see BACKUP DATABASE command in the IBM Db2 documentation.
USE_STREAMING_RESTORE	Specifies whether to stream backup data directly during restoration rather than first downloading the entire backup to your RDS for Db2 DB instance and then extracting it. Setting USE_STREAMING_RESTORE to TRUE significantly reduces storage requirements and can improve restore performance. This setting requires IBM Db2 engine version 11.5.9.0.sb00063198.r1 or higher, and Amazon S3 connectivity through your database's elastic network interface (ENI). For more information, see Remote storage in the IBM Db2 documentation.

Examples

Example 1: Specifying number of buffers to create

The following example sets the RESTORE_DATABASE_NUM_BUFFERS configuration to 150.

```
db2 "call rdsadmin.set_configuration(
    'RESTORE_DATABASE_NUM_BUFFERS',
```

```
    '150')"
```

Example 2: Specifying number of buffer manipulators to create

The following example sets the RESTORE_DATABASE_PARALLELISM configuration to 8.

```
db2 "call rdsadmin.set_configuration(
    'RESTORE_DATABASE_PARALLELISM',
    '8')"
```

Example 3: Specifying number of paths or I/O streams to use during restore

The following example sets the RESTORE_DATABASE_NUM_MULTI_PATHS configuration to 5.

```
db2 "call rdsadmin.set_configuration(
    'RESTORE_DATABASE_NUM_MULTI_PATHS',
    '5')"
```

Example 4: Setting restoration to stream backup data

The following example sets the USE_STREAMING_RESTORE configuration to TRUE.

```
db2 "call rdsadmin.set_configuration(
    'USE_STREAMING_RESTORE',
    'TRUE')"
```

rdsadmin.show_configuration

Returns the current settings that you can set by using the stored procedure rdsadmin.set_configuration.

Syntax

```
db2 "call rdsadmin.show_configuration(
    'name')"
```

Parameters

The following parameter is optional:

name

The name of the configuration setting to return information about. The data type is `varchar`.

The following configuration names are valid:

- `RESTORE_DATABASE_NUM_BUFFERS` – The number of buffers to create during a restore operation.
- `RESTORE_DATABASE_PARALLELISM` – The number of buffer manipulators to create during a restore operation.
- `RESTORE_DATABASE_NUM_MULTI_PATHS` – The number of paths (or I/O streams) to use during a restore from Amazon S3 operation.
- `USE_STREAMING_RESTORE` – Specifies whether to stream backup data directly during restoration rather than first downloading the entire backup data to your RDS for Db2 DB instance and then extracting it.

Usage notes

If you don't specify the name of a configuration setting, `rdsadmin.show_configuration` returns information for all configuration settings that you can set by using the stored procedure `rdsadmin.set_configuration`.

Examples

The following example returns information about the current `RESTORE_DATABASE_PARALLELISM` configuration.

```
db2 "call rdsadmin.show_configuration(
    'RESTORE_DATABASE_PARALLELISM')"
```

rdsadmin.backup_database

Backs up a database from an RDS for Db2 DB instance to an Amazon S3 bucket.

Syntax

```
db2 "call rdsadmin.backup_database(
    ?,
    'database_name',
```

```
    's3_bucket_name',
    's3_prefix',
    'backup_type',
    'compression_option',
    'util_impact_priority',
    'num_files',
    'parallelism',
    'num_buffers')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the target database on an RDS for Db2 DB instance to back up. The data type is `varchar`.

The database must exist in the DB instance. You can't back up the `rdsadmin` database.

s3_bucket_name

The name of the Amazon S3 bucket where you want your backup to reside. The data type is `varchar`.

The S3 bucket must exist before calling `rdsadmin.backup_database`, be in the same AWS Region as the target database in the RDS for Db2 DB instance that you want to back up, and be accessible through the IAM role attached to the DB instance.

s3_prefix

The prefix of the path to Amazon S3 where RDS for Db2 uploads the backup files. The data type is `varchar`.

The prefix is limited to 1024 characters. It must not include a leading or trailing slash (/). Because of a limitation with IBM streaming to Amazon S3, we recommend that the prefix includes subdirectories.

For better file management, RDS for Db2 creates extra directories after *s3_prefix*. RDS for Db2 uploads all backup files to *s3_prefix/dbi_resource_id/db_name*. If you set *num_files* higher than 1, the *db_name* directory will contain more than one backup file.

The following is an example Amazon S3 location for backup files. In the example, `backups/daily` is the value set for the *s3_prefix* parameter.

```
backups/daily/db-5N7FXOY4GDP7RG2NSH2ZTAI2W4/SAMPLEDB
```

backup_type

The type of backup that determines if the database remains available during backup. The data type is `varchar`.

Valid values:

- `OFFLINE` – The database is unavailable during backup. This type is faster, but it causes downtime.
- `ONLINE` – The database remains available during backup. By default, `ONLINE` is set to `INCLUDE LOGS`.

The following parameters are optional:

compression_option

The type of compression algorithm used that impacts backup time, CPU usage, and storage costs. The data type is `varchar`. The default is `NONE`.

Valid values:

- `NONE` – The largest file size, the least CPU usage, and cheapest storage costs.
- `STANDARD` – Standard Db2 compression. Uses `libdb2compr.so`.
- `ZLIB` – Enhanced Db2 compression. Uses `libdb2zcompr.so`, but is more CPU-intensive and most expensive storage cost.

util_impact_priority

The setting that controls the impact of the backup on the system resources. The data type is `integer`. Valid values: `1`–`100` (from low to high). The default is `50`.

Lower values reduce the impact of the backup on the system resources, but might increase the time it takes to back up the database. Higher values might complete the backup of

the database faster, but could affect other operations. The actual impact depends on the overall system utilization and the `util_impact_lim` setting. You can view and modify the `util_impact_lim` setting in parameter groups. For more information, see Amazon RDS for Db2 parameters.

`num_files`

The number of parallel upload streams to Amazon S3. The data type is `integer`. Valid values: 1–256.

We recommend that you only set this parameter after observing the backup performance at the default that Amazon RDS automatically calculates. Higher values could improve performance for large backups, especially with high-bandwidth connections, but at a certain point, higher values degrade performance. Also, make sure to take into account your available system resources and network capacity.

`parallelism`

The number of tablespaces that the backup utility can read in parallel. The data type is `integer`. Valid values: 1–256.

We recommend that you only set this parameter after observing the backup performance at the default that the Db2 engine automatically calculates as the optimal value. If you set this parameter, Amazon RDS validates against the available processors and won't execute the backup request if processing power is insufficient.

`num_buffers`

The number of buffers to use. The data type is `integer`. Valid values: 1–268435456.

We recommend that you only set this parameter after observing the backup performance at the default that Amazon RDS automatically calculates based on memory. If you set this parameter, Amazon RDS validates against the available memory and won't execute the backup request if available memory is insufficient. If you are backing up to multiple locations (`num_files` is set to more than 1), then a higher number of buffers could improve performance. If you don't set `compression_option` to `NONE`, then you can improve performance by increasing `num_buffers` and `parallelism`.

Usage notes

This stored procedure creates asynchronous backup tasks that stream the backup of your database directly to your Amazon S3 bucket by using the Amazon S3 integration. You can make backups

both from your local server or from an RDS for Db2 DB instance, stream them to Amazon S3, and then restore them wherever you want. For information about restoring a database to an RDS for Db2 DB instance, see rdsadmin.restore_database.

Before calling the stored procedure, review the following considerations:

- You can only back up one database at a time.
- You can't perform a backup and restore together on a DB instance.
- Amazon S3 server-side encryption with AWS KMS (SSE-KMS) isn't supported. Even if the S3 bucket is set to SSE-KMS, the files uploaded to the S3 bucket won't use SSE-KMS encryption.
- To stream the backup files to Amazon S3, you must have already configured the integration. For more information, see Integrating an Amazon RDS for Db2 DB instance with Amazon S3.
- For an RDS for Db2 DB instance to be able to interact with Amazon S3, you must have a VPC and an Amazon S3 gateway endpoint for private subnets to use. For more information, see Step 1: Create a VPC gateway endpoint for Amazon S3 and Step 2: Confirm that your VPC gateway endpoint for Amazon S3 exists.

Before calling `rdsadmin.backup_database`, you must connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your RDS for Db2 DB instance information:

```
db2 connect to rdsadmin user master_username using master_password
```

After you back up your database, be sure to terminate the connection.

```
terminate
```

For information about checking the status of backing up a database, see rdsadmin.get_task_status.

For error messages returned when calling `rdsadmin.backup_database`, see the section called "Stored procedure errors".

Examples

All the examples back up a database called `MYDB` to the Amazon S3 bucket called `amzn-s3-demo-bucket` and set the *`s3_prefix`* to `backups/daily`.

Example #1: Specifying database offline and unavailable with median utilization and a single upload stream

In the following example, the database is offline, which is faster but means that the database is unavailable during backup. The example performs no compression of the files, has median impact on system resources, and uses a single upload stream to Amazon S3.

```
db2 call "rdsadmin.backup_database(
    ?,
    'MYDB',
    'amzn-s3-demo-bucket',
    'backups/daily',
    'OFFLINE',
    'NONE',
    50,
    1)"
```

Example #2: Specifying database online and available with enhanced compression, median utilization, and few parallel upload streams

In the following example, the database in online and available during backup. The example performs enhanced compression, which results in a small file size, but is CPU-intensive. It has a slightly higher than median impact on system resources and uses five upload streams to Amazon S3.

```
db2 call "rdsadmin.backup_database(
    ?,
    'MYDB',
    'amzn-s3-demo-bucket',
    'backups/daily',
    'ONLINE',
    'ZLIB',
    60,
    5)"
```

Example #3: Specifying database offline and unavailable with defaults and system calculations

In the following example, the database is offline, which is faster but means that the database is unavailable during backup. The example uses the default compression of the files and impact on system resources. It also allows RDS for Db2 to calculate the number of parallel upload streams to Amazon S3, tablespaces to read in parallel, and buffers to use.

```
db2 "call rdsadmin.backup_database(
    ?,
    'MYDB',
    'amzn-s3-demo-bucket',
    'backups/daily',
    'OFFLINE')"
```

Example #4: Specifying database offline and unavailable with no compression, high utilization, and custom calculations

In the following example, the database is offline, which is faster but means that the database is unavailable during backup. The example performs no compression of the files, has a high impact on system resources, and uses 20 upload streams to Amazon S3. It sets the maximum number of tablespaces to read in parallel, which can cause the backup request to fail if processing power is insufficient. It also sets the maximum number of buffers to use, which can cause the backup request to fail if memory is insufficient.

```
db2 "call rdsadmin.backup_database(
    ?,
    'MYDB',
    'amzn-s3-demo-bucket',
    'backups/daily',
    'OFFLINE',
    'NONE',
    90,
    20,
    256,
    268435456)"
```

rdsadmin.restore_database

Restores a database from an Amazon S3 bucket to your RDS for Db2 DB instance.

Syntax

```
db2 "call rdsadmin.restore_database(
    ?,
    'database_name',
    's3_bucket_name',
    's3_prefix',
    restore_timestamp,
```

```
    'backup_type')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the target database to restore in RDS for Db2. The data type is `varchar`.

For example, if the source database name was `TESTDB` and you set *database_name* to `NEWDB`, then Amazon RDS restores `NEWDB` as the source database.

s3_bucket_name

The name of the Amazon S3 bucket where your backup resides. The data type is `varchar`.

s3_prefix

The prefix to use for file matching during download. The data type is `varchar`.

If this parameter is empty, then all files in the Amazon S3 bucket will be processed. The following is an example prefix:

```
backupfolder/SAMPLE.0.rdsdb.DBPART000.20230615010101
```

restore_timestamp

The timestamp of the database backup image. The data type is `varchar`.

The timestamp is included in the backup file name. For example, `20230615010101` is the timestamp for the file name `SAMPLE.0.rdsdb.DBPART000.20230615010101.001`.

backup_type

The type of backup. The data type is `varchar`. Valid values: `OFFLINE`, `ONLINE`.

Use `ONLINE` for near-zero downtime migrations. For more information, see Migrating from Linux to Linux with near-zero downtime for Amazon RDS for Db2.

Usage notes

You can use this stored procedure to migrate a Db2 database to an RDS for Db2 DB instance. For more information, see Using AWS services to migrate data from Db2 to Amazon RDS for Db2. You can also use this stored procedure to create multiple copies of the same database with different database names that use the same restore image.

Before calling the stored procedure, review the following considerations:

- Before restoring a database, you must provision storage space for your RDS for Db2 DB instance that is greater than the original Db2 database on disk. If you enabled `USE_STREAMING_RESTORE`, then when you restore your backup, Amazon RDS streams the backup files directly from your S3 bucket to your RDS for Db2 DB instance. If you don't enable `USE_STREAMING_RESTORE`, you must provision storage space for your RDS for Db2 DB instance that is equal to or greater than the sum of the backup size plus the original Db2 database on disk. For more information, see Insufficient disk space.
- When you restore the backup, Amazon RDS extracts the backup file on your RDS for Db2 DB instance. Each backup file must be 5 TB or smaller. If a backup file exceeds 5 TB, then you must split the backup file into smaller files.
- To restore all files using the `rdsadmin.restore_database` stored procedure, don't include the file number suffix after the timestamp in the file names. For example, the *`s3_prefix`* `backupfolder/SAMPLE.0.rdsdb.DBPART000.20230615010101` restores the following files:

```
SAMPLE.0.rdsdb.DBPART000.20230615010101.001
SAMPLE.0.rdsdb.DBPART000.20230615010101.002
SAMPLE.0.rdsdb.DBPART000.20230615010101.003
SAMPLE.0.rdsdb.DBPART000.20230615010101.004
SAMPLE.0.rdsdb.DBPART000.20230615010101.005
```

- RDS for Db2 doesn't support non-automatic storage. For more information, see Tablespaces not restored.
- RDS for Db2 doesn't support non-fenced routines. For more information, see Non-fenced routines not allowed.
- To improve the performance of database restore operations, you can configure the number of buffers, buffer manipulators, and the number of multiple backup paths for RDS to use.

To optimize storage usage and to potentially improve performance, you can also directly stream a backup from Amazon S3. To check the current configuration, use the section called "rdsadmin.show_configuration". To change the configuration, use the section called "rdsadmin.set_configuration".

To bring the database online and apply additional transaction logs after restoring the database, see rdsadmin.rollforward_database.

For information about checking the status of restoring your database, see rdsadmin.get_task_status.

For error messages returned when calling `rdsadmin.restore_database`, see the section called "Stored procedure errors".

Examples

The following example restores an offline backup with a single file or multiple files that have the *s3_prefix* `backupfolder/SAMPLE.0.rdsdb.DBPART000.20230615010101`:

```
db2 "call rdsadmin.restore_database(
    ?,
    'SAMPLE',
    'amzn-s3-demo-bucket',
    'backupfolder/SAMPLE.0.rdsdb.DBPART000.20230615010101',
    20230615010101,
    'OFFLINE')"
```

rdsadmin.rollforward_database

Brings the database online and applies additional transaction logs after restoring a database by calling rdsadmin.restore_database.

Syntax

```
db2 "call rdsadmin.rollforward_database(
    ?,
    'database_name',
    's3_bucket_name',
    s3_prefix,
    'rollforward_to_option',
```

```
    'complete_rollforward')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database to perform the operation on. The data type is `varchar`.

s3_bucket_name

The name of the Amazon S3 bucket where your backup resides. The data type is `varchar`.

s3_prefix

The prefix to use for file matching during download. The data type is `varchar`.

If this parameter is empty, then all files in the S3 bucket will be downloaded. The following example is an example prefix:

```
backupfolder/SAMPLE.0.rdsdb.DBPART000.20230615010101
```

The following input parameters are optional:

rollforward_to_option

The point to which you want to roll forward. The data type is `varchar`. Valid values: `END_OF_LOGS`, `END_OF_BACKUP` with the timestamp in the format `YYYY-MM-DD-HH.MM.SS`. The default is `END_OF_LOGS`.

complete_rollforward

Specifies whether to complete the roll-forward process. The data type is `varchar`. The default is `TRUE`.

If TRUE, then after completion, the database is online and accessible. If FALSE, then the database remains in a ROLL-FORWARD PENDING state.

Usage notes

You can use rds.rollforward_database for an online backup with include logs that are produced on-premises in many different scenarios.

Scenario 1: Restoring the database, rolling forward the included logs, and bringing the database online

After rdsadmin.restore_database() completes, use the syntax in Example 1 to bring the database with transaction logs online.

Scenario 2: Bringing the database online but not rolling forward the included logs.

After rdsadmin.restore_database() completes, use the syntax in Example 2 to bring the database without the transaction logs online.

Scenario 3: Rolling forward the included logs in the backup, and applying additional transaction logs as they are produced on-premises

After rdsadmin.restore_database() completes, use the syntax in Example 3 or Example 4 to rollforward logs without bringing the database online.

If you set complete_rollforward to FALSE, then your database is in a ROLL-FORWARD PENDING state and offline. To bring the database online, you must call rdsadmin.complete_rollforward.

For information about checking the status of rolling forward the database, see rdsadmin.rollforward_status.

Examples

Example 1: Bringing database with transaction logs online

The following example rolls forward to an online backup of the database with transaction logs and then brings the database online:

```
db2 "call rdsadmin.rollforward_database(
    ?,
```

```
    null,
    null,
    'END_OF_LOGS',
    'TRUE')"
```

Example 2: Bringing database without transaction logs online

The following example rolls forward to an online backup of the database without transaction logs, and then brings the database online:

```
db2 "call rdsadmin.rollforward_database(
    ?,
    'TESTDB',
    'amzn-s3-demo-bucket',
    'logsfolder/,
    'END_OF_BACKUP',
    'TRUE')"
```

Example 3: Not bringing database with transaction logs online

The following example rolls forward to an online backup of the database with transaction logs, and then doesn't bring the database online:

```
db2 "call rdsadmin.rollforward_database(
    ?,
    'TESTDB',
    null,
    'onlinebackup/TESTDB',
    'END_OF_LOGS',
    'FALSE')"
```

Example 4: Not bringing database with additional transaction logs online

The following example rolls forward to an online backup of the database with additional transaction logs, and then doesn't bring the database online:

```
db2 "call rdsadmin.rollforward_database(
    ?,
    'TESTDB',
    'amzn-s3-demo-bucket',
    'logsfolder/S0000155.LOG',
    'END_OF_LOGS',
```

```
    'FALSE')"
```

rdsadmin.rollforward_status

Returns the output of `ROLLFORWARD DATABASE` *`database_name`* `QUERY STATUS`.

Syntax

```
db2 "call rdsadmin.rollforward_status(
    ?,
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

`database_name`

The name of the database to perform the operation on. The data type is `varchar`.

Usage notes

After you call rdsadmin.rollforward_database, you can call `rdsadmin.rollforward_status` to check on the status of the rollforward in the database.

For information about checking the status of this stored procedure, see rdsadmin.get_task_status.

rdsadmin.complete_rollforward

Brings database online from a `ROLL-FORWARD PENDING` state.

Syntax

```
db2 "call rdsadmin.complete_rollforward(
    ?,
```

```
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

database_name

The name of the database that you want to bring online. The data type is `varchar`.

Usage notes

If you called rdsadmin.rollforward_database with `complete_rollforward` set to `FALSE`, then your database is in a `ROLL-FORWARD PENDING` state and offline. To complete the roll-forward process and bring the database online, call `rdsadmin.complete_rollforward`.

For information about checking the status of completing the rollforward process, see rdsadmin.rollforward_status.

Examples

The following example brings the TESTDB database online:

```
db2 "call rdsadmin.complete_rollforward(
    ?,
    'TESTDB')"
```

rdsadmin.db2pd_command

Collects information about an RDS for Db2 database.

Syntax

```
db2 "call rdsadmin.db2pd_command('db2pd_cmd')"
```

Parameters

The following input parameter is required:

db2pd_cmd

The name of the db2pd command that you want to run. The data type is `varchar`.

The parameter must start with a hyphen. For a list of parameters, see db2pd - Monitor and troubleshoot Db2 database command in the IBM Db2 documentation.

The following options aren't supported:

- `-addnode`
- `-alldatabases`
- `-alldbp`
- `-alldbs`
- `-allmembers`
- `-alm_in_memory`
- `-cfinfo`
- `-cfpool`
- `-command`
- `-dbpartitionnum`
- `-debug`
- `-dump`
- `-everything`
- `-file | -o`
- `-ha`
- `-interactive`
- `-member`
- `-pages`

> **Note**
>
> `-pages summary` is supported.

- `-pdcollection`
- `-repeat`
- `-stack`
- `-totalmem`

The `file` suboption isn't supported, for example, `db2pd -db testdb -tcbstats file=tcbstat.out`.

The use of the `stacks` option isn't supported, for example, `db2pd -edus interval=5 top=10 stacks`.

Usage notes

This stored procedure gathers information that can help with monitoring and troubleshooting RDS for Db2 databases.

The stored procedure uses the IBM db2pd utility to run various commands. The db2pd utility requires `SYSADM` authorization, which the RDS for Db2 master user doesn't have. However, with the Amazon RDS stored procedure, the master user is able to use the utility to run various commands. For more information about the utility, see db2pd - Monitor and troubleshoot Db2 database command in the IBM Db2 documentation.

The output is restricted to a maximum of 2 GB.

For information about checking the status of collecting information about the database, see rdsadmin.get_task_status.

Examples

Example 1: Returning uptime of DB instance

The following example returns the uptime of an RDS for Db2 DB instance:

```
db2 "call rdsadmin.db2pd_command('-')"
```

Example 2: Returning uptime of database

The following example returns the uptime of a database called `TESTDB`:

```
db2 "call rdsadmin.db2pd_command('-db TESTDB -')"
```

Example 3: Returning memory usage of DB instance

The following example returns the memory usage of an RDS for Db2 DB instance:

```
db2 "call rdsadmin.db2pd_command('-dbptnmem')"
```

Example 4: Returning memory sets of DB instance and database

The following example returns the memory sets of an RDS for Db2 DB instance and a database called TESTDB:

```
db2 "call rdsadmin.db2pd_command('-inst -db TESTDB -memsets')"
```

rdsadmin.force_application

Forces applications off of an RDS for Db2 database.

Syntax

```
db2 "call rdsadmin.force_application(
    ?,
    'applications')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

applications

The applications that you want to force off of an RDS for Db2 database. The data type is varchar. Valid values: ALL or *application_handle*.

Separate the names of multiple applications with commas. Example: '*application_handle_1*, *application_handle_2*'.

Usage notes

This stored procedure forces all applications off of a database so you can perform maintenance.

The stored procedure uses the IBM `FORCE APPLICATION` command. The `FORCE APPLICATION` command requires `SYSADM`, `SYSMAINT`, or `SYSCTRL` authorization, which the RDS for Db2 master user doesn't have. However, with the Amazon RDS stored procedure, the master user is able to use the command. For more information, see FORCE APPLICATION command in the IBM Db2 documentation.

For information about checking the status of forcing applications off of a database, see rdsadmin.get_task_status.

Examples

Example 1: Specifying all applications

The following example forces all applications off of an RDS for Db2 database:

```
db2 "call rdsadmin.force_application(
    ?,
    'ALL')"
```

Example 2: Specifying multiple applications

The following example forces application handles 9991, 8891, and 1192 off of an RDS for Db2 database:

```
db2 "call rdsadmin.force_application(
    ?,
    '9991, 8891, 1192')"
```

rdsadmin.set_archive_log_retention

Configures the amount of time (in hours) to retain archive log files for the specified RDS for Db2 database.

Syntax

```
db2 "call rdsadmin.set_archive_log_retention(
    ?,
    'database_name',
    'archive_log_retention_hours')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database to configure archive log retention for. The data type is `varchar`.

archive_log_retention_hours

The number of hours to retain the archive log files. The data type is `smallint`. The default is `0`, and the maximum is `840` (35 days).

If the value is `0`, Amazon RDS doesn't retain the archive log files.

Usage notes

By default, RDS for Db2 retains logs for 5 minutes. We recommend that if you use replication tools such as AWS DMS for change data capture (CDC) or IBM Q Replication, you set log retention in those tools for longer than 5 minutes.

You can view the current archive log retention setting by calling the section called "rdsadmin.show_archive_log_retention".

You can't configure the archive log retention setting on the `rdsadmin` database.

Examples

Example 1: Setting retention time

The following example sets the archive log retention time for a database called TESTDB to 24 hours.

```
db2 "call rdsadmin.set_archive_log_retention(
    ?,
    'TESTDB',
    '24')"
```

Example 2: Disabling retention time

The following example disables archive log retention for a database called TESTDB.

```
db2 "call rdsadmin.set_archive_log_retention(
    ?,
    'TESTDB',
    '0')"
```

rdsadmin.show_archive_log_retention

Returns the current archive log retention setting for the specified database.

Syntax

```
db2 "call rdsadmin.show_archive_log_retention(
    ?,
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

database_name

The name of the database to show the archive log retention setting for. The data type is varchar.

Examples

The following example shows the archive log retention setting for a database called TESTDB.

```
db2 "call rdsadmin.show_archive_log_retention(?,'TESTDB')"
```

rdsadmin.list_archive_log_information

Returns details about the archive log files, such as the size, the creation date and time, and the name of individual log files for the specified database. It also returns the total storage amount used by the log files in the database.

Syntax

```
db2 "call rdsadmin.list_archive_log_information(
    ?,
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

database_name

The name of the database to list archive log information for. The data type is varchar.

Examples

The following example returns archive log information for a database called TESTDB.

```
db2 "call rdsadmin.list_archive_log_information(
    ?,
    'TESTDB')"
```

rdsadmin.enable_archive_log_copy

Enables RDS Db2 database archive log copy to Amazon S3.

Syntax

```
db2 "call rdsadmin.enable_archive_log_copy(
    ?,
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

database_name

The name of the database for which to enable archive log copy to Amazon S3. The data type is `varchar`.

Examples

The following example enables archive log copy for a database called `TESTDB`.

```
db2 "call rdsadmin.enable_archive_log_copy(
    ?,
    'TESTDB')"
```

rdsadmin.disable_archive_log_copy

Disables RDS Db2 database archive log copy to Amazon S3.

Syntax

```
db2 "call rdsadmin.disable_archive_log_copy(
```

```
    ?,
    'database_name')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameter is required:

database_name

The name of the database for which to disable archive log copy to Amazon S3. The data type is varchar.

Examples

The following example disables archive log copy for a database called TESTDB.

```
db2 "call rdsadmin.disable_archive_log_copy(
    ?,
    'TESTDB')"
```

rdsadmin.fgac_command

Runs fine-grained access control (FGAC) commands.

Syntax

```
db2 "call rdsadmin.fgac_command(
    ?,
    'database_name',
    'fgac_cmd')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database that you want to run FGAC commands on. The data type is `varchar`.

fgac_cmd

The fine-grained access control command that you want to run. The data type is `varchar`.

The following commands are valid:

- `ALTER MASK` – Alters an existing column mask in row and column access control (RCAC).
- `ALTER PERMISSION` – Alters properties of an existing row permission in RCAC.
- `ALTER SECURITY POLICY` – Alters an existing security policy for RCAC.
- `ALTER SECURITY LABEL` – Alters properties of an existing security label in label-based access control (LBAC).
- `ALTER TABLE` – Alters table structure, including adding RCAC or LBAC controls.
- `CREATE MASK` – Creates a new column mask for RCAC.
- `CREATE PERMISSION` – Creates a new row permission for RCAC.
- `CREATE SECURITY LABEL` – Creates a new security label for LBAC.
- `CREATE SECURITY POLICY` – Creates a new security policy for RCAC.
- `DROP MASK` – Drops an existing column mask.
- `DROP PERMISSION` – Drops an existing row permission.
- `DROP SECURITY LABEL` – Drops a security label from LBAC.
- `DROP SECURITY POLICY` – Drops an existing RCAC security policy.
- `GRANT EXEMPTION ON RULE` – Allows a user to bypass specific LBAC rules.
- `GRANT SECURITY LABEL` – Assigns an LBAC security label to a user.
- `REVOKE EXEMPTION ON RULE` – Removes a user's exemption from LBAC rules.
- `REVOKE SECURITY LABEL` – Removes an LBAC security label from a user.

Usage notes

This stored procedure controls access at the row or column level to table data in your database on an RDS for Db2 DB instance. RDS for Db2 supports two types of FGAC on the database:

- Label-based access control (LBAC)
- Row and column access control (RCAC)

Before calling the stored procedure, review the following considerations:

- To escape a single quote ('), use an additional single quote. The following examples show how to escape `'apple'`, `'banana'`, and `'fruit'`.

```
db2 "call rdsadmin.fgac_command(
    ?,
    'testdb',
    'CREATE SECURITY LABEL COMPONENT FRUITSET SET{''apple'',''banana''}')"
```

```
db2 "call rdsadmin.fgac_command(
    ?,
    'testdb',
    'CREATE SECURITY LABEL COMPONENT FRUITTREE TREE(''fruit'' ROOT, ''apple'' UNDER
 ''fruit'', ''banana'' UNDER ''fruit'')')"
```

- To escape brackets ([ ]), use a backslash (\). The following example shows how to escape `[''apple'',''banana'']`.

```
db2 "call rdsadmin.fgac_command(
    ?, '
    testdb',
    'CREATE SECURITY LABEL COMPONENT FRUITARRAY ARRAY\[''apple'',''banana''\]')"
```

Examples

The following examples all run FGAC commands on a database called `testdb`.

Example 1: Creating a new security label called FRUITSET

```
db2 "call rdsadmin.fgac_command(
```

```
    ?,
    'testdb',
    'CREATE SECURITY LABEL COMPONENT FRUITSET SET{''apple'',''banana''}')"
```

Example 2: Creating a new mask for the EMP_ID column that is enabled when EMP_ID is set to less than three

```
db2 "call rdsadmin.fgac_command(
    ?,
    'testdb',
    'CREATE MASK id_MASK ON EMPLOYEE FOR COLUMN EMP_ID RETURN CASE WHEN (EMP_ID < 3)
 THEN EMP_ID ELSE NULL END ENABLE')"
```

Example 3: Creating a new mask for the DEPARTMENT column that is enabled when SESSION_USER is set to security_user

```
db2 "call rdsadmin.fgac_command(
    ?,
    'testdb',
    'CREATE MASK DEPARTMENT_MASK ON EMPLOYEE FOR COLUMN DEPARTMENT RETURN CASE  WHEN
 SESSION_USER = ''security_user'' THEN DEPARTMENT ELSE NULL END ENABLE')"
```

Example 4: Creating a new security label called treelabel

```
db2 "call rdsadmin.fgac_command(
    ?,
    'testdb',
    'CREATE SECURITY LABEL COMPONENT treelabel  TREE(''COMPANY'' ROOT, ''HR'' UNDER
 ''COMPANY'', ''FINANCE'' UNDER ''COMPANY'', ''IT'' UNDER ''COMPANY'')')"
```

rdsadmin.db2support_command

Collects diagnostic information about an RDS for Db2 database and uploads it to an Amazon S3 bucket.

Syntax

```
db2 "call rdsadmin.db2support_command(
    ?,
    'database_name',
    's3_bucket_name',
```

```
    's3_prefix')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

The following input parameters are required:

database_name

The name of the database to collect diagnostic information for. The data type is `varchar`.

s3_bucket_name

The name of the Amazon S3 bucket where you want to upload the diagnostic information. The data type is `varchar`.

s3_prefix

The prefix of the path to Amazon S3 where RDS for Db2 uploads the diagnostic files. The data type is `varchar`.

Usage notes

This stored procedure collects diagnostic information that can help with troubleshooting RDS for Db2 databases and uploads the information to an Amazon S3 bucket.

The stored procedure uses the IBM `db2support` utility to collect diagnostic data. For more information about the utility, see db2support - Problem analysis and environment collection tool command in the IBM Db2 documentation.

Before calling the stored procedure, review the following considerations:

- To upload the diagnostic files to Amazon S3, you must have already configured the integration. For more information, see Integrating an Amazon RDS for Db2 DB instance with Amazon S3.
- For an RDS for Db2 DB instance to be able to interact with Amazon S3, you must have a VPC and an Amazon S3 gateway endpoint for private subnets to use. For more information, see Step

1: Create a VPC gateway endpoint for Amazon S3 and Step 2: Confirm that your VPC gateway endpoint for Amazon S3 exists.

Before calling `rdsadmin.db2support_command`, you must connect to the `rdsadmin` database. In the following example, replace *`master_username`* and *`master_password`* with your RDS for Db2 DB instance information:

```
db2 connect to rdsadmin user master_username using master_password
```

For information about checking the status of collecting diagnostic information, see rdsadmin.get_task_status.

Examples

Example 1: Collecting diagnostic information for a database

The following example collects diagnostic information for a database called `TESTDB` and uploads it to the Amazon S3 bucket called `amzn-s3-demo-bucket` with the prefix `diagnostics/testdb`:

```
db2 "call rdsadmin.db2support_command(
    ?,
    'TESTDB',
    'amzn-s3-demo-bucket',
    'diagnostics/testdb')"
```

Example 2: Collecting diagnostic information with a date-based prefix

The following example collects diagnostic information for a database called `MYDB` and uploads it to the Amazon S3 bucket called `amzn-s3-demo-bucket` with a date-based prefix:

```
db2 "call rdsadmin.db2support_command(
    ?,
    'MYDB',
    'amzn-s3-demo-bucket',
    'support/2024/01/15')"
```

Stored procedures for storage access for RDS for Db2

The built-in stored procedures described in this topic manage storage access for RDS for Db2 databases that use Amazon S3 for migrating data. For more information, see the section called "Migrating with Amazon S3".

Refer to the following built-in stored procedures for information about their syntax, parameters, usage notes, and examples.

Stored procedures

- rdsadmin.catalog_storage_access
- rdsadmin.uncatalog_storage_access

rdsadmin.catalog_storage_access

Catalogs a storage alias for accessing an Amazon S3 bucket with Db2 data files.

Syntax

```
db2 "call rdsadmin.catalog_storage_access(
    ?,
    'alias',
    's3_bucket_name',
    'grantee_type',
    'grantee'
    )"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. The datatype is `varchar`.

The following input parameters are required:

alias

The alias name for accessing remote storage in an Amazon S3 bucket. The datatype is `varchar`.

s3_bucket_name

The name of the Amazon S3 bucket where your data resides. The data type is `varchar`.

grantee_type

The type of grantee to receive authorization. The data type is `varchar`. Valid values: USER, GROUP.

grantee

The user or group to receive authorization. The data type is `varchar`.

Usage notes

Amazon RDS includes the cataloged alias in the IAM role that you added to your RDS for Db2 DB instance. If you remove the IAM role from your DB instance, then Amazon RDS deletes the alias. For more information, see the section called “Migrating with Amazon S3”.

For information about checking the status of cataloging your alias, see rdsadmin.get_task_status.

Examples

The following example registers an alias called SAMPLE. The user `jorge_souza` is granted access to the Amazon S3 bucket called `amzn-s3-demo-bucket`.

```
db2 "call rdsadmin.catalog_storage_access(
    ?,
    'SAMPLE',
    'amzn-s3-demo-bucket',
    'USER',
    'jorge_souza')"
```

rdsadmin.uncatalog_storage_access

Removes a storage access alias.

Syntax

```
db2 "call rdsadmin.uncatalog_storage_access(
    ?,
    'alias')"
```

Parameters

The following output parameter is required:

?

A parameter marker that outputs an error message. The datatype is `varchar`.

The following input parameter is required:

alias

The name of the storage alias to remove. The datatype is `varchar`.

Usage notes

For information about checking the status of removing your alias, see rdsadmin.get_task_status.

Examples

The following example removes an alias called `SAMPLE`. This alias no longer provides access to the Amazon S3 bucket it was associated with.

```
db2 "call rdsadmin.uncatalog_storage_access(
    ?,
    'SAMPLE')"
```

Stored procedures for tablespaces for RDS for Db2

The built-in stored procedures described in this topic manage tablespaces for Amazon RDS for Db2 databases. To run these procedures, the master user must first connect to the `rdsadmin` database.

These stored procedures are used in a variety of tasks. This list isn't exhaustive.

- Common tasks for tablespaces
- Generating performance reports
- Copying database metadata with db2look
- Creating a repository database for IBM Db2 Data Management Console

Refer to the following built-in stored procedures for information about their syntax, parameters, usage notes, and examples.

Stored procedures

- rdsadmin.create_tablespace
- rdsadmin.alter_tablespace
- rdsadmin.rename_tablespace
- rdsadmin.drop_tablespace

rdsadmin.create_tablespace

Creates a tablespace.

Syntax

```
db2 "call rdsadmin.create_tablespace(
    'database_name',
    'tablespace_name',
    'buffer_pool_name',
    tablespace_page_size,
    tablespace_initial_size,
    tablespace_increase_size,
    'tablespace_type',
    'tablespace_prefetch_size')"
```

Parameters

The following parameters are required:

database_name

The name of the database to create the tablespace in. The data type is `varchar`.

tablespace_name

The name of the tablespace to create. The data type is `varchar`.

The tablespace name has the following restrictions:

- It can't be the same as the name of an existing tablespace in this database.
- It can only contain the characters `_$#@a-zA-Z0-9`.
- It can't start with _ or $.
- It can't start with SYS.

The following parameters are optional:

buffer_pool_name

The name of the buffer pool to assign the tablespace. The data type is `varchar`. The default is an empty string.

> **Important**
>
> You must already have a buffer pool of the same page size to associate with the tablespace.

tablespace_page_size

The page size of the tablespace in bytes. The data type is `integer`. Valid values: `4096`, `8192`, `16384`, `32768`. The default is the page size used when you created the database by calling rdsadmin.create_database.

> **Important**
>
> Amazon RDS supports write atomicity for 4 KiB, 8 KiB, and 16 KiB pages. In contrast, 32 KiB pages risk torn writes, or partial data being written to the desk. If you use 32

KiB pages, we recommend that you enable point-in-time recovery and automated backups. Otherwise, you run the risk of being unable to recover from torn pages. For more information, see the section called “Introduction to backups” and the section called “Point-in-time recovery”.

tablespace_initial_size

The initial size of the tablespace in kilobytes (KB). The data type is `integer`. Valid values: `48` or higher. The default is null.

If you don't set a value, Db2 sets an appropriate value for you.

Note

This parameter isn't applicable for temporary tablespaces because the system manages temporary tablespaces.

tablespace_increase_size

The percentage by which to increase the tablespace when it becomes full. The data type is `integer`. Valid values: `1`–`100`. The default is null.

If you don't set a value, Db2 sets an appropriate value for you.

Note

This parameter isn't applicable for temporary tablespaces because the system manages temporary tablespaces.

tablespace_type

The type of the tablespace. The data type is `char`. Valid values: `U` (for user data), `T` (for user temporary data), or `S` (for system temporary data). The default is `U`.

tablespace_prefetch_size

The prefetch page size of the tablespace. The data type is `char`. Valid values: `AUTOMATIC` (case insensitive), or non-zero positive integers that are less than or equal to `32767`.

Usage notes

RDS for Db2 always creates a large database for data.

For information about checking the status of creating a tablespace, see rdsadmin.get_task_status.

Examples

Example 1: Creating a tablespace and assigning a buffer pool

The following example creates a tablespace called SP8 and assigns a buffer pool called BP8 for a database called TESTDB. The tablespace has an initial tablespace page size of 4,096 bytes, an initial tablespace of 1,000 KB, and a table size increase set to 50%.

```
db2 "call rdsadmin.create_tablespace(
    'TESTDB',
    'SP8',
    'BP8',
    4096,
    1000,
    50)"
```

Example 2: Creating a temporary tablespace and assigning a buffer pool

The following example creates a temporary tablespace called SP8. It assigns a buffer pool called BP8 that is 8 KiB in size for a database called TESTDB.

```
db2 "call rdsadmin.create_tablespace(
    'TESTDB',
    'SP8',
    'BP8',
    8192,
    NULL,
    NULL,
    'T')"
```

Example 3: Creating a tablespace and assigning a prefetch page size

The following example creates a tablespace called SP8 for a database called TESTDB. The tablespace has an initial tablespace increase size of 50 and a prefetch page size of 800.

```
db2 "call rdsadmin.create_tablespace(
```

```
    'TESTDB',
    'SP8',
    NULL,
    NULL,
    NULL,
    50,
    NULL,
    '800')"
```

rdsadmin.alter_tablespace

Alters a tablespace.

Syntax

```
db2 "call rdsadmin.alter_tablespace(
    'database_name',
    'tablespace_name',
    'buffer_pool_name',
    tablespace_increase_size,
    'max_size',
    'reduce_max',
    'reduce_stop',
    'reduce_value',
    'lower_high_water',
    'lower_high_water_stop',
    'switch_online',
    'tablespace_prefetch_size')"
```

Parameters

The following parameters are required:

database_name

The name of the database that uses the tablespace. The data type is `varchar`.

tablespace_name

The name of the tablespace to alter. The data type is `varchar`.

The following parameters are optional:

buffer_pool_name

The name of the buffer pool to assign the tablespace. The data type is `varchar`. The default is an empty string.

> **Important**
>
> You must already have a buffer pool of the same page size to associate with the tablespace.

tablespace_increase_size

The percentage by which to increase the tablespace when it becomes full. The data type is `integer`. Valid values: `1`–`100`. The default is `0`.

max_size

The maximum size for the tablespace. The data type is `varchar`. Valid values: *integer* `K` | `M` | `G`, or `NONE`. The default is `NONE`.

reduce_max

Specifies whether to reduce the high water mark to its maximum limit. The data type is `char`. The default is `N`.

reduce_stop

Specifies whether to interrupt a previous `reduce_max` or `reduce_value` command. The data type is `char`. The default is `N`.

reduce_value

The number or percentage to reduce the tablespace high water mark by. The data type is `varchar`. Valid values: *integer* `K`| `M` | `G`, or `1`–`100`. The default is `N`.

lower_high_water

Specifies whether to run the `ALTER TABLESPACE LOWER HIGH WATER MARK` command. The data type is `char`. The default is `N`.

lower_high_water_stop

Specifies whether to run the `ALTER TABLESPACE LOWER HIGH WATER MARK STOP` command. The data type is `char`. The default is `N`.

switch_online

Specifies whether to run the ALTER TABLESPACE SWITCH ONLINE command. The data type is char. The default is N.

tablespace_prefetch_size

The prefetch page size of the tablespace. The data type is char. Valid values: AUTOMATIC (case insensitive), or non-zero positive integers that are less than or equal to 32767.

> **Note**
>
> This parameter only works with buffer_pool_name, table_increase_size, max_size, and switch_online. It doesn't work with reduce_max, reduce_stop, reduce_value, lower_high_water, and lower_high_water_stop.

Usage notes

Before calling the stored procedure, review the following considerations:

- The rdsadmin.alter_tablespace stored procedure won't work on a tablespace with the tablespace_type set to T for user temporary data.
- The optional parameters reduce_max, reduce_stop, reduce_value, lower_high_water, lower_high_water_stop, and switch_online are mutually exclusive. You can't combine them with any other optional parameter, such as buffer_pool_name, in the rdsadmin.alter_tablespace command. For more information, see Statement not valid.

For information about checking the status of altering a tablespace, see rdsadmin.get_task_status.

For error messages returned when calling stored procedures, see the section called "Stored procedure errors".

Examples

Example 1: Lowering the high water mark

The following example alters a tablespace called SP8 and assigns a buffer pool called BP8 for a database called TESTDB to lower the high water mark.

```
db2 "call rdsadmin.alter_tablespace(
```

```
    'TESTDB',
    'SP8',
    'BP8',
    NULL,
    NULL,
    'Y')"
```

Example 2: Reducing the high water mark

The following example runs the `REDUCE MAX` command on a tablespace called `TBSP_TEST` in the database `TESTDB`.

```
db2 "call rdsadmin.alter_tablespace(
    'TESTDB',
    'TBSP_TEST',
    NULL,
    NULL,
    NULL,
    'Y')"
```

Example 3: Interrupting commands to reduce high water mark

The following example runs the `REDUCE STOP` command on a tablespace called `TBSP_TEST` in the database `TESTDB`.

```
db2 "call rdsadmin.alter_tablespace(
    'TESTDB',
    'TBSP_TEST',
    NULL,
    NULL,
    NULL,
    NULL,
    'Y')"
```

Example 4: Changing existing prefetch page size

The following example runs the `ALTER TABLESPACE SWITCH ONLINE` command on a tablespace called `TSBP_TEST` and changes the existing prefetch page size to 64.

```
db2 "call rdsadmin.alter_tablespace(
    'TESTDB',
```

```
    'TBSP_TEST',
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    ‘Y’,
    ‘64’)"
```

rdsadmin.rename_tablespace

Renames a tablespace.

Syntax

```
db2 "call rdsadmin.rename_tablespace(
    ?,
    'database_name',
    'source_tablespace_name',
    'target_tablespace_name')"
```

Parameters

The following parameters are required:

?

A parameter marker that outputs an error message. This parameter only accepts ?.

database_name

The name of the database that the tablespace belongs to. The data type is `varchar`.

source_tablespace_name

The name of the tablespace to rename. The data type is `varchar`.

target_tablespace_name

The new name of the tablespace. The data type is `varchar`.

The new name has the following restrictions:

- It can't be the same as the name of an existing tablespace.
- It can only contain the characters `_$#@a-zA-Z0-9`.
- It can't start with _ or $.
- It can't start with SYS.

Usage notes

For information about checking the status of renaming a tablespace, see rdsadmin.get_task_status.

You can't rename tablespaces that belong to the `rdsadmin` database.

Examples

The following example renames a tablespace called SP8 to SP9 in a database called TESTDB.

```
db2 "call rdsadmin.rename_tablespace(
    ?,
    'TESTDB',
    'SP8',
    'SP9')"
```

rdsadmin.drop_tablespace

Drops a tablespace.

Syntax

```
db2 "call rdsadmin.drop_tablespace(
    'database_name',
    'tablespace_name')"
```

Parameters

The following parameters are required:

database_name

The name of the database that the tablespace belongs to. The data type is `varchar`.

tablespace_name

The name of the tablespace to drop. The data type is `varchar`.

Usage notes

For information about checking the status of dropping a tablespace, see rdsadmin.get_task_status.

Examples

The following example drops a tablespace called SP8 from a database called TESTDB.

```
db2 "call rdsadmin.drop_tablespace(
    'TESTDB',
    'SP8')"
```

Amazon RDS for Db2 user-defined function reference

The following user-defined functions are available for Amazon RDS DB instances running the Db2 engine.

Topics

- rdsadmin.get_task_status
- rdsadmin.list_databases
- rdsadmin.list_modifiable_db_params

rdsadmin.get_task_status

Returns the status of a task.

Syntax

```
db2 "select task_id, task_type, database_name, lifecycle,
    varchar(bson_to_json(task_input_params), 500) as task_params,
    cast(task_output as varchar(500)) as task_output
    from table(rdsadmin.get_task_status(task_id,'database_name','task_type'))"
```

Parameters

The following parameters are optional. If you do not provide any parameters, the user-defined function returns the status of all tasks for all databases. Amazon RDS retains task history for 35 days.

task_id

The ID of the task being run. This ID is returned when you run a task. Default: `0`.

database_name

The name of the database for which the task is being run.

task_type

The type of the task to query. Valid values: `ADD_GROUPS`, `ADD_USER`, `ALTER_BUFFERPOOL`, `ALTER_TABLESPACE`, `CHANGE_PASSWORD`, `COMPLETE_ROLLFORWARD`, `CREATE_BUFFERPOOL`,

CREATE_DATABASE, CREATE_ROLE, CREATE_TABLESPACE, DROP_BUFFERPOOL, DROP_DATABASE, DROP_TABLESPACE, LIST_USERS, REMOVE_GROUPS, REMOVE_USER, RESTORE_DB, ROLLFORWARD_DB_LOG, ROLLFORWARD_STATUS, UPDATE_DB_PARAM.

Usage notes

You can use the `rdsadmin.get_task_status` user-defined function to check the status of the following tasks for Amazon RDS for Db2. This list is not exhaustive.

- Creating, altering, or dropping a buffer pool
- Creating, altering, or dropping a tablespace
- Creating or dropping a database
- Restoring a database backup from Amazon S3
- Rolling forward database logs from Amazon S3

Examples

The following example displays the columns returned when `rdsadmin.get_task_status` is called.

```
db2 "describe select * from table(rdsadmin.get_task_status())"
```

The following example lists the status of all tasks.

```
db2 "select task_id, task_type, database_name, lifecycle,
    varchar(bson_to_json(task_input_params), 500) as task_params,
    cast(task_output as varchar(500)) as task_output
    from table(rdsadmin.get_task_status(null,null,null))"
```

The following example lists the status of a specific task.

```
db2 "select task_id, task_type, database_name,
    varchar(bson_to_json(task_input_params), 500) as task_params
    from table(rdsadmin.get_task_status(1,null,null))"
```

The following example lists the status of a specific task and database.

```
db2 "select task_id, task_type, database_name,
    varchar(bson_to_json(task_input_params), 500) as task_params
    from table(rdsadmin.get_task_status(2,'SAMPLE',null))"
```

The following example lists the status of all ADD_GROUPS tasks.

```
db2 "select task_id, task_type, database_name,
    varchar(bson_to_json(task_input_params), 500) as task_params
    from table(rdsadmin.get_task_status(null,null,'add_groups'))"
```

The following example lists the status of all tasks for a specific database.

```
db2 "select task_id, task_type, database_name,
    varchar(bson_to_json(task_input_params), 500) as task_params
    from table(rdsadmin.get_task_status(null,'testdb', null))"
```

The following example outputs the JSON values as columns.

```
db2 "select varchar(r.task_type,25) as task_type, varchar(r.lifecycle,10) as lifecycle,
 r.created_at, u.* from
    table(rdsadmin.get_task_status(null,null,'restore_db')) as r,
 json_table(r.task_input_params, 'strict $' columns(s3_prefix varchar(500)
    null on empty, s3_bucket_name varchar(500) null on empty) error on error ) as U"
```

Response

The `rdsadmin.get_task_status` user-defined function returns the following columns:

`TASK_ID`

The ID of the task.

`TASK_TYPE`

Depends on the input parameters.

- ADD_GROUPS – Adds groups.
- ADD_USER – Adds a user.
- ALTER_BUFFERPOOL – Alters a buffer pool.

- ALTER_TABLESPACE – Alters a tablespace.
- CHANGE_PASSWORD – Changes a user's password.
- COMPLETE_ROLLFORWARD – Completes an rdsadmin.rollforward_database task and activates a database.
- CREATE_BUFFERPOOL – Creates a buffer pool.
- CREATE_DATABASE – Creates a database.
- CREATE_ROLE – Creates a Db2 role for a user.
- CREATE_TABLESPACE – Creates a tablespace.
- DROP_BUFFERPOOL – Drops a buffer pool.
- DROP_DATABASE – Drops a database.
- DROP_TABLESPACE – Drops a tablespace.
- LIST_USERS – Lists all users.
- REMOVE_GROUPS – Removes groups.
- REMOVE_USER – Removes a user.
- RESTORE_DB – Restores a full database.
- ROLLFORWARD_DB_LOG – Performs an rdsadmin.rollforward_database task on database logs.
- ROLLFORWARD_STATUS – Returns the status of an rdsadmin.rollforward_database task.
- UPDATE_DB_PARAM – Updates the data parameters.

DATABASE_NAME

The name of the database with which the task is associated.

COMPLETED_WORK_BYTES

The number of bytes restored by the task.

DURATION_MINS

The time taken to complete the task.

LIFECYCLE

The status of the task. Possible statuses:

- CREATED – After a task is submitted to Amazon RDS, Amazon RDS sets the status to CREATED.
- IN_PROGRESS – After a task starts, Amazon RDS sets the status to IN_PROGRESS. It can take up to 5 minutes for a status to change from CREATED to IN_PROGRESS.
- SUCCESS – After a task completes, Amazon RDS sets the status to SUCCESS.
- ERROR – If a restore task fails, Amazon RDS sets the status to ERROR. For more information about the error, see TASK_OUPUT.

CREATED_BY

The authid that created the command.

CREATED_AT

The date and time when the task was created.

LAST_UPDATED_AT

The data and time when the task was last updated.

TASK_INPUT_PARAMS

The parameters differ based on the task type. All of the input parameters are represented as a JSON object. For example, the JSON keys for the RESTORE_DB task are the following:

- DBNAME
- RESTORE_TIMESTAMP
- S3_BUCKET_NAME
- S3_PREFIX

TASK_OUTPUT

Additional information about the task. If an error occurs during native restore, this column includes information about the error.

Response examples

The following response example shows that a database called TESTJP was successfully created. For more information, see the the section called “rdsadmin.create_database” stored procedure.

```
`1 SUCCESS CREATE_DATABASE RDSDB 2023-10-24-18.32.44.962689 2023-10-24-18.34.50.038523
 1 TESTJP { "CODESET" : "IBM-437", "TERRITORY" : "JP", "COLLATION" : "SYSTEM",
 "AUTOCONFIGURE_CMD" : "", "PAGESIZE" : 4096 }
2023-10-24-18.33.30.079048 Task execution has started.

2023-10-24-18.34.50.038523 Task execution has completed successfully`.
```

The following response example explains why dropping a database failed. For more information, see the the section called “rdsadmin.drop_database” stored procedure.

```
1 ERROR DROP_DATABASE RDSDB 2023-10-10-16.33.03.744122 2023-10-10-16.33.30.143797 -
 2023-10-10-16.33.30.098857 Task execution has started.
2023-10-10-16.33.30.143797 Caught exception during executing task id 1, Aborting task.
Reason Dropping database created via rds CreateDBInstance api is not allowed.
Only database created using rdsadmin.create_database can be dropped
```

The following response example shows the successful restoration of a database. For more information, see the the section called “rdsadmin.restore_database” stored procedure.

```
1 RESTORE_DB  SAMPLE  SUCCESS

{ "S3_BUCKET_NAME" : "amzn-s3-demo-bucket", "S3_PREFIX" :
 "SAMPLE.0.rdsdb3.DBPART000.20230413183211.001", "RESTORE_TIMESTAMP" :
 "20230413183211", "BACKUP_TYPE" : "offline" }




2023-11-06-18.31.03.115795 Task execution has started.
2023-11-06-18.31.04.300231 Preparing to download
2023-11-06-18.31.08.368827 Download complete. Starting Restore
2023-11-06-18.33.13.891356 Task Completed Successfully
```

rdsadmin.list_databases

Returns a list of all databases running on an RDS for Db2 DB instance.

Syntax

```
db2 "select * from table(rdsadmin.list_databases())"
```

Usage notes

This user-defined function doesn't specify whether databases are in an activated or deactivated state.

If you don't see your databases in the list, call the the section called “rdsadmin.get_task_status” user-defined function and look for error messages.

Response

The `rdsadmin.list_databases` user-defined function returns the following columns:

`DATABASE_NAME`

The name of a database.

`CREATE_TIME`

The date and time when the database was created.

`DATABASE_UNIQUE_ID`

The RDS created GUID to uniquely identify Db2 database.

`ARCHIVE_LOG_RETENTION_HOUR`

The number of hours to retain the archive log files.

`ARCHIVE_LOG_COPY`

Displays if the feature is ENABLED or DISABLED for the database.

`ARCHIVE_LOG_LAST_UPLOAD_FILE`

Indicates last archive log uploaded to S3.

`ARCHIVE_LOG_LAST_UPLOAD_FILE_TIME`

Indicates the time when the log file was archived.

`ARCHIVE_LOG_COPY_STATUS`

Displays the status of the archive log copy.

UPLOADING : indicates that archive log files are being uploaded to S3.

CONFIGURATION_ERROR : indicates that there is a configuration issue requiring your attention.

To view detailed error look at RDS Event Messages for you Db Instance. The Event Messages can be viewed at Viewing Amazon RDS events.

Response examples

The following response example shows a list of databases and the times when they were created. `rdsadmin` is a database that Amazon RDS manages and always appears in the output.

```
DATABASE_NAME   CREATE_TIME                DATABASE_UNIQUE_ID
           ARCHIVE_LOG_RETENTION_HOURS ARCHIVE_LOG_COPY ARCHIVE_LOG_LAST_UPLOAD_FILE
 ARCHIVE_LOG_LAST_UPLOAD_FILE_TIME ARCHIVE_LOG_COPY_STATUS
-------------- -------------------------
 ------------------------------------------------- -------------------------
 --------------- --------------------------- ---------------------------------
 ------------------------------
RDSADMIN        2026-01-06-02.03.42.569069 RDSADMIN
                                0 DISABLED         -                              -
                             -
FOO             2026-01-06-02.13.42.885650 F0D81C7E-7213-4565-B376-4F33FCF420E3
                                   0 ENABLED          S0006536.LOG
 2026-01-28-19.15.10.000000        UPLOADING
CODEP           2026-01-14-19.42.42.508476 106EEF95-6E30-4FFF-85AE-B044352DF095
                                0 DISABLED         -                              -
                             -
...
```

rdsadmin.list_modifiable_db_params

Returns a list of all the modifiable database configuration parameters.

Syntax

```
db2 "select * from table(rdsadmin.list_modifiable_db_params())"
```

Usage notes

This user-defined function displays a selected lists of modifiable database parameters. These parameters can be updated using the stored procedure the section called "rdsadmin.update_db_param".

Any database parameter not included in this list has been restricted and cannot be modified.

Response

The `rdsadmin.list_modifiable_db_params` user-defined function returns the following columns:

`PARAM_NAME`

The name of the parameter that can be modified.

`DEFAULT_VALUE`

Default parameter value at the time of database creation.

`RESTART_REQUIRED`

If database recycle is required of not.

Y = Yes, Database restart is required.

N = No, Database restart is not required.

Response examples

The following is a sample (truncated) list of expected output.

```
PARAM_NAME              DEFAULT_VALUE RESTART_REQUIRED
----------------------- ------------- ----------------
ACT_SORTMEM_LIMIT       NONE          N
ARCHRETRYDELAY          20            N
AUTHN_CACHE_DURATION    3             N
AUTHN_CACHE_USERS       0             N
AUTO_CG_STATS           OFF           N
...
```

Troubleshooting for Amazon RDS for Db2

The following content can help you troubleshoot issues that you encounter with RDS for Db2.

For more information about general Amazon RDS troubleshooting issues, see Troubleshooting for Amazon RDS.

Topics

- Database connection error
- File I/O error
- Stored procedure errors

Database connection error

The following error message indicates that a database failed to connect because the server doesn't have sufficient memory.

```
SQL1643C The database manager failed to allocate shared memory because the
database manager instance memory limit has been reached.
```

Increase the memory for your DB instance and then try to connect to your database again. For information about memory usage and recommendations for databases, see the section called "Multiple Db2 databases". For information about how to update the memory for an RDS for Db2 database, see the section called "rdsadmin.update_db_param".

File I/O error

You might encounter a file I/O error for different reasons, such as when you use the LOAD command or call the `rdsadmin.restore_database` stored procedure.

In this example, you run the following LOAD command.

```
db2 "call sysproc.admin_cmd('load from "DB2REMOTE://s3test//public/datapump/t6.del" of
 del lobs from "DB2REMOTE://s3test/public/datapump/" modified by lobsinfile MESSAGES ON
 SERVER insert INTO RDSDB.t6 nonrecoverable ')"
```

The LOAD command returns the following message:

```
  Result set 1
  --------------

  ROWS_READ            ROWS_SKIPPED         ROWS_LOADED          ROWS_REJECTED
 ROWS_DELETED         ROWS_COMMITTED       ROWS_PARTITIONED     NUM_AGENTINFO_ENTRIES
 MSG_RETRIEVAL




 MSG_REMOVAL
  -------------------- -------------------- -------------------- --------------------
 -------------------- -------------------- -------------------- --------------------
 ------------------------------------------------------------------------------------------
 ------------------------------------------------------------------------------------------
                      -                    -                    -                    -
                     -                    -                    -                    -
 SELECT SQLCODE, MSG FROM TABLE(SYSPROC.ADMIN_GET_MSGS('1594987316_285548770')) AS MSG




                                                                                    CALL
 SYSPROC.ADMIN_REMOVE_MSGS('1594987316_285548770')

  1 record(s) selected.

  Return Status = 0

SQL20397W  Routine "SYSPROC.ADMIN_CMD" execution has completed, but at least
one error, "SQL1652", was encountered during the execution. More information
is available.  SQLSTATE=01H52
```

To view the error message, you run the SQL command as suggested in the previous response. `SELECT SQLCODE, MSG FROM TABLE(SYSPROC.ADMIN_GET_MSGS('1594987316_285548770')) AS MSG` returns the following message:

```
SQLCODE   MSG
---------
 ------------------------------------------------------------------------------------------
```

```
SQL2025N  An I/O error occurred. Error code “438”. Media on which this error occurred:
 “DB2REMOTE://s3test//public/datapump/t6.del”

SQL3500W The utility is beginning the LOAD phase at time “07/05/2024 21:21:48.082954”

SQL1652N File I/O error occurred
```

The Db2 diagnostic logs contain a log file similar to the following one:

```
2024-07-05-21.20.09.440609+000 I1191321E864          LEVEL: Error
PID     : 2710                 TID : 139619509200640 PROC : db2sysc 0
INSTANCE: rdsdb                NODE : 000            DB   : NTP
APPHDL  : 0-12180              APPID: xxx.xx.x.xxx.xxxxx.xxxxxxxxxxxx
UOWID   : 5                    ACTID: 1
AUTHID  : ADMIN                HOSTNAME: ip-xx-xx-x-xx
EDUID   : 147                  EDUNAME: db2lmr 0
FUNCTION: DB2 UDB, oper system services, sqloS3Client_GetObjectInfo, probe:219
MESSAGE : ZRC=0x870F01B6=-2029059658=SQLO_FAILED
          "An unexpected error is encountered"
DATA #1 : String, 29 bytes
S3:HeadObject request failed.
DATA #2 : signed integer, 4 bytes
99
DATA #3 : String, 0 bytes
Object not dumped: Address: 0x00007EFC08A9AE38 Size: 0 Reason: Zero-length data
DATA #4 : String, 33 bytes
curlCode: 28, Timeout was reached
```

This file I/O error could result from a number of different scenarios. For example, the VPC associated with the security group used to create your RDS for Db2 DB instance might lack an Amazon S3 gateway endpoint. This endpoint is essential for enabling RDS for Db2 to access Amazon S3. If your RDS for Db2 DB instance is in private subnets, then an Amazon S3 gateway endpoint is required. You can specify whether your DB instance uses private or public subnets by configuring Amazon RDS subnet groups. For more information, see Working with DB subnet groups.

Topics

- Step 1: Create a VPC gateway endpoint for Amazon S3
- Step 2: Confirm that your VPC gateway endpoint for Amazon S3 exists

Step 1: Create a VPC gateway endpoint for Amazon S3

For your RDS for Db2 DB instance to interact with Amazon S3, create a VPC and then an Amazon S3 gateway endpoint for private subnets to use.

To create a VPC gateway endpoint for S3

1. Create a VPC. For more information see Create a VPC in the *Amazon Virtual Private Cloud User Guide*.
2. Create an Amazon S3 gateway endpoint for private subnets to use. For more information, see Gateway endpoints in the *AWS PrivateLink Guide*.

Step 2: Confirm that your VPC gateway endpoint for Amazon S3 exists

Confirm that you successfully created an Amazon S3 gateway endpoint by using the AWS Management Console or the AWS CLI.

Console

To confirm an Amazon S3 gateway endpoint

1. Sign in to the AWS Management Console and open the Amazon VPC Console at https://console.aws.amazon.com/vpc.
2. In the upper-right corner of the console, choose the AWS Region of your VPC.
3. Select the VPC that you created.
4. On the **Resource map** tab, under **Network connections**, confirm that an Amazon S3 gateway endpoint is listed.

AWS CLI

To confirm an Amazon S3 gateway endpoint, run the describe-vpc-endpoints command. In the following example, replace *vpc_id* with the VPC ID, *region* with your AWS Region, and *profile* with your profile name.

For Linux, macOS, or Unix:

```
aws ec2 describe-vpc-endpoints \
    --filters "Name=vpc-id,Values=$vpc_id" \
```

```
    "Name=service-name,\
    Values=com.amazonaws.${region}.s3" \
    --region $region --profile=$profile \
    --query "VpcEndpoints[*].VpcEndpointId" --output text
```

For Windows:

```
aws ec2 describe-vpc-endpoints ^
    --filters "Name=vpc-id,Values=$vpc_id" ^
    "Name=service-name,^
    Values=com.amazonaws.${region}.s3" ^
    --region $region --profile=$profile ^
    --query "VpcEndpoints[*].VpcEndpointId" --output text
```

This command produces output similar to the following example if an Amazon S3 gateway endpoint exists.

```
[
    "vpce-0ea810434ff0b97e4"
]
```

This command produces output similar to the following example if an Amazon S3 gateway endpoint doesn't exist.

```
[]
```

If you don't see an Amazon S3 gateway endpoint listed, then Step 1: Create a VPC gateway endpoint for Amazon S3.

Stored procedure errors

This section describes various errors returned when calling stored procedures and how to resolve them.

Category	Stored procedure errors
Databases	rdsadmin.activate_database errors
Databases	rdsadmin.backup_database errors

Category	Stored procedure errors
Databases	rdsadmin.create_database errors
Databases	rdsadmin.deactivate_database errors
Databases	rdsadmin.drop_database errors
Databases	rdsadmin.reactivate_database errors
Databases	rdsadmin.restore_database errors
Databases	rdsadmin.update_db_param errors
Tablespaces	rdsadmin.alter_tablespace errors

rdsadmin.activate_database errors

The following errors can occur when you call the the section called “rdsadmin.activate_database” stored procedure.

Error	Error message
Failed to allocate shared memory	`SQL1643C The database manager failed to allocate shared memory because the database manager instance memory limit has been reached.`
Unable to activate because of running processes	`The database can’t be activated because it's in the process of being created or restored.`

Failed to allocate shared memory

The following error message indicates that the stored procedure failed to activate a database because the DB instance doesn't have sufficient memory.

```
SQL1643C The database manager failed to allocate shared memory because the database
 manager instance memory limit has been reached.
```

Increase the memory for your DB instance and then call the `rdsadmin.activate_database` stored procedure again. For information about memory usage and recommendations for databases, see the section called “Multiple Db2 databases”.

Unable to activate because of running processes

The following error message indicates that the stored procedure couldn't activate a database because the `rdsadmin.create_database` or `rdsadmin.restore_database` stored procedure is running.

```
The database can’t be activated because it's in the process of being created or
 restored.
```

Wait a few minutes, and then call the `rdsadmin.activate_database` stored procedure again.

rdsadmin.alter_tablespace errors

The following errors can occur when you call the the section called “rdsadmin.alter_tablespace” stored procedure.

Error	Error message

Statement not valid	`DB21034E The command was processed as an SQL statement because it was not a valid Command Line Processor command. During SQL processing it returned:`
	
	`SQL1763N Invalid ALTER TABLESPACE statement for table space "TBSP_TEST" due to reason "12"`

tablespace_prefetch_size value not valid	`Invalid tablespace_prefetch_size. Set value to AUTOMATIC or to a non-zero positive numerical value.`
tablespace_prefetch_size numerical value not valid	`Invalid tablespace_prefetch_size. The number of pages can't be greater than 32767.`
Parameter can't be used with tablespace_prefetch_size	`You can't use tablespace_prefetch_size with {parameter }.`

Error	Error message
Tablespace change failed	`The change to tablespace { tablespace_name} failed because you can only alter LARGE or REGULAR tablespaces.`

Statement not valid

The following error message indicates that the stored procedure combined mutually exclusive optional parameters with other optional parameters. The optional parameters `reduce_max`, `reduce_stop`, `reduce_value`, `lower_high_water`, `lower_high_water_stop`, and `switch_online` for the `rdsadmin.alter_tablespace` stored procedure are mutually exclusive. You can't combine them with any other optional parameter, such as `buffer_pool_name`, in the `rdsadmin.alter_tablespace` stored procedure. If you combine them, then when you call the `rdsadmin.get_task_status` user-defined function, Db2 will return this error message.

```
DB21034E The command was processed as an SQL statement because it was not a valid
 Command Line Processor command. During SQL processing it returned:
SQL1763N Invalid ALTER TABLESPACE statement for table space "TBSP_TEST" due to reason
 "12"
```

Call the `rdsadmin.alter_tablespace` stored procedure again without combining mutually exclusive optional parameters with other optional parameters. Then call the `rdsadmin.get_task_status` user-defined function. For more information, see rdsadmin.alter_tablespace and rdsadmin.get_task_status.

tablespace_prefetch_size value not valid

The following error message indicates that you didn't set `tablespace_prefetch_size` to `AUTOMATIC` or a non-positive numerical value. For example, you tried to set it to `testinput`.

```
Invalid tablespace_prefetch_size. Set value to AUTOMATIC or to a non-zero positive
 numerical value.
```

Call the `rdsadmin.alter_tablespace` stored procedure again and set `tablespace_prefetch_size` to `AUTOMATIC` or a non-positive numerical value.

tablespace_prefetch_size numerical value not valid

The following error message indicates that you set `tablespace_prefetch_size` to a numerical value larger than 32767.

```
Invalid tablespace_prefetch_size. The number of pages can't be greater than 32767.
```

Call the `rdsadmin.alter_tablespace` stored procedure again and set `tablespace_prefetch_size` to a non-zero positive numerical value less than or equal to 32767.

Parameter can't be used with tablespace_prefetch_size

The following error message indicates that you tried to use `tablespace_prefetch_size` with an incompatible parameter.

```
You can't use tablespace_prefetch_size with {parameter}.
```

Call the `rdsadmin.alter_tablespace` stored procedure again and only use `tablespace_prefetch_size` with compatible parameters. For information about parameters you can use with `tablespace_prefetch_size`, see rdsadmin.alter_tablespace.

Tablespace change failed

The following error message indicates that you tried to alter a tablespace.

```
The change to tablespace {tablespace_name} failed because you can only alter LARGE or
 REGULAR tablespaces.
```

rdsadmin.backup_database errors

The following errors can occur when you call the the section called “rdsadmin.backup_database” stored procedure.

Error	Error message
Insufficient disk space	`Aborting task. Reason Backing up your database failed because of insufficient disk space. Increase the storage for your DB instance and rerun the rdsadmin.backup_database stored procedure.`
Internal error	`Caught exception during executing task id 104, Aborting task. Reason Internal Error`

Insufficient disk space

The following error message indicates that your DB instance has insufficient disk space to back up your database:

```
Aborting task. Reason Backing up your database failed because of insufficient
 disk space. Increase the storage for your DB instance and rerun the
 rdsadmin.backup_database stored procedure.
```

When backing up a database to remote storage, make sure that you have sufficient free disk space for the backup session and the working files. Each backup session processes up to 5 GB of data, but additional space is needed for transaction logs, temporary files, and ongoing database operations.

We recommend that you have the following free disk space for backups based on the database size:

- For databases under 5 GB – The database size + 3 GB buffer
- For databases 5 GB and larger – At least 10 GB of free space

This amount of free disk space accounts for backup session processing, transaction log accumulation during backup, temporary working files, and parallel backup streams if configured. For more information, see Increasing DB instance storage capacity.

Increase your disk space and then call the rdsadmin.backup_database stored procedure again. To confirm that the database was backed up correctly, check the task status by using rdsadmin.get_task_status. You can also verify that the backup files exist in your Amazon S3 bucket under *s3_prefix/dbi_resource_id/db_name*.

Internal error

The following error message indicates that the stored procedure encountered an internal error:

```
Caught exception during executing task id 104, Aborting task. Reason Internal Error
```

Contact AWS Support.

rdsadmin.create_database errors

The following error can occur when you call the the section called “rdsadmin.create_database” stored procedure.

Error	Error message
Failed to allocate shared memory	`SQL1643C The database manager failed to allocate shared memory because the database manager instance memory limit has been reached.`

Failed to allocate shared memory

The following error message indicates that the stored procedure failed to create a database because the DB instance doesn't have sufficient memory.

```
SQL1643C The database manager failed to allocate shared memory because the database
 manager instance memory limit has been reached.
```

Increase the memory for your DB instance and then call the `rdsadmin.create_database` stored procedure again. For information about memory usage and recommendations for databases, see the section called “Multiple Db2 databases”.

To confirm that the database was created, call the the section called “rdsadmin.list_databases” user-defined function and check that the new database is listed.

rdsadmin.deactivate_database errors

The following error can occur when you call the the section called “rdsadmin.deactivate_database” stored procedure.

Error	Error message
Unable to deactivate because of running processes	`The database can’t be deactivated because it's in the process of being created or restored.`

Unable to deactivate because of running processes

The following error message indicates that the stored procedure couldn't deactivate a database because the `rdsadmin.create_database` or `rdsadmin.restore_database` stored procedure is running.

```
The database can’t be deactivated because it's in the process of being created or
 restored.
```

Wait a few minutes, and then call the `rdsadmin.deactivate_database` stored procedure again.

rdsadmin.drop_database errors

The following errors can occur when you call the the section called “rdsadmin.drop_database” stored procedure.

Error	Error message
Database name doesn't exist	`SQL0438N Application raised error or warning with diagnostic text: "Cannot drop database. Database with provided name does not exist". SQLSTATE=99993`
Return status = 0	`Return Status = 0`
Dropping database not allowed	`1 ERROR DROP_DATABASE RDSDB 2023-10-10-16.33.03.744122 2023-10-10-16.33.30.143797 - 2023-10-10-16.33.30.098857 Task execution has started. 2023-10-10-16.33.30.143797 Caught exception during executing task id 1, Aborting task. Reason Dropping database created via rds CreateDBInstance api is not allowed. Only database created using rdsadmin.create_database can be dropped`

Database name doesn't exist

The following error message indicates that you passed an incorrect database name in the `rdsadmin.drop_database` stored procedure.

```
SQL0438N Application raised error or warning with diagnostic text: "Cannot
drop database. Database with provided name does not exist". SQLSTATE=99993
```

Call the `rdsadmin.drop_database` stored procedure again with a correct database name. To confirm that the database was dropped, call the the section called “rdsadmin.list_databases” user-defined function and check that the dropped database isn't listed.

Return status = 0

The following error message indicates that the stored procedure couldn't be completed.

```
Return Status = 0
```

After you receive `Return Status = 0`, call the rdsadmin.get_task_status user-defined function.

Dropping database not allowed

The following error message indicates that you created the database by using either the Amazon RDS console or the AWS CLI. You can only use the `rdsadmin.drop_database` stored procedure if you created the database by calling the the section called “rdsadmin.create_database” stored procedure.

```
1 ERROR DROP_DATABASE RDSDB 2023-10-10-16.33.03.744122 2023-10-10-16.33.30.143797 -
 2023-10-10-16.33.30.098857 Task execution has started.
2023-10-10-16.33.30.143797 Caught exception during executing task id 1, Aborting task.
Reason Dropping database created via rds CreateDBInstance api is not allowed.
Only database created using rdsadmin.create_database can be dropped
```

To drop a database that you created by using either the Amazon RDS console or the AWS CLI, use a client to connect to the database and then run the appropriate command.

rdsadmin.reactivate_database errors

The following error can occur when you call the the section called “rdsadmin.reactivate_database” stored procedure.

Error	Error message
Failed to allocate shared memory	`SQL1643C The database manager failed to allocate shared memory because the database manager instance memory limit has been reached.`
Unable to reactivate because of running processes	`The database can’t be reactivated because it's in the process of being created or restored.`

Failed to allocate shared memory

The following error message indicates that the stored procedure failed to activate a database because the DB instance doesn't have sufficient memory.

```
SQL1643C The database manager failed to allocate shared memory because the database
 manager instance memory limit has been reached.
```

Increase the memory for your DB instance and then call the `rdsadmin.activate_database` stored procedure again. For information about memory usage and recommendations for databases, see the section called “Multiple Db2 databases”.

Unable to reactivate because of running processes

The following error message indicates that the stored procedure couldn't reactivate a database because the `rdsadmin.create_database` or `rdsadmin.restore_database` stored procedure is running.

```
The database can’t be reactivated because it's in the process of being created or
 restored.
```

Wait a few minutes, and then call the `rdsadmin.reactivate_database` stored procedure again.

rdsadmin.restore_database errors

The following errors can occur when you call the the section called “rdsadmin.restore_database” stored procedure:

Error	Error message
Insufficient disk space	`Aborting task. Reason Restoring your database failed because of insufficient disk space. Increase the storage for your DB instance and rerun the rdsadmin.restore_database stored procedure.`
Internal error	`Caught exception during executing task id 104, Aborting task. Reason Internal Error`
Non-fenced routines not allowed	`Caught exception during executing task id 2, Aborting task. Reason Non fenced routines are not allowed. Please delete the routines and retry the restore.`

Error	Error message
Tablespaces not restored	`Reason SQL0970N The system attempted to write to a read-only file. Reason SQL2563W The Restore process has completed successfully. However one or more table spaces from the backup were not restored.`

Insufficient disk space

The following error message indicates that your DB instance has insufficient disk space to restore your database:

```
Aborting task. Reason Restoring your database failed because of insufficient
 disk space. Increase the storage for your DB instance and rerun the
 rdsadmin.restore_database stored procedure.
```

The free space on your DB instance must be more than double the size of your backup image. If your backup image is compressed, the free space on your DB instance must be more than triple the size of your backup image. For more information, see the section called “Increasing DB instance storage capacity”.

Increase your disk space and then call the `rdsadmin.restore_database` stored procedure again. To confirm that the database was restored, call the the section called “rdsadmin.list_databases” user-defined function and check that the restored database is listed.

Internal error

The following error message indicates that the stored procedure encountered an internal error:

```
Caught exception during executing task id 104, Aborting task. Reason Internal Error
```

Contact AWS Support.

Non-fenced routines not allowed

The following error message indicates that your database contains non-fenced routines:

```
Caught exception during executing task id 2, Aborting task. Reason Non fenced routines
 are not allowed. Please delete the routines and retry the restore.
```

RDS for Db2 doesn't support non-fenced routines. Remove the non-fenced routines from the source database, and then call `rdsadmin.restore_database` again. To confirm that the database was restored, call the the section called “rdsadmin.list_databases” user-defined function and check that the restored database is listed. For more information, see the section called “Non-fenced routines”.

Tablespaces not restored

The following error message indicates that RDS for Db2 successfully restored your database, but couldn't restore one or more tablespaces:

```
Reason SQL0970N The system attempted to write to a read-only file.
Reason SQL2563W The Restore process has completed successfully. However one or more
 table spaces from the backup were not restored.
```

RDS for Db2 doesn't support non-automatic storage. Convert non-automatic storage to automatic storage and then call `rdsadmin.restore_database` again. For more information, see Converting a nonautomatic storage database to use automatic storage in the IBM Db2 documentation.

Databases with non-automatic SMS storage require manual restoration. If your database has non-automatic SMS storage, contact AWS Support.

For information about non-automatic storage and one-time migrations, see the section called “Non-automatic storage tablespaces during migration”.

rdsadmin.update_db_param errors

The following error can occur when you call the the section called “rdsadmin.update_db_param” stored procedure.

Error	Error message
Parameter not supported or modifiable	`SQL0438N Application raised error or warning with diagnostic text: "Parameter is either not supported or not modifiable to customers". SQLSTATE=99993`

Parameter not supported or modifiable

The following error message indicates that you tried to modify a database configuration parameter that either isn't supported or isn't modifiable.

```
SQL0438N Application raised error or warning with diagnostic text: "Parameter
is either not supported or not modifiable to customers". SQLSTATE=99993
```

You can see which parameters are modifiable by viewing your parameter groups. For more information, see the section called “View parameter values for a DB parameter group”.

Amazon RDS for MariaDB

Amazon RDS supports several versions of MariaDB for DB instances. For complete information about the supported versions, see MariaDB on Amazon RDS versions.

To create a MariaDB DB instance, use the Amazon RDS management tools or interfaces. You can then use the Amazon RDS tools to perform management actions for the DB instance. These include actions such as the following:

- Reconfiguring or resizing the DB instance
- Authorizing connections to the DB instance
- Creating and restoring from backups or snapshots
- Creating Multi-AZ secondaries
- Creating read replicas
- Monitoring the performance of your DB instance

To store and access the data in your DB instance, use standard MariaDB utilities and applications.

MariaDB is available in all of the AWS Regions. For more information about AWS Regions, see Regions, Availability Zones, and Local Zones.

You can use Amazon RDS for MariaDB databases to build HIPAA-compliant applications. You can store healthcare-related information, including protected health information (PHI), under a Business Associate Agreement (BAA) with AWS. For more information, see HIPAA compliance. AWS Services in Scope have been fully assessed by a third-party auditor and result in a certification, attestation of compliance, or Authority to Operate (ATO). For more information, see AWS services in scope by compliance program.

Before creating a DB instance, complete the steps in Setting up your Amazon RDS environment. When you create a DB instance, the RDS master user gets DBA privileges, with some limitations. Use this account for administrative tasks such as creating additional database accounts.

You can create the following:

- DB instances
- DB snapshots
- Point-in-time restores

- Automated backups
- Manual backups

You can use DB instances running MariaDB inside a virtual private cloud (VPC) based on Amazon VPC. You can also add features to your MariaDB DB instance by enabling various options. Amazon RDS supports Multi-AZ deployments for MariaDB as a high-availability, failover solution.

> **Important**
>
> To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. It also restricts access to certain system procedures and tables that need advanced privileges. You can access your database using standard SQL clients such as the mysql client. However, you can't access the host directly by using Telnet or Secure Shell (SSH).

Topics

- MariaDB feature support on Amazon RDS
- MariaDB on Amazon RDS versions
- Connecting to your MariaDB DB instance
- Securing MariaDB DB instance connections
- Improving query performance for RDS for MariaDB with Amazon RDS Optimized Reads
- Improving write performance with Amazon RDS Optimized Writes for MariaDB
- Upgrades of the MariaDB DB engine
- Upgrading a MariaDB DB snapshot engine version
- Importing data into an Amazon RDS for MariaDB DB instance
- Working with MariaDB replication in Amazon RDS
- Options for MariaDB database engine
- Parameters for MariaDB
- Migrating data from a MySQL DB snapshot to a MariaDB DB instance
- MariaDB on Amazon RDS SQL reference
- Local time zone for MariaDB DB instances
- Known issues and limitations for RDS for MariaDB

MariaDB feature support on Amazon RDS

RDS for MariaDB supports most of the features and capabilities of MariaDB. Some features might have limited support or restricted privileges.

You can filter new Amazon RDS features on the What's New with Database? page. For **Products**, choose **Amazon RDS**. Then search using keywords such as **`MariaDB 2023`**.

> **Note**
>
> The following lists are not exhaustive.

For more information about MariaDB feature support on Amazon RDS, see the following topics.

Topics

- Supported storage engines for MariaDB on Amazon RDS
- Cache warming for MariaDB on Amazon RDS
- MariaDB features not supported by Amazon RDS

MariaDB feature support on Amazon RDS for MariaDB major versions

In the following sections, find information about MariaDB feature support on Amazon RDS for MariaDB major versions:

Topics

- MariaDB 11.8 support on Amazon RDS
- MariaDB 11.4 support on Amazon RDS
- MariaDB 10.11 support on Amazon RDS
- MariaDB 10.6 support on Amazon RDS
- MariaDB 10.5 support on Amazon RDS
- MariaDB 10.4 support on Amazon RDS

For information about supported minor versions of Amazon RDS for MariaDB, see MariaDB on Amazon RDS versions.

MariaDB 11.8 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 11.8 or higher.

> **Note**
>
> In MariaDB 11.8, the default value for `require_secure_transport` is now 1, requiring secure SSL/TLS connections. Set to `0` if non-secure connections are needed.

- **New default value for parameter** – The default value of `require_secure_transport` parameter changed from `0` to 1, enforcing secure transport connections by default. For more information, see Requiring SSL/TLS for all connections to a MariaDB DB instance on Amazon RDS.
- **Vector support** – You can use the MariaDB Vector to store and search AI-generated vectors directly in MariaDB. This feature introduces the following system variables:
  - The variable `mhnsw_default_distance` specifies the default distance metric for MHNSW vector indexing.
  - The variable `mhnsw_default_m` defines the default value for the `M` parameter in MHNSW vector indexing.
  - The variable `mhnsw_ef_search` defines the minimal number of result candidates for vector index searches.
  - The variable `mhnsw_max_cache_size` sets the upper limit for one MHNSW vector index cache.
- **Temporary file size limits** – You can now limit the size of created disk temporary files and tables using two system variables available in the RDS Maria DB 11.8 parameter group:
  - The variable `max_tmp_session_space_usage` limits the temporary space allowance per user.
  - The variable `max_tmp_total_space_usage` limits the temporary space allowance for all users.
- **Temporary tablespace management** – The temporary tablespace stores temporary tables and grows as data is added. When temporary tables are dropped, the space is not automatically reclaimed. You can use the mysql.rds_execute_operation procedure to shrink the temporary tablespace and reclaim disk space.

For a list of all MariaDB 11.8 features and their documentation, see Changes and improvements in MariaDB 11.8 and Release notes - MariaDB 11.8 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 11.4 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 11.4 or higher.

- **Crypographic library** – RDS for MariaDB replaced OpenSSL with AWS Libcrypto (AWS-LC), which is FIPS 140-3 certified.
- **Simple Password Check plugin** – You can use the MariaDB Simple Password Check Plugin to check whether a password contains at least a specific number of characters of a specific type. For more information, see the section called "Password validation plugins".
- **Cracklib Password Check plugin** – You can use the MariaDB Cracklib Password Check Plugin to check the strength of new passwords. For more information, see the section called "Password validation plugins".
- **InnoDB enhancements** – These enhancements include the following items:
  - The change buffer was removed. For more information, see InnoDB Change Buffering.
  - InnoDB Defragmentation was removed. For more information, see InnoDB Defragmentation.
- **New privilege** – The admin user now also has the `SHOW CREATE ROUTINE` privilege. This privilege permits the grantee to view the `SHOW CREATE` definition statement of a routine that's owned by another user. For more information, see Database Privileges.
- **Replication improvement** – MariaDB version 11.4 DB instances support binlog indexing. You can create a GTID index for each binlog file. These indexes improve the performance of replication by reducing the time it takes to locate a GTID. For more information, see Binlog Indexing.
- **Deprecated or removed parameters** – The following parameters have been deprecated or removed for MariaDB version 11.4 DB instances:
  - `engine_condition_pushdown` is removed from optimizer_switch
  - innodb_change_buffer_max_size
  - innodb_defragment
  - `TLSv1.0` and `TLSv1.1` are removed from tls_version
- **New default values for a parameter** – The default value of the innodb_undo_tablespaces parameter changed from `0` to `3`.

- **New valid values for parameters** – The following parameters have new valid values for MariaDB version 11.4 DB instances:
  - The valid values for the binlog_row_image parameter now include FULL_NODUP.
  - The valid values for the OLD_MODE parameter now include NO_NULL_COLLATION_IDS.
- **New parameters** – The following parameters are new for MariaDB version 11.4 DB instances:
  - The transaction_isolation parameter replaces the tx_isolation parameter.
  - The transaction_read_only parameter replaces the tx_read_only parameter.
  - The block_encryption_mode parameter defines the default block encryption mode for the AES_ENCRYPT() and AES_DECRYPT() functions.
  - The character_set_collations defines overrides for character set default collations.
  - The binlog_gtid_index, binlog_gtid_index_page_size, and binlog_gtid_index_span_min define the properties of the binlog GTID index. For more information, see Binlog Indexing.

For a list of all MariaDB 11.4 features and their documentation, see Changes and improvements in MariaDB 11.4 and Release notes - MariaDB 11.4 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 10.11 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 10.11 or higher.

- **Password Reuse Check plugin** – You can use the MariaDB Password Reuse Check plugin to prevent users from reusing passwords and to set the retention period of passwords. For more information, see Password Reuse Check Plugin.
- **GRANT TO PUBLIC authorization** – You can grant privileges to all users who have access to your server. For more information, see GRANT TO PUBLIC.
- **Separation of SUPER and READ ONLY ADMIN privileges** – You can remove READ ONLY ADMIN privileges from all users, even users that previously had SUPER privileges.
- **Security** – You can now set option `--ssl` as the default for your MariaDB client. MariaDB no longer silently disables SSL if the configuration is incorrect.
- **SQL commands and functions** – You can now use the `SHOW ANALYZE FORMAT=JSON` command and the functions `ROW_NUMBER`, `SFORMAT`, and `RANDOM_BYTES`. `SFORMAT` allows string formatting and is enabled by default. You can convert partition to table and table to partition

in a single command. There are also several improvements around `JSON_*()` functions. `DES_ENCRYPT` and `DES_DECRYPT` functions were deprecated for version 10.10 and higher. For more information, see SFORMAT.

- **InnoDB enhancements** – These enhancements include the following items:
  - Performance improvements in the redo log to reduce write amplification and to improve concurrency.
  - The ability for you to change the undo tablespace without reinitializing the data directory. This enhancement reduces control plane overhead. It requires restarting but it doesn't require reinitialization after changing undo tablespace.
  - Support for `CHECK TABLE ... EXTENDED` and for descending indexes internally.
  - Improvements to bulk insert.
- **Binlog changes** – These changes include the following items:
  - Logging `ALTER` in two phases to decrease replication latency. The `binlog_alter_two_phase` parameter is disabled by default, but can be enabled through parameter groups.
  - Logging `explicit_defaults_for_timestamp`.
  - No longer logging `INCIDENT_EVENT` if the transaction can be safely rolled back.
- **Replication improvement**s – MariaDB version 10.11 DB instances use GTID replication by default if the master supports it. Also, `Seconds_Behind_Master` is more precise.
- **Clients** – You can use new command-line options for `mysqlbinglog` and `mariadb-dump`. You can use `mariadb-dump` to dump and restore historical data.
- **System versioning** – You can modify history. MariaDB automatically creates new partitions.
- **Atomic DDL** – `CREATE OR REPLACE` is now atomic. Either the statement succeeds or it's completely reversed.
- **Redo log write** – Redo log writes asynchronously.
- **Stored functions** – Stored functions now support the same `IN`, `OUT`, and `INOUT` parameters as in stored procedures.
- **Deprecated or removed parameters** – The following parameters have been deprecated or removed for MariaDB version 10.11 DB instances:
  - innodb_change_buffering
  - innodb_disallow_writes
  - innodb_log_write_ahead_size

- innodb_prefix_index_cluster_optimization
- keep_files_on_create
- old

- **Dynamic parameters** – The following parameters are now dynamic for MariaDB version 10.11 DB instances:
  - innodb_log_file_size
  - innodb_write_io_threads
  - innodb_read_io_threads
- **New default values for parameters** – The following parameters have new default values for MariaDB version 10.11 DB instances:
  - The default value of the explicit_defaults_for_timestamp parameter changed from `OFF` to `ON`.
  - The default value of the optimizer_prune_level parameter changed from 1 to 2.
- **New valid values for parameters** – The following parameters have new valid values for MariaDB version 10.11 DB instances:
  - The valid values for the old parameter were merged into those for the old_mode parameter.
  - The valid values for the histogram_type parameter now include `JSON_HB`.
  - The valid value range for the innodb_log_buffer_size parameter is now `262144` to `4294967295` (256KB to 4096MB).
  - The valid value range for the innodb_log_file_size parameter is now `4194304` to `512GB` (4MB to 512GB).
  - The valid values for the optimizer_prune_level parameter now include 2.
- **New parameters** – The following parameters are new for MariaDB version 10.11 DB instances:
  - The binlog_alter_two_phase parameter can improve replication performance.
  - The log_slow_min_examined_row_limit parameter can improve performance.
  - The log_slow_query parameter and the log_slow_query_file parameter are aliases for `slow_query_log` and `slow_query_log_file`, respectively.
  - optimizer_extra_pruning_depth
  - system_versioning_insert_history

For a list of all MariaDB 10.11 features and their documentation, see Changes and improvements in MariaDB 10.11 and Release notes - MariaDB 10.11 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 10.6 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 10.6 or higher:

- **MyRocks storage engine** – You can use the MyRocks storage engine with RDS for MariaDB to optimize storage consumption of your write-intensive, high-performance web applications. For more information, see Supported storage engines for MariaDB on Amazon RDS and MyRocks.
- **AWS Identity and Access Management (IAM) DB authentication** – You can use IAM DB authentication for better security and central management of connections to your MariaDB DB instances. For more information, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.
- **Upgrade options** – You can now upgrade to RDS for MariaDB version 10.6 from any prior major release (10.3, 10.4, 10.5). You can also restore a snapshot of an existing MySQL 5.6 or 5.7 DB instance to a MariaDB 10.6 instance. For more information, see Upgrades of the MariaDB DB engine.
- **Delayed replication** – You can now set a configurable time period for which a read replica lags behind the source database. In a standard MariaDB replication configuration, there is minimal replication delay between the source and the replica. With delayed replication, you can set an intentional delay as a strategy for disaster recovery. For more information, see Configuring delayed replication with MariaDB.
- **Oracle PL/SQL compatibility** – By using RDS for MariaDB version 10.6, you can more easily migrate your legacy Oracle applications to Amazon RDS. For more information, see SQL_MODE=ORACLE.
- **Atomic DDL** – Your dynamic data language (DDL) statements can be relatively crash-safe with RDS for MariaDB version 10.6. `CREATE TABLE`, `ALTER TABLE`, `RENAME TABLE`, `DROP TABLE`, `DROP DATABASE` and related DDL statements are now atomic. Either the statement succeeds, or it's completely reversed. For more information, see Atomic DDL.
- **Other enhancements** – These enhancements include a `JSON_TABLE` function for transforming JSON data to relational format within SQL, and faster empty table data load with Innodb. They also include new `sys_schema` for analysis and troubleshooting, optimizer enhancement for ignoring unused indexes, and performance improvements. For more information, see JSON_TABLE.

- **New default values for parameters** – The following parameters have new default values for MariaDB version 10.6 DB instances:
  - The default value for the following parameters has changed from `utf8` to `utf8mb3`:
    - character_set_client
    - character_set_connection
    - character_set_results
    - character_set_system

    Although the default values have changed for these parameters, there is no functional change. For more information, see Supported Character Sets and Collations in the MariaDB documentation.
  - The default value of the collation_connection parameter has changed from `utf8_general_ci` to `utf8mb3_general_ci`. Although the default value has changed for this parameter, there is no functional change.
  - The default value of the old_mode parameter has changed from unset to `UTF8_IS_UTF8MB3`. Although the default value has changed for this parameter, there is no functional change.

For a list of all MariaDB 10.6 features and their documentation, see Changes and improvements in MariaDB 10.6 and Release notes - MariaDB 10.6 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 10.5 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 10.5 or later:

- **InnoDB enhancements** – MariaDB version 10.5 includes InnoDB enhancements. For more information, see InnoDB: Performance Improvements etc. in the MariaDB documentation.
- **Performance schema updates** – MariaDB version 10.5 includes performance schema updates. For more information, see Performance Schema Updates to Match MySQL 5.7 Instrumentation and Tables in the MariaDB documentation.
- **One file in the InnoDB redo log** – In versions of MariaDB before version 10.5, the value of the `innodb_log_files_in_group` parameter was set to 2. In MariaDB version 10.5, the value of this parameter is set to 1.

If you are upgrading from a prior version to MariaDB version 10.5, and you don't modify the parameters, the `innodb_log_file_size` parameter value is unchanged. However, it applies to one log file instead of two. The result is that your upgraded MariaDB version 10.5 DB instance uses half of the redo log size that it was using before the upgrade. This change can have a noticeable performance impact. To address this issue, you can double the value of the `innodb_log_file_size` parameter. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

- **SHOW SLAVE STATUS command not supported** – In versions of MariaDB before version 10.5, the `SHOW SLAVE STATUS` command required the `REPLICATION SLAVE` privilege. In MariaDB version 10.5, the equivalent `SHOW REPLICA STATUS` command requires the `REPLICATION REPLICA ADMIN` privilege. This new privilege isn't granted to the RDS master user.

  Instead of using the `SHOW REPLICA STATUS` command, run the new `mysql.rds_replica_status` stored procedure to return similar information. For more information, see mysql.rds_replica_status.

- **SHOW RELAYLOG EVENTS command not supported** – In versions of MariaDB before version 10.5, the `SHOW RELAYLOG EVENTS` command required the `REPLICATION SLAVE` privilege. In MariaDB version 10.5, this command requires the `REPLICATION REPLICA ADMIN` privilege. This new privilege isn't granted to the RDS master user.

- **New default values for parameters** – The following parameters have new default values for MariaDB version 10.5 DB instances:

  - The default value of the max_connections parameter has changed to `LEAST({DBInstanceClassMemory/25165760},12000)`. For information about the `LEAST` parameter function, see DB parameter functions.
  - The default value of the innodb_adaptive_hash_index parameter has changed to `OFF` (`0`).
  - The default value of the innodb_checksum_algorithm parameter has changed to `full_crc32`.
  - The default value of the innodb_log_file_size parameter has changed to 2 GB.

For a list of all MariaDB 10.5 features and their documentation, see Changes and improvements in MariaDB 10.5 and Release notes - MariaDB 10.5 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

MariaDB 10.4 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MariaDB version 10.4 or later:

- **User account security enhancements** – Password expiration and account locking improvements
- **Optimizer enhancements** – Optimizer trace feature
- **InnoDB enhancements** – Instant DROP COLUMN support and instant VARCHAR extension for `ROW_FORMAT=DYNAMIC` and `ROW_FORMAT=COMPACT`
- **New parameters** – Including tcp_nodedelay, tls_version, and gtid_cleanup_batch_size

For a list of all MariaDB 10.4 features and their documentation, see Changes and improvements in MariaDB 10.4 and Release notes - MariaDB 10.4 series on the MariaDB website.

For a list of unsupported features, see MariaDB features not supported by Amazon RDS.

Supported storage engines for MariaDB on Amazon RDS

RDS for MariaDB supports the following storage engines.

Topics

- The InnoDB storage engine
- The MyRocks storage engine

Other storage engines aren't currently supported by RDS for MariaDB.

The InnoDB storage engine

Although MariaDB supports multiple storage engines with varying capabilities, not all of them are optimized for recovery and data durability. InnoDB is the recommended storage engine for MariaDB DB instances on Amazon RDS. Amazon RDS features such as point-in-time restore and snapshot restore require a recoverable storage engine and are supported only for the recommended storage engine for the MariaDB version.

For more information, see InnoDB.

The MyRocks storage engine

The MyRocks storage engine is available in RDS for MariaDB version 10.6 and higher. Before using the MyRocks storage engine in a production database, we recommend that you perform thorough benchmarking and testing to verify any potential benefits over InnoDB for your use case.

The default parameter group for MariaDB version 10.6 includes MyRocks parameters. For more information, see Parameters for MariaDB and Parameter groups for Amazon RDS.

To create a table that uses the MyRocks storage engine, specify `ENGINE=RocksDB` in the `CREATE TABLE` statement. The following example creates a table that uses the MyRocks storage engine.

```
CREATE TABLE test (a INT NOT NULL, b CHAR(10)) ENGINE=RocksDB;
```

We strongly recommend that you don't run transactions that span both InnoDB and MyRocks tables. MariaDB doesn't guarantee ACID (atomicity, consistency, isolation, durability) for transactions across storage engines. Although it is possible to have both InnoDB and MyRocks tables in a DB instance, we don't recommend this approach except during a migration from one storage engine to the other. When both InnoDB and MyRocks tables exist in a DB instance, each storage engine has its own buffer pool, which might cause performance to degrade.

MyRocks doesn't support `SERIALIZABLE` isolation or gap locks. So, generally you can't use MyRocks with statement-based replication. For more information, see MyRocks and Replication.

Currently, you can modify only the following MyRocks parameters:

- `rocksdb_block_cache_size`
- `rocksdb_bulk_load`
- `rocksdb_bulk_load_size`
- `rocksdb_deadlock_detect`
- `rocksdb_deadlock_detect_depth`
- `rocksdb_max_latest_deadlocks`

The MyRocks storage engine and the InnoDB storage engine can compete for memory based on the settings for the `rocksdb_block_cache_size` and `innodb_buffer_pool_size` parameters. In some cases, you might only intend to use the MyRocks storage engine on a particular DB instance. If so, we recommend setting the `innodb_buffer_pool_size minimal` parameter to a minimal value and setting the `rocksdb_block_cache_size` as high as possible.

You can access MyRocks log files by using the `DescribeDBLogFiles` and `DownloadDBLogFilePortion` operations.

For more information about MyRocks, see MyRocks on the MariaDB website.

Cache warming for MariaDB on Amazon RDS

InnoDB cache warming can provide performance gains for your MariaDB DB instance by saving the current state of the buffer pool when the DB instance is shut down, and then reloading the buffer pool from the saved information when the DB instance starts up. This approach bypasses the need for the buffer pool to "warm up" from normal database use and instead preloads the buffer pool with the pages for known common queries. For more information on cache warming, see Dumping and restoring the buffer pool in the MariaDB documentation.

Cache warming is enabled by default on MariaDB 10.3 and higher DB instances.
To enable it, set the `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` parameters to 1 in the parameter group for your DB instance. Changing these parameter values in a parameter group affects all MariaDB DB instances that use that parameter group. To enable cache warming for specific MariaDB DB instances, you might need to create a new parameter group for those DB instances. For information on parameter groups, see Parameter groups for Amazon RDS.

Cache warming primarily provides a performance benefit for DB instances that use standard storage. If you use PIOPS storage, you don't commonly see a significant performance benefit.

Important

If your MariaDB DB instance doesn't shut down normally, such as during a failover, then the buffer pool state isn't saved to disk. In this case, MariaDB loads whatever buffer pool file is available when the DB instance is restarted. No harm is done, but the restored buffer pool might not reflect the most recent state of the buffer pool before the restart. To ensure that you have a recent state of the buffer pool available to warm the cache on startup, we recommend that you periodically dump the buffer pool "on demand." You can dump or load the buffer pool on demand.
You can create an event to dump the buffer pool automatically and at a regular interval. For example, the following statement creates an event named `periodic_buffer_pool_dump` that dumps the buffer pool every hour.

```
CREATE EVENT periodic_buffer_pool_dump
```

```
ON SCHEDULE EVERY 1 HOUR
DO CALL mysql.rds_innodb_buffer_pool_dump_now();
```

For more information, see Events in the MariaDB documentation.

Dumping and loading the buffer pool on demand

You can save and load the cache on demand using the following stored procedures:

- To dump the current state of the buffer pool to disk, call the mysql.rds_innodb_buffer_pool_dump_now stored procedure.
- To load the saved state of the buffer pool from disk, call the mysql.rds_innodb_buffer_pool_load_now stored procedure.
- To cancel a load operation in progress, call the mysql.rds_innodb_buffer_pool_load_abort stored procedure.

MariaDB features not supported by Amazon RDS

The following MariaDB features are not supported on Amazon RDS:

- S3 storage engine
- Authentication plugin – GSSAPI
- Authentication plugin – Unix Socket
- AWS Key Management encryption plugin
- Delayed replication for MariaDB versions lower than 10.6
- Native MariaDB encryption at rest for InnoDB and Aria

  You can enable encryption at rest for a MariaDB DB instance by following the instructions in Encrypting Amazon RDS resources.
- HandlerSocket
- JSON table type for MariaDB versions lower than 10.6
- MariaDB ColumnStore
- MariaDB Galera Cluster
- Multisource replication

- MyRocks storage engine for MariaDB versions lower than 10.6
- Password validation plugin, `simple_password_check`, and `cracklib_password_check` for MariaDB versions lower than 11.4
- Spider storage engine
- Sphinx storage engine
- TokuDB storage engine
- Storage engine-specific object attributes, as described in Engine-defined new Table/Field/Index attributes in the MariaDB documentation
- Table and tablespace encryption
- Hashicorp Key Management plugin
- Running two upgrades in parallel

To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances, and it restricts access to certain system procedures and tables that require advanced privileges. Amazon RDS supports access to databases on a DB instance using any standard SQL client application. Amazon RDS doesn't allow direct host access to a DB instance by using Telnet, Secure Shell (SSH), or Windows Remote Desktop Connection.

MariaDB on Amazon RDS versions

For MariaDB, version numbers are organized as version X.Y.Z. In Amazon RDS terminology, X.Y denotes the major version, and Z is the minor version number. For Amazon RDS implementations, a version change is considered major if the major version number changes, for example going from version 10.5 to 10.6. A version change is considered minor if only the minor version number changes, for example going from version 10.6.14 to 10.6.16.

Topics

- Supported MariaDB minor versions on Amazon RDS
- Supported MariaDB major versions on Amazon RDS
- Working with the Database Preview environment
- MariaDB version 11.7 in the Database Preview environment
- Deprecated versions for Amazon RDS for MariaDB

Supported MariaDB minor versions on Amazon RDS

Amazon RDS currently supports the following minor versions of MariaDB.

> **Note**
>
> Dates with only a month and a year are approximate and are updated with an exact date when it's known.

The following table shows the minor versions of MariaDB 11.8 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
11.8.6	04 February 2026	10 February 2026	February 2027
11.8.5	14 November 2025	20 November 2025	November 2026
11.8.3	6 August 2025	25 August 2025	September 2026

The following table shows the minor versions of MariaDB 11.4 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
11.4.10	04 February 2026	10 February 2026	February 2027
11.4.9	6 November 2025	18 November 2025	November 2026
11.4.8	6 August 2025	13 August 2025	September 2026
11.4.7	22 May 2025	4 June 2025	September 2026
11.4.5	4 February 2025	24 February 2025	May 2026
11.4.4	1 November 2024	20 December 2024	May 2026
11.4.3	8 August 2024	15 October 2024	May 2026

The following table shows the minor versions of MariaDB 10.11 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.11.16	04 February 2026	10 February 2026	February 2027
10.11.15	6 November 2025	18 November 2025	November 2026
10.11.14	6 August 2025	13 August 2025	September 2026
10.11.13	22 May 2025	4 June 2025	September 2026
10.11.11	4 February 2025	24 February 2025	May 2026
10.11.10	1 November 2024	20 December 2024	May 2026
10.11.9	8 August 2024	4 September 2024	May 2026

The following table shows the minor versions of MariaDB 10.6 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.6.25	04 February 2026	10 February 2026	July 2026
10.6.24	6 November 2025	18 November 2025	July 2026
10.6.23	6 August 2025	13 August 2025	July 2026
10.6.22	6 May 2025	20 May 2025	July 2026
10.6.21	4 February 2025	24 February 2025	May 2026
10.6.20	1 November 2024	20 December 2024	May 2026
10.6.19	8 August 2024	4 September 2024	May 2026

The following table shows the minor versions of MariaDB 10.5 that Amazon RDS currently supports.

MariaDB engine version	Community release date	RDS release date	RDS end of standard support date
10.5.29	6 May 2025	20 May 2025	June 2026
10.5.28	4 February 2025	24 February 2025	June 2026
10.5.27	1 November 2024	20 December 2024	May 2026

You can specify any currently supported MariaDB version when creating a new DB instance. You can specify the major version (such as MariaDB 10.5), and any supported minor version for the specified major version. If no version is specified, Amazon RDS defaults to a supported version, typically the most recent version. If a major version is specified but a minor version is not, Amazon RDS defaults to a recent release of the major version you have specified. To see a list of supported versions, as

well as defaults for newly created DB instances, use the `describe-db-engine-versions` AWS CLI command.

For example, to list the supported engine versions for RDS for MariaDB, run the following CLI command:

```
aws rds describe-db-engine-versions --engine mariadb --query "*[].
{Engine:Engine,EngineVersion:EngineVersion}" --output text
```

The default MariaDB version might vary by AWS Region. To create a DB instance with a specific minor version, specify the minor version during DB instance creation. You can determine the default minor version for an AWS Region by running the following AWS CLI command:

```
aws rds describe-db-engine-versions --default-only --engine mariadb
 --engine-version major_engine_version --region region --query "*[].
{Engine:Engine,EngineVersion:EngineVersion}" --output text
```

Replace *major_engine_version* with the major engine version, and replace *region* with the AWS Region. For example, the following AWS CLI command returns the default MariaDB minor engine version for the 10.5 major version and the US West (Oregon) AWS Region (us-west-2):

```
aws rds describe-db-engine-versions --default-only --engine mariadb --engine-version
 10.5 --region us-west-2 --query "*[].{Engine:Engine,EngineVersion:EngineVersion}" --
output text
```

MariaDB minor versions on Amazon RDS

Minor versions

- MariaDB version 11.8.6
- MariaDB version 11.8.5
- MariaDB version 11.8.3
- MariaDB version 11.4.10
- MariaDB version 11.4.9
- MariaDB version 11.4.8
- MariaDB version 11.4.7
- MariaDB version 11.4.5
- MariaDB version 11.4.4

- MariaDB version 10.11.16
- MariaDB version 10.11.15
- MariaDB version 10.11.14
- MariaDB version 10.11.13
- MariaDB version 10.11.11
- MariaDB version 10.11.10
- MariaDB version 10.6.25
- MariaDB version 10.6.24
- MariaDB version 10.6.23
- MariaDB version 10.6.22
- MariaDB version 10.6.21
- MariaDB version 10.6.20
- MariaDB version 10.5.29
- MariaDB version 10.5.28
- MariaDB version 10.5.27

MariaDB version 11.8.6

MariaDB version 11.8.6 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025c`.
- Fixed an issue which can cause some SQL statements to not get logged in the audit log.

MariaDB version 11.8.5

MariaDB version 11.8.5 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 11.8.3

MariaDB version 11.8.3 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- **New default value for parameter** – The default value of `require_secure_transport` parameter changed from `0` to `1`, enforcing secure transport connections by default. For more information, see Requiring SSL/TLS for all connections to a MariaDB DB instance on Amazon RDS.

MariaDB version 11.4.10

MariaDB version 11.4.10 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025c`.
- Fixed an issue which can cause some SQL statements to not get logged in the audit log.

MariaDB version 11.4.9

MariaDB version 11.4.9 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 11.4.8

MariaDB version 11.4.8 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 11.4.7

MariaDB version 11.4.7 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025b`.

MariaDB version 11.4.5

MariaDB version 11.4.5 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.

MariaDB version 11.4.4

MariaDB version 11.4.4 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Reverted two MariaDB community changes that cause point-in-time recovery (PITR) to fail. For more information, see MariaDB Server Jira issue MDEV-35528.

MariaDB version 10.11.16

MariaDB version 10.11.16 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025c`.
- Fixed an issue which can cause some SQL statements to not get logged in the audit log.

MariaDB version 10.11.15

MariaDB version 10.11.15 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 10.11.14

MariaDB version 10.11.14 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 10.11.13

MariaDB version 10.11.13 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025b`.

MariaDB version 10.11.11

MariaDB version 10.11.11 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.

MariaDB version 10.11.10

MariaDB version 10.11.10 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Reverted two MariaDB community changes that cause point-in-time recovery (PITR) to fail. For more information, see MariaDB Server Jira issue MDEV-35528.

MariaDB version 10.6.25

MariaDB version 10.6.25 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025c`.
- Fixed an issue which can cause some SQL statements to not get logged in the audit log.

MariaDB version 10.6.24

MariaDB version 10.6.24 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 10.6.23

MariaDB version 10.6.23 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

MariaDB version 10.6.22

MariaDB version 10.6.22 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025b`.

MariaDB version 10.6.21

MariaDB version 10.6.21 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.

MariaDB version 10.6.20

MariaDB version 10.6.20 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Reverted two MariaDB community changes that cause point-in-time recovery (PITR) to fail. For more information, see MariaDB Server Jira issue MDEV-35528.

MariaDB version 10.5.29

MariaDB version 10.5.29 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025b`.

MariaDB version 10.5.28

MariaDB version 10.5.28 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.

MariaDB version 10.5.27

MariaDB version 10.5.27 is now available on Amazon RDS. This release contains fixes and improvements added by the MariaDB community and Amazon RDS.

New features and enhancements

- Reverted two MariaDB community changes that cause point-in-time recovery (PITR) to fail. For more information, see MariaDB Server Jira issue MDEV-35528.

Supported MariaDB major versions on Amazon RDS

RDS for MariaDB major versions remain available at least until community end of life for the corresponding community version. You can use the following dates to plan your testing and upgrade cycles. If Amazon extends support for an RDS for MariaDB version for longer than originally stated, we plan to update this table to reflect the later date.

> **Note**
>
> Dates with only a month and a year are approximate and are updated with an exact date when it's known.
> You can also view information about support dates for major engine versions by running the describe-db-major-engine-versions AWS CLI command or by using the DescribeDBMajorEngineVersions RDS API operation.

MariaDB major version	Community release date	RDS release date	Community end of life date	RDS end of standard support date
MariaDB 11.8	6 August 2025	25 August 2025	June 2030	June 2030
MariaDB 11.4	8 August 2024	15 October 2024	May 2029	May 2029

MariaDB major version	Community release date	RDS release date	Community end of life date	RDS end of standard support date
MariaDB 10.11	16 February 2023	21 August 2023	16 February 2028	February 2028
MariaDB 10.6	6 July 2021	3 February 2022	6 July 2026	August 2026
MariaDB 10.5	24 June 2020	21 January 2021	24 June 2025	August 2026

Working with the Database Preview environment

RDS for MariaDB DB instances in the Database Preview environment are functionally similar to other RDS for MariaDB DB instances. However, you can't use the Database Preview environment for production workloads.

Preview environments have the following limitations:

- Amazon RDS deletes all DB instances 60 days after you create them, along with any backups and snapshots.
- You can only use General Purpose SSD and Provisioned IOPS SSD storage.
- You can't get help from Support with DB instances. Instead, you can post your questions to the AWS-managed Q&A community, AWS re:Post.
- You can't copy a snapshot of a DB instance to a production environment.

The following options are supported by the preview.

- You can create DB instances using db.m6i, db.r6i, db.m6g, db.m5, db.t3, db.r6g, and db.r5 DB instance classes. For more information about RDS instance classes, see DB instance classes.
- You can use both single-AZ and Multi-AZ deployments.
- You can use standard MariaDB dump and load functions to export databases from or import databases to the Database Preview environment.

Features not supported in the Database Preview environment

The following features aren't available in the Database Preview environment:

- Cross-Region snapshot copy
- Cross-Region read replicas
- RDS Proxy

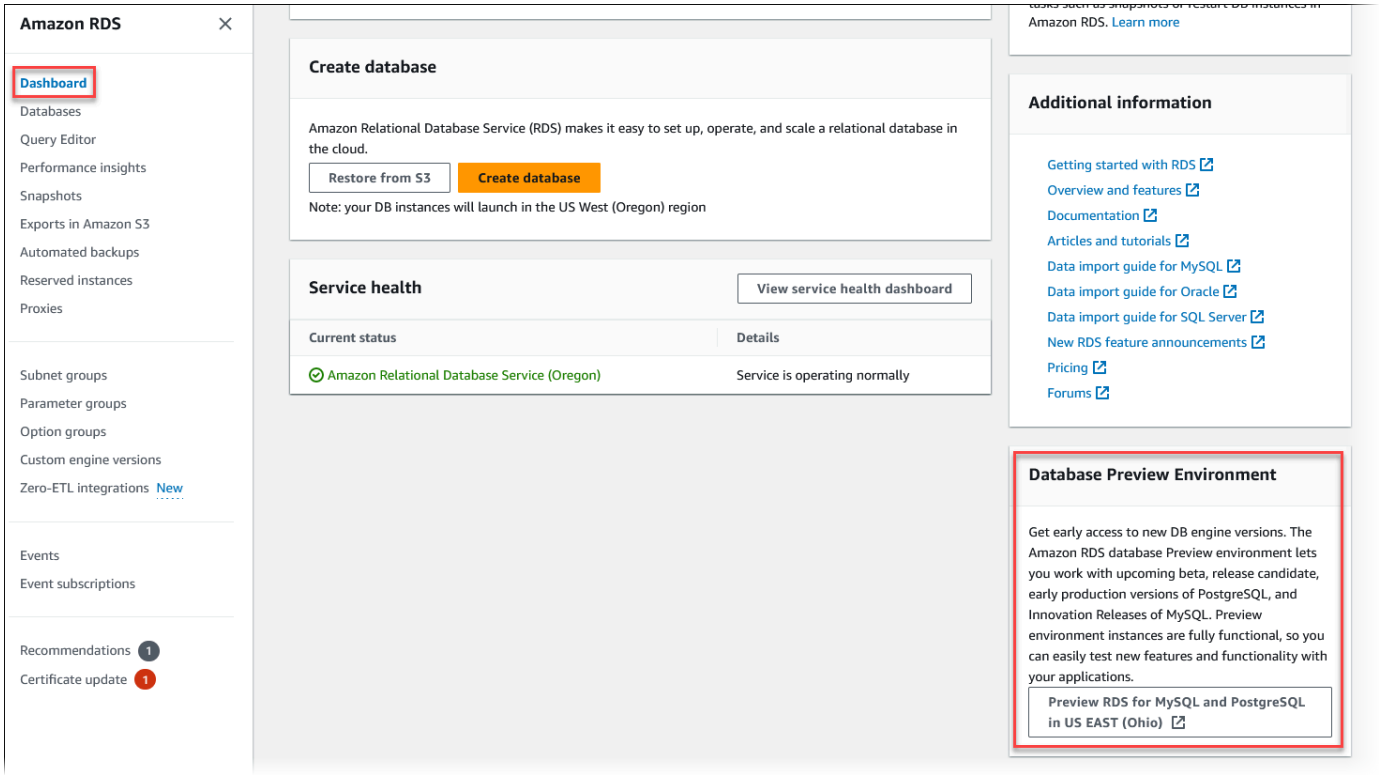
Creating a new DB instance in the Database Preview environment

You can create a DB instance in the Database Preview environment using the AWS Management Console, AWS CLI, or RDS API.

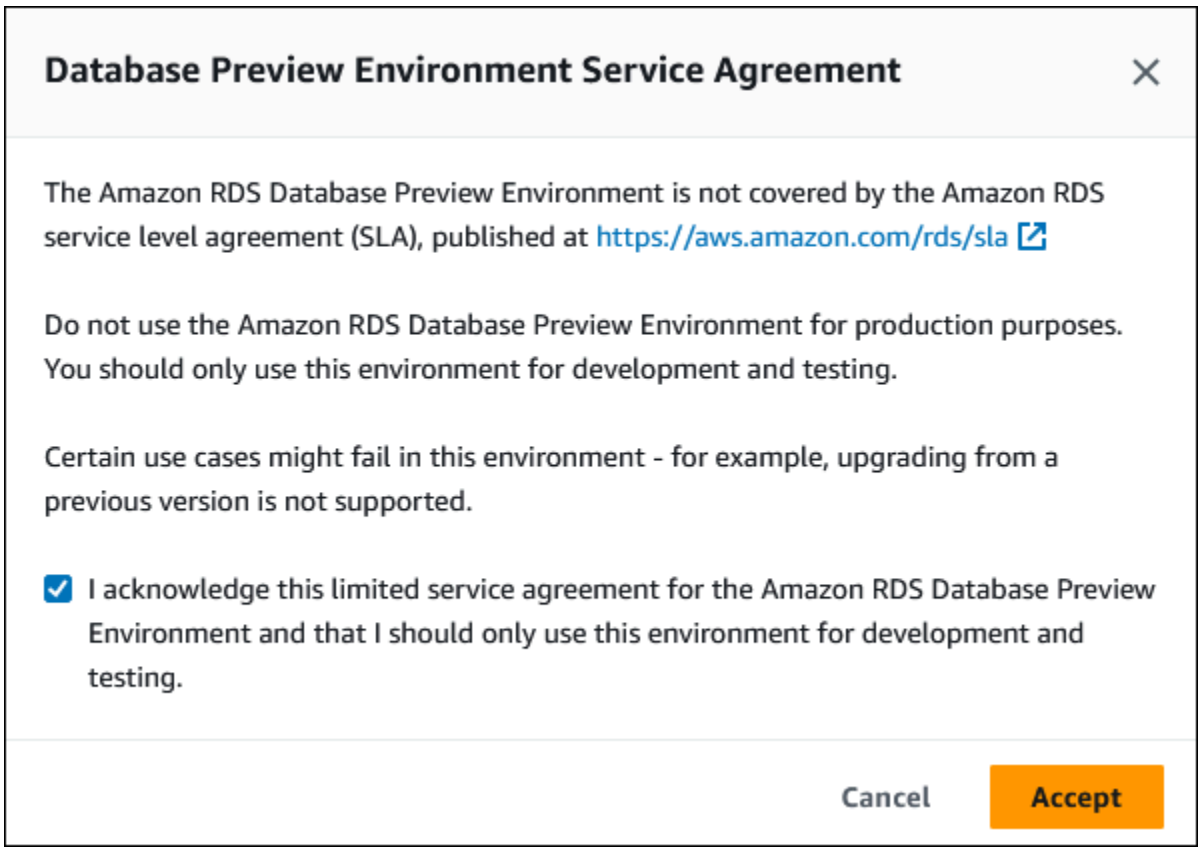
Console

To create a DB instance in the Database Preview environment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Dashboard** from the navigation pane.
3. In the **Dashboard** page, locate the **Database Preview Environment** section, as shown in the following image.



You can navigate directly to the Database Preview environment. Before you can proceed, you must acknowledge and accept the limitations.



4. To create the RDS for MariaDB DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the Console procedure in Creating a DB instance.

AWS CLI

To create a DB instance in the Database Preview environment using the AWS CLI, use the following endpoint.

```
rds-preview.us-east-2.amazonaws.com
```

To create the RDS for MariaDB DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the AWS CLI procedure in Creating a DB instance.

RDS API

To create a DB instance in the Database Preview environment using the RDS API, use the following endpoint.

```
rds-preview.us-east-2.amazonaws.com
```

To create the RDS for MariaDB DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the RDS API procedure in Creating a DB instance.

MariaDB version 11.7 in the Database Preview environment

MariaDB version 11.7 is now available in the Amazon RDS Database Preview environment. MariaDB version 11.7 contains several improvements that are described in Changes and improvements in MariaDB 11.7. This version also includes support for the vector data type, indexing, and search. For more information, see Vector Overview in the MariaDB documentation.

You can use the Database Preview environment to test your workloads against this release before it is available in all AWS Regions for production workloads. For information on the Database Preview environment, see the section called “The Database Preview environment”. To access the Preview Environment from the console, select rds-preview/.

Deprecated versions for Amazon RDS for MariaDB

Amazon RDS for MariaDB versions 10.0, 10.1, 10.2, and 10.3 are deprecated.

For information about the Amazon RDS deprecation policy for MariaDB, see Amazon RDS FAQs.

Connecting to your MariaDB DB instance

After Amazon RDS provisions your DB instance, you can use any standard MariaDB client application or utility to connect to the instance. In the connection string, you specify the Domain Name System (DNS) address from the DB instance endpoint as the host parameter. You also specify the port number from the DB instance endpoint as the port parameter.

You can connect to an Amazon RDS for MariaDB DB instance by using tools like the MySQL command-line client. For more information on using the MySQL command-line client, see mysql command-line client in the MariaDB documentation. One GUI-based application that you can use to connect is Heidi. For more information, see the Download HeidiSQL page. For information about installing MySQL (including the MySQL command-line client), see Installing and upgrading MySQL.

Most Linux distributions include the MariaDB client instead of the Oracle MySQL client. To install the MySQL command-line client on Amazon Linux 2023, run the following command:

```
sudo dnf install mariadb105
```

To install the MySQL command-line client on Amazon Linux 2, run the following command:

```
sudo yum install mariadb
```

To install the MySQL command-line client on most DEB-based Linux distributions, run the following command.

```
apt-get install mariadb-client
```

To check the version of your MySQL command-line client, run the following command.

```
mysql --version
```

To read the MySQL documentation for your current client version, run the following command.

```
man mysql
```

To connect to a DB instance from outside of a virtual private cloud (VPC) based on Amazon VPC, the DB instance must be publicly accessible. Also, access must be granted using the inbound rules of the DB instance's security group, and other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance.

You can use SSL encryption on connections to a MariaDB DB instance. For information, see SSL/TLS support for MariaDB DB instances on Amazon RDS.

To find and connect to a RDS for MariaDB DB instance, see the following topics.

Topics

- Finding the connection information for a MariaDB DB instance
- Connecting from the MySQL command-line client (unencrypted) for RDS for MariaDB
- Connecting to RDS for MariaDB with the AWS JDBC Driver and AWS Python Driver;
- Troubleshooting connections to your MariaDB DB instance

Finding the connection information for a MariaDB DB instance

The connection information for a DB instance includes its endpoint, port, and a valid database user, such as the master user. For example, suppose that an endpoint value is `mydb.123456789012.us-east-1.rds.amazonaws.com`. In this case, the port value is `3306`, and the database user is `admin`. Given this information, you specify the following values in a connection string:

- For host or host name or DNS name, specify `mydb.123456789012.us-east-1.rds.amazonaws.com`.
- For port, specify `3306`.
- For user, specify `admin`.

To connect to a DB instance, use any client for the MariaDB DB engine. For example, you might use the MySQL command-line client or MySQL Workbench.

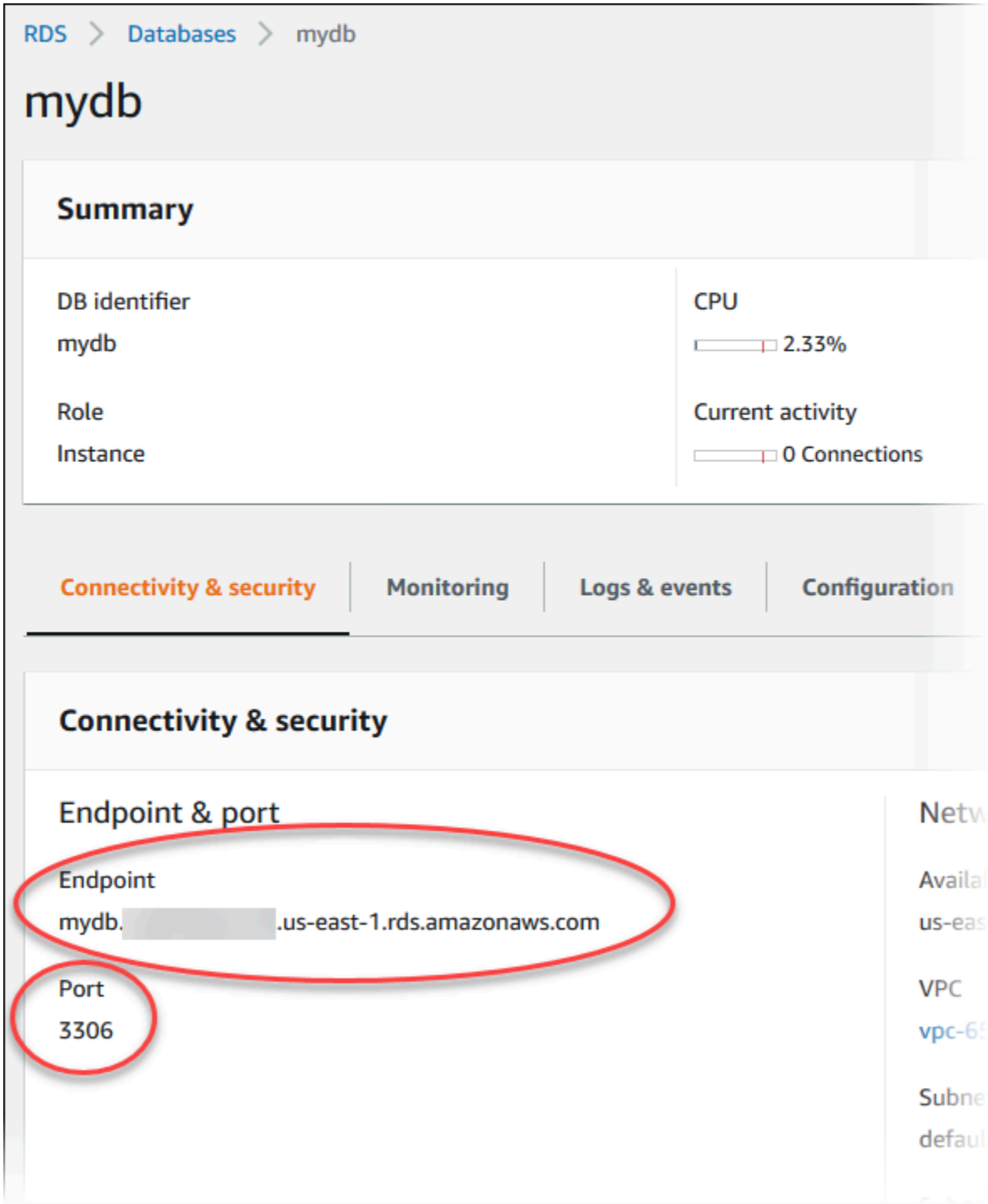
To find the connection information for a DB instance, you can use the AWS Management Console, the AWS Command Line Interface (AWS CLI) describe-db-instances command, or the Amazon RDS API DescribeDBInstances operation to list its details.

Console

To find the connection information for a DB instance in the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Databases** to display a list of your DB instances.
3. Choose the name of the MariaDB DB instance to display its details.
4. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



5. If you need to find the master user name, choose the **Configuration** tab and view the **Master username** value.

AWS CLI

To find the connection information for a MariaDB DB instance by using the AWS CLI, run the describe-db-instances command. In the call, query for the DB instance ID, endpoint, port, and master user name.

For Linux, macOS, or Unix:

```
aws rds describe-db-instances \
  --filters "Name=engine,Values=mariadb" \
  --query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

For Windows:

```
aws rds describe-db-instances ^
  --filters "Name=engine,Values=mariadb" ^
  --query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

Your output should be similar to the following.

```
[
    [
        "mydb1",
        "mydb1.123456789012.us-east-1.rds.amazonaws.com",
        3306,
        "admin"
    ],
    [
        "mydb2",
        "mydb2.123456789012.us-east-1.rds.amazonaws.com",
        3306,
        "admin"
    ]
]
```

RDS API

To find the connection information for a DB instance by using the Amazon RDS API, call the DescribeDBInstances operation. In the output, find the values for the endpoint address, endpoint port, and master user name.

Connecting from the MySQL command-line client (unencrypted) for RDS for MariaDB

> **Important**
>
> Only use an unencrypted MySQL connection when the client and server are in the same VPC and the network is trusted. For information about using encrypted connections, see Connecting to your MariaDB DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted).

To connect to a DB instance using the MySQL command-line client, enter the following command at a command prompt on a client computer. Doing this connects you to a database on a MariaDB DB instance. Substitute the DNS name (endpoint) for your DB instance for *<endpoint>* and the master user name that you used for *<mymasteruser>*. Provide the master password that you used when prompted for a password.

```
mysql -h <endpoint> -P 3306 -u <mymasteruser> -p
```

After you enter the password for the user, you see output similar to the following.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.6.10-MariaDB-log Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Connecting to RDS for MariaDB with the AWS JDBC Driver and AWS Python Driver;

Connect to RDS for MariaDB DB instances with the AWS JDBC Driver and the AWS Python Driver. For more information, see the following topics.

Topics

- Connecting to RDS for MariaDB with the Amazon Web Services (AWS) JDBC Driver
- Connecting to RDS for MariaDB with the Amazon Web Services (AWS) Python Driver

Connecting to RDS for MariaDB with the Amazon Web Services (AWS) JDBC Driver

The Amazon Web Services (AWS) JDBC Driver is designed as an advanced JDBC wrapper. This wrapper is complementary to and extends the functionality of an existing JDBC driver. The driver is drop-in compatible with the community MySQL Connector/J driver and the community MariaDB Connector/J driver.

To install the AWS JDBC Driver, append the AWS JDBC Driver .jar file (located in the application CLASSPATH), and keep references to the respective community driver. Update the respective connection URL prefix as follows:

- `jdbc:mysql://` to `jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://` to `jdbc:aws-wrapper:mariadb://`

For more information about the AWS JDBC Driver and complete instructions for using it, see the Amazon Web Services (AWS) JDBC Driver GitHub repository.

Connecting to RDS for MariaDB with the Amazon Web Services (AWS) Python Driver

The Amazon Web Services (AWS) Python Driver is designed as an advanced Python wrapper. This wrapper is complementary to and extends the functionality of the open-source Psycopg driver. The AWS Python Driver supports Python versions 3.8 and higher. You can install the `aws-advanced-python-wrapper` package using the `pip` command, along with the `psycopg` open-source packages.

For more information about the AWS Python Driver and complete instructions for using it, see the Amazon Web Services (AWS) Python Driver GitHub repository.

Troubleshooting connections to your MariaDB DB instance

Two common causes of connection failures to a new DB instance are the following:

- The DB instance was created using a security group that doesn't authorize connections from the device or Amazon EC2 instance where the MariaDB application or utility is running. The DB instance must have a VPC security group that authorizes the connections. For more information, see Amazon VPC and Amazon RDS.

  You can add or edit an inbound rule in the security group. For **Source**, choose **My IP**. This allows access to the DB instance from the IP address detected in your browser.
- The DB instance was created using the default port of 3306, and your company has firewall rules blocking connections to that port from devices in your company network. To fix this failure, recreate the instance with a different port.

For more information on connection issues, see Can't connect to Amazon RDS DB instance.

Securing MariaDB DB instance connections

You can manage the security of your MariaDB DB instances.

Topics

- MariaDB security on Amazon RDS
- Using the password validation plugins for RDS for MariaDB
- Encrypting client connections with SSL/TLS to MariaDB DB instances on Amazon RDS
- Updating applications to connect to MariaDB instances using new SSL/TLS certificates

MariaDB security on Amazon RDS

Security for MariaDB DB instances is managed at three levels:

- AWS Identity and Access Management controls who can perform Amazon RDS management actions on DB instances. When you connect to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see Identity and access management for Amazon RDS.
- When you create a DB instance, you use a VPC security group to control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance. These connections can be made using Secure Socket Layer (SSL) and Transport Layer Security (TLS). In addition, firewall rules at your company can control whether devices running at your company can open connections to the DB instance.
- Once a connection has been opened to a MariaDB DB instance, authentication of the login and permissions are applied the same way as in a stand-alone instance of MariaDB. Commands such as `CREATE USER`, `RENAME USER`, `GRANT`, `REVOKE`, and `SET PASSWORD` work just as they do in stand-alone databases, as does directly modifying database schema tables.

When you create an Amazon RDS DB instance, the master user has the following default privileges:

- `alter`
- `alter routine`
- `create`
- `create routine`
- `create temporary tables`

- `create user`
- `create view`
- `delete`
- `drop`
- `event`
- `execute`
- `grant option`
- `index`
- `insert`
- `lock tables`
- `process`
- `references`
- `reload`

  This privilege is limited on MariaDB DB instances. It doesn't grant access to the `FLUSH LOGS` or `FLUSH TABLES WITH READ LOCK` operations.
- `replication client`
- `replication slave`
- `select`
- `show create routine`

  This privilege is only on MariaDB DB instances running version 11.4 and higher.
- `show databases`
- `show view`
- `trigger`
- `update`

For more information about these privileges, see User account management in the MariaDB documentation.

> **Note**
>
> Although you can delete the master user on a DB instance, we don't recommend doing so. To recreate the master user, use the `ModifyDBInstance` API or the `modify-db-instance` AWS CLI and specify a new master user password with the appropriate parameter. If the master user does not exist in the instance, the master user is created with the specified password.

To provide management services for each DB instance, the `rdsadmin` user is created when the DB instance is created. Attempting to drop, rename, change the password for, or change privileges for the `rdsadmin` account results in an error.

To allow management of the DB instance, the standard `kill` and `kill_query` commands have been restricted. The Amazon RDS commands `mysql.rds_kill`, `mysql.rds_kill_query`, and `mysql.rds_kill_query_id` are provided for use in MariaDB and also MySQL so that you can end user sessions or queries on DB instances.

Using the password validation plugins for RDS for MariaDB

Starting with RDS for MariaDB version 11.4, you can use the following password validation plugins to enhance the security of your database connections:

- simple_password_check – checks whether a password contains at least a specific number of characters of a specific type.
- cracklib_password_check – checks whether a password appears in a dictionary file of the CrackLib library.

To enable these plugins, set the value of the parameter `simple_password_check` or `cracklib_password_check` to `FORCE_PLUS_PERMANENT` in the DB parameter group associated with the DB instance. When this value is set, the plugin can't be uninstalled by using the `UNINSTALL PLUGIN` statement at runtime.

To disable these plugins, set the value of the parameter `simple_password_check` or `cracklib_password_check` to `OFF` in the DB parameter group associated with the DB instance. When this value is set, the plugin validation rules no longer apply for new passwords.

For information about setting the values of parameters in parameter groups, see the section called "Modifying parameters in a DB parameter group".

Encrypting client connections with SSL/TLS to MariaDB DB instances on Amazon RDS

Secure Sockets Layer (SSL) is an industry-standard protocol for securing network connections between client and server. After SSL version 3.0, the name was changed to Transport Layer Security (TLS). Amazon RDS supports SSL/TLS encryption for MariaDB DB instances. Using SSL/TLS, you can encrypt a connection between your application client and your MariaDB DB instance. SSL/TLS support is available in all AWS Regions.

With Amazon RDS, you can secure data in transit by encrypting client connections to MariaDB DB instances with SSL/TLS, requiring SSL/TLS for all connections to a MariaDB DB instance, and connecting from the MySQL command-line client with SSL/TLS (encrypted). The following sections provide guidance on configuring and utilizing SSL/TLS encryption for MariaDB DB instances on Amazon RDS.

Topics

- SSL/TLS support for MariaDB DB instances on Amazon RDS
- Requiring SSL/TLS for specific user accounts to a MariaDB DB instance on Amazon RDS
- Requiring SSL/TLS for all connections to a MariaDB DB instance on Amazon RDS
- Connecting to your MariaDB DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted)

SSL/TLS support for MariaDB DB instances on Amazon RDS

Amazon RDS creates an SSL/TLS certificate and installs the certificate on the DB instance when Amazon RDS provisions the instance. These certificates are signed by a certificate authority. The SSL/TLS certificate includes the DB instance endpoint as the Common Name (CN) for the SSL/TLS certificate to guard against spoofing attacks.

An SSL/TLS certificate created by Amazon RDS is the trusted root entity and should work in most cases, but might fail if your application doesn't accept certificate chains. If your application doesn't accept certificate chains, try using an intermediate certificate to connect to your AWS Region. For example, you must use an intermediate certificate to connect to the AWS GovCloud (US) Regions with SSL/TLS.

For information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . For more information about using SSL/TLS with MySQL, see Updating applications to connect to MariaDB instances using new SSL/TLS certificates.

Amazon RDS for MariaDB supports Transport Layer Security (TLS) versions 1.3, 1.2, 1.1, and 1.0. TLS support depends on the MariaDB minor version. The following table shows the TLS support for MariaDB minor versions.

TLS version	MariaDB 11.8	MariaDB 11.4	MariaDB 10.11	MariaDB 10.6	MariaDB 10.5	MariaDB 10.4
TLS 1.3	All minor versions	All minor versions	All minor versions	All minor versions	All minor versions	All minor versions
TLS 1.2	All minor versions	All minor versions	All minor versions	All minor versions	All minor versions	All minor versions
TLS 1.1	Not supported	Not supported	Not supported	10.6.16 and lower	10.5.23 and lower	10.4.32 and lower
TLS 1.0	Not supported	Not supported	Not supported	10.6.16 and lower	10.5.23 and lower	10.4.32 and lower

Requiring SSL/TLS for specific user accounts to a MariaDB DB instance on Amazon RDS

You can require SSL/TLS encryption for specified user account connections to your MariaDB DB instances on Amazon RDS. Protecting sensitive information from unauthorized access or interception is crucial to enforce security policies where data confidentiality is a concern.

To require SSL/TLS connections for specific users' accounts, use one of the following statements, depending on your MySQL version, to require SSL/TLS connections on the user account `encrypted_user`.

To do so, use the following statement.

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

For more information on SSL/TLS connections with MariaDB, see Securing Connections for Client and Server in the MariaDB documentation.

Requiring SSL/TLS for all connections to a MariaDB DB instance on Amazon RDS

Use the `require_secure_transport` parameter to require that all user connections to your MariaDB DB instance use SSL/TLS. For versions 11.4 and earlier, the `require_secure_transport` parameter is set to `OFF` by default. For 11.8 and later versions, the default value is set to `ON`, enforcing SSL/TLS for connections to your DB instance. You can change the `require_secure_transport` parameter to `OFF` if non-secure connections are needed.

> **Note**
>
> The `require_secure_transport` parameter is only supported for MariaDB version 10.5 and higher.

You can set the `require_secure_transport` parameter value by updating the DB parameter group for your DB instance. You don't need to reboot your DB instance for the change to take effect.

When the `require_secure_transport` parameter is set to `ON` for a DB instance, a database client can connect to it if it can establish an encrypted connection. Otherwise, an error message similar to the following is returned to the client:

```
ERROR 1045 (28000): Access denied for user 'USER'@'localhost' (using password: YES |
 NO)
```

For information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

For more information about the `require_secure_transport` parameter, see the MariaDB documentation.

Connecting to your MariaDB DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted)

The `mysql` client program parameters are slightly different if you are using the MySQL 5.7 version, the MySQL 8.0 version, or the MariaDB version.

To find out which version you have, run the `mysql` command with the `--version` option. In the following example, the output shows that the client program is from MariaDB.

```
$ mysql --version
mysql  Ver 15.1 Distrib 10.5.15-MariaDB, for osx10.15 (x86_64) using readline 5.1
```

Most Linux distributions, such as Amazon Linux, CentOS, SUSE, and Debian have replaced MySQL with MariaDB, and the `mysql` version in them is from MariaDB.

To connect to your DB instance using SSL/TLS, follow these steps:

To connect to a DB instance with SSL/TLS using the MySQL command-line client

1. Download a root certificate that works for all AWS Regions.

   For information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

2. Use a MySQL command-line client to connect to a DB instance with SSL/TLS encryption. For the `-h` parameter, substitute the DNS name (endpoint) for your DB instance. For the `--ssl-ca` parameter, substitute the SSL/TLS certificate file name. For the `-P` parameter, substitute the port for your DB instance. For the `-u` parameter, substitute the user name of a valid database user, such as the master user. Enter the master user password when prompted.

   The following example shows how to launch the client using the `--ssl-ca` parameter using the MariaDB client:

   ```
   mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-bundle.pem --ssl -P 3306 -u myadmin -p
   ```

   To require that the SSL/TLS connection verifies the DB instance endpoint against the endpoint in the SSL/TLS certificate, enter the following command:

   ```
   mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-bundle.pem --ssl-verify-server-cert -P 3306 -u myadmin -p
   ```

   The following example shows how to launch the client using the `--ssl-ca` parameter using the MySQL 5.7 client or later:

```
mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-bundle.pem --ssl-mode=REQUIRED -P 3306 -u myadmin -p
```

3. Enter the master user password when prompted.

You should see output similar to the following.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.6.10-MariaDB-log Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Updating applications to connect to MariaDB instances using new SSL/TLS certificates

As of January 13, 2023, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your RDS DB instances using Secure Socket Layer or Transport Layer Security (SSL/TLS). Following, you can find information about updating your applications to use the new certificates.

This topic can help you to determine whether your applications require certificate verification to connect to your DB instances.

> **Note**
>
> Some applications are configured to connect to MariaDB only if they can successfully verify the certificate on the server. For such applications, you must update your client application trust stores to include the new CA certificates.
> You can specify the following SSL modes: `disabled`, `preferred`, and `required`. When you use the `preferred` SSL mode and the CA certificate doesn't exist or isn't up to date, the connection falls back to not using SSL and still connects successfully.

We recommend avoiding `preferred` mode. In `preferred` mode, if the connection encounters an invalid certificate, it stops using encryption and proceeds unencrypted.

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB instances. We strongly recommend testing these procedures in a development or staging environment before implementing them in your production environments.

For more information about certificate rotation, see Rotating your SSL/TLS certificate. For more information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . For information about using SSL/TLS with MariaDB DB instances, see SSL/TLS support for MariaDB DB instances on Amazon RDS.

Topics

- Determining whether a client requires certificate verification in order to connect
- Updating your application trust store
- Example Java code for establishing SSL connections

Determining whether a client requires certificate verification in order to connect

You can check whether JDBC clients and MySQL clients require certificate verification to connect.

JDBC

The following example with MySQL Connector/J 8.0 shows one way to check an application's JDBC connection properties to determine whether successful connections require a valid certificate. For more information on all of the JDBC connection options for MySQL, see Configuration properties in the MySQL documentation.

When using the MySQL Connector/J 8.0, an SSL connection requires verification against the server CA certificate if your connection properties have `sslMode` set to `VERIFY_CA` or `VERIFY_IDENTITY`, as in the following example.

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

Note

If you use either the MySQL Java Connector v5.1.38 or later, or the MySQL Java Connector v8.0.9 or later to connect to your databases, even if you haven't explicitly configured your applications to use SSL/TLS when connecting to your databases, these client drivers default to using SSL/TLS. In addition, when using SSL/TLS, they perform partial certificate verification and fail to connect if the database server certificate is expired.
Specify a password other than the prompt shown here as a security best practice.

MySQL

The following examples with the MySQL Client show two ways to check a script's MySQL connection to determine whether successful connections require a valid certificate. For more information on all of the connection options with the MySQL Client, see Client-side configuration for encrypted connections in the MySQL documentation.

When using the MySQL 5.7 or MySQL 8.0 Client, an SSL connection requires verification against the server CA certificate if for the `--ssl-mode` option you specify `VERIFY_CA` or `VERIFY_IDENTITY`, as in the following example.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

When using the MySQL 5.6 Client, an SSL connection requires verification against the server CA certificate if you specify the `--ssl-verify-server-cert` option, as in the following example.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-verify-server-cert
```

Updating your application trust store

For information about updating the trust store for MySQL applications, see Using TLS/SSL with MariaDB Connector/J in the MariaDB documentation.

For information about downloading the root certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

For sample scripts that import certificates, see Sample script for importing certificates into your trust store.

> **Note**
>
> When you update the trust store, you can retain older certificates in addition to adding the new certificates.

If you are using the MariaDB Connector/J JDBC driver in an application, set the following properties in the application.

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

When you start the application, set the following properties.

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

> **Note**
>
> Specify passwords other than the prompts shown here as a security best practice.

Example Java code for establishing SSL connections

The following code example shows how to set up the SSL connection using JDBC.

```
private static final String DB_USER = "admin";
```

```
        private static final String DB_USER = "user name";
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void main(String[] args) throws Exception {
        Class.forName("org.mariadb.jdbc.Driver");

        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);


        Connection connection = DriverManager.getConnection("jdbc:mysql://ssl-mariadb-
public.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306?useSSL=true",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
```

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-rsa2048-g1 certificates. For instructions, see step 3 in Updating your CA certificate by modifying your DB instance or cluster.

Specify a password other than the prompt shown here as a security best practice.

Improving query performance for RDS for MariaDB with Amazon RDS Optimized Reads

You can achieve faster query processing for RDS for MariaDB with Amazon RDS Optimized Reads. An RDS for MariaDB DB instance that uses RDS Optimized Reads can achieve up to 2x faster query processing compared to a DB instance that doesn't use it.

Topics

- Overview of RDS Optimized Reads
- Use cases for RDS Optimized Reads
- Best practices for RDS Optimized Reads
- Using RDS Optimized Reads
- Monitoring DB instances that use RDS Optimized Reads
- Limitations for RDS Optimized Reads

Overview of RDS Optimized Reads

When you use an RDS for MariaDB DB instance that has RDS Optimized Reads turned on, your DB instance achieves faster query performance through the use of an instance store. An *instance store* provides temporary block-level storage for your DB instance. The storage is located on Non-Volatile Memory Express (NVMe) solid state drives (SSDs) that are physically attached to the host server. This storage is optimized for low latency, high random I/O performance, and high sequential read throughput.

RDS Optimized Reads is turned on by default when a DB instance uses a DB instance class with an instance store, such as db.m5d or db.m6gd. With RDS Optimized Reads, some temporary objects are stored on the instance store. These temporary objects include internal temporary files, internal on-disk temp tables, memory map files, and binary log (binlog) cache files. For more information about the instance store, see Amazon EC2 instance store in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

The workloads that generate temporary objects in MariaDB for query processing can take advantage of the instance store for faster query processing. This type of workload includes queries involving sorts, hash aggregations, high-load joins, Common Table Expressions (CTEs), and queries on unindexed columns. These instance store volumes provide higher IOPS and performance, regardless of the storage configurations used for persistent Amazon EBS storage. Because RDS

Optimized Reads offloads operations on temporary objects to the instance store, the input/output operations per second (IOPS) or throughput of the persistent storage (Amazon EBS) can now be used for operations on persistent objects. These operations include regular data file reads and writes and background engine operations, such as flushing and insert buffer merges.

> **Note**
>
> Both manual and automated RDS snapshots contain only the engine files for persistent objects. The temporary objects created in the instance store aren't included in RDS snapshots.

Use cases for RDS Optimized Reads

If you have workloads that rely heavily on temporary objects, such as internal tables or files, for their query execution, then you can benefit from turning on RDS Optimized Reads. The following use cases are candidates for RDS Optimized Reads:

- Applications that run analytical queries with complex common table expressions (CTEs), derived tables, and grouping operations
- Read replicas that serve heavy read traffic with unoptimized queries
- Applications that run on-demand or dynamic reporting queries that involve complex operations, such as queries with `GROUP BY` and `ORDER BY` clauses
- Workloads that use internal temporary tables for query processing

  You can monitor the engine status variable `created_tmp_disk_tables` to determine the number of disk-based temporary tables created on your DB instance.
- Applications that create large temporary tables, either directly or in procedures, to store intermediate results
- Database queries that perform grouping or ordering on non-indexed columns

Best practices for RDS Optimized Reads

Use the following best practices for RDS Optimized Reads:

- Add retry logic for read-only queries in case they fail because the instance store is full during the execution.

- Monitor the storage space available on the instance store with the CloudWatch metric `FreeLocalStorage`. If the instance store is reaching its limit because of workload on the DB instance, modify the DB instance to use a larger DB instance class.
- When your DB instance has sufficient memory but is still reaching the storage limit on the instance store, increase the `binlog_cache_size` value to maintain the session-specific binlog entries in memory. This configuration prevents writing the binlog entries to temporary binlog cache files on disk.

  The `binlog_cache_size` parameter is session-specific. You can change the value for each new session. The setting for this parameter can increase the memory utilization on the DB instance during peak workload. Therefore, consider increasing the parameter value based on the workload pattern of your application and available memory on the DB instance.
- Use the default value of `MIXED` for the `binlog_format`. Depending on the size of the transactions, setting `binlog_format` to `ROW` can result in large binlog cache files on the instance store.
- Avoid performing bulk changes in a single transaction. These types of transactions can generate large binlog cache files on the instance store and can cause issues when the instance store is full. Consider splitting writes into multiple small transactions to minimize storage use for binlog cache files.

Using RDS Optimized Reads

When you provision an RDS for MariaDB DB instance with one of the following DB instance classes in a Single-AZ DB instance deployment or Multi-AZ DB instance deployment, the DB instance automatically uses RDS Optimized Reads.

To turn on RDS Optimized Reads, do one of the following:

- Create an RDS for MariaDB DB instance using one of these DB instance classes. For more information, see Creating an Amazon RDS DB instance.
- Modify an existing RDS for MariaDB DB instance to use one of these DB instance classes. For more information, see Modifying an Amazon RDS DB instance.

RDS Optimized Reads is available in all AWS Regions where one or more of the DB instance classes with local NVMe SSD storage are supported. For information about DB instance classes, see the section called “DB instance classes”.

DB instance class availability differs for AWS Regions. To determine whether a DB instance class is supported in a specific AWS Region, see the section called "Determining DB instance class support in AWS Regions".

If you don't want to use RDS Optimized Reads, modify your DB instance so that it doesn't use a DB instance class that supports the feature.

Monitoring DB instances that use RDS Optimized Reads

You can monitor DB instances that use RDS Optimized Reads with the following CloudWatch metrics:

- `FreeLocalStorage`
- `ReadIOPSLocalStorage`
- `ReadLatencyLocalStorage`
- `ReadThroughputLocalStorage`
- `WriteIOPSLocalStorage`
- `WriteLatencyLocalStorage`
- `WriteThroughputLocalStorage`

These metrics provide data about available instance store storage, IOPS, and throughput. For more information about these metrics, see Amazon CloudWatch instance-level metrics for Amazon RDS.

Limitations for RDS Optimized Reads

The following limitations apply to RDS Optimized Reads:

- RDS Optimized Reads is supported for the following RDS for MariaDB versions:
  - All available minor versions of 11.4 and higher major releases
  - 10.11.4 and higher 10.11 versions
  - 10.6.7 and higher 10.6 versions
  - 10.5.16 and higher 10.5 versions
  - 10.4.25 and higher 10.4 versions

  For information about RDS for MariaDB versions, see MariaDB on Amazon RDS versions.

- You can't change the location of temporary objects to persistent storage (Amazon EBS) on the DB instance classes that support RDS Optimized Reads.
- When binary logging is enabled on a DB instance, the maximum transaction size is limited by the size of the instance store. In MariaDB, any session that requires more storage than the value of `binlog_cache_size` writes transaction changes to temporary binlog cache files, which are created on the instance store.
- Transactions can fail when the instance store is full.

Improving write performance with Amazon RDS Optimized Writes for MariaDB

You can improve the performance of write transactions with RDS Optimized Writes for MariaDB. When your RDS for MariaDB database uses RDS Optimized Writes, it can achieve up to two times higher write transaction throughput.

Topics

- Overview of RDS Optimized Writes
- Using RDS Optimized Writes
- Enabling RDS Optimized Writes on an existing database
- Limitations for RDS Optimized Writes

Overview of RDS Optimized Writes

When you turn on RDS Optimized Writes, your RDS for MariaDB databases write only once when flushing data to durable storage without the need for the doublewrite buffer. The databases continue to provide ACID property protections for reliable database transactions, along with improved performance.

Relational databases, like MariaDB, provide the *ACID properties* of atomicity, consistency, isolation, and durability for reliable database transactions. To help provide these properties, MariaDB uses a data storage area called the *doublewrite buffer* that prevents partial page write errors. These errors occur when there is a hardware failure while the database is updating a page, such as in the case of a power outage. A MariaDB database can detect partial page writes and recover with a copy of the page in the doublewrite buffer. While this technique provides protection, it also results in extra write operations. For more information about the MariaDB doublewrite buffer, see InnoDB Doublewrite Buffer in the MariaDB documentation.

With RDS Optimized Writes turned on, RDS for MariaDB databases write only once when flushing data to durable storage without using the doublewrite buffer. RDS Optimized Writes is useful if you run write-heavy workloads on your RDS for MariaDB databases. Examples of databases with write-heavy workloads include ones that support digital payments, financial trading, and gaming applications.

These databases run on DB instance classes that use the AWS Nitro System. Because of the hardware configuration in these systems, the database can write 16-KiB pages directly to data files reliably and durably in one step. The AWS Nitro System makes RDS Optimized Writes possible.

You can set the new database parameter `rds.optimized_writes` to control the RDS Optimized Writes feature for RDS for MariaDB databases. Access this parameter in the DB parameter groups of RDS for MariaDB for the following versions:

- All available minor versions of 11.8 and higher major releases
- 11.4.3 and higher 11.4 versions
- 10.11.4 and higher 10.11 versions
- 10.6.10 and higher 10.6 versions

Set the parameter using the following values:

- `AUTO` – Turn on RDS Optimized Writes if the database supports it. Turn off RDS Optimized Writes if the database doesn't support it. This setting is the default.
- `OFF` – Turn off RDS Optimized Writes even if the database supports it.

If you migrate an RDS for MariaDB database that is configured to use RDS Optimized Writes to a DB instance class that doesn't support the feature, RDS automatically turns off RDS Optimized Writes for the database.

When RDS Optimized Writes is turned off, the database uses the MariaDB doublewrite buffer.

To determine whether an RDS for MariaDB database is using RDS Optimized Writes, view the current value of the `innodb_doublewrite` parameter for the database. If the database is using RDS Optimized Writes, this parameter is set to `FALSE` (`0`).

Using RDS Optimized Writes

You can turn on RDS Optimized Writes when you create an RDS for MariaDB database with the RDS console, the AWS CLI, or the RDS API. RDS Optimized Writes is turned on automatically when both of the following conditions apply during database creation:

- You specify a DB engine version and DB instance class that support RDS Optimized Writes.
  - RDS Optimized Writes is supported for the following RDS for MariaDB versions:

- All available minor versions of 11.8 and higher major releases
- 11.4.3 and higher 11.4 versions
- 10.11.4 and higher 10.11 versions
- 10.6.10 and higher 10.6 versions

For information about RDS for MariaDB versions, see MariaDB on Amazon RDS versions.

- RDS Optimized Writes is supported for RDS for MariaDB databases that use the following DB instance classes:
  - db.m7i
  - db.m7g
  - db.m6g
  - db.m6gd
  - db.m6i
  - db.m5
  - db.m5d
  - db.r7i
  - db.r7g
  - db.r6g
  - db.r6gd
  - db.r6i
  - db.r5
  - db.r5b
  - db.r5d
  - db.x2idn
  - db.x2iedn

  For information about DB instance classes, see the section called “DB instance classes”.

  DB instance class availability differs for AWS Regions. To determine whether a DB instance class is supported in a specific AWS Region, see the section called “Determining DB instance class support in AWS Regions”.

- In the parameter group associated with the database, the `rds.optimized_writes` parameter is set to AUTO. In default parameter groups, this parameter is always set to AUTO.

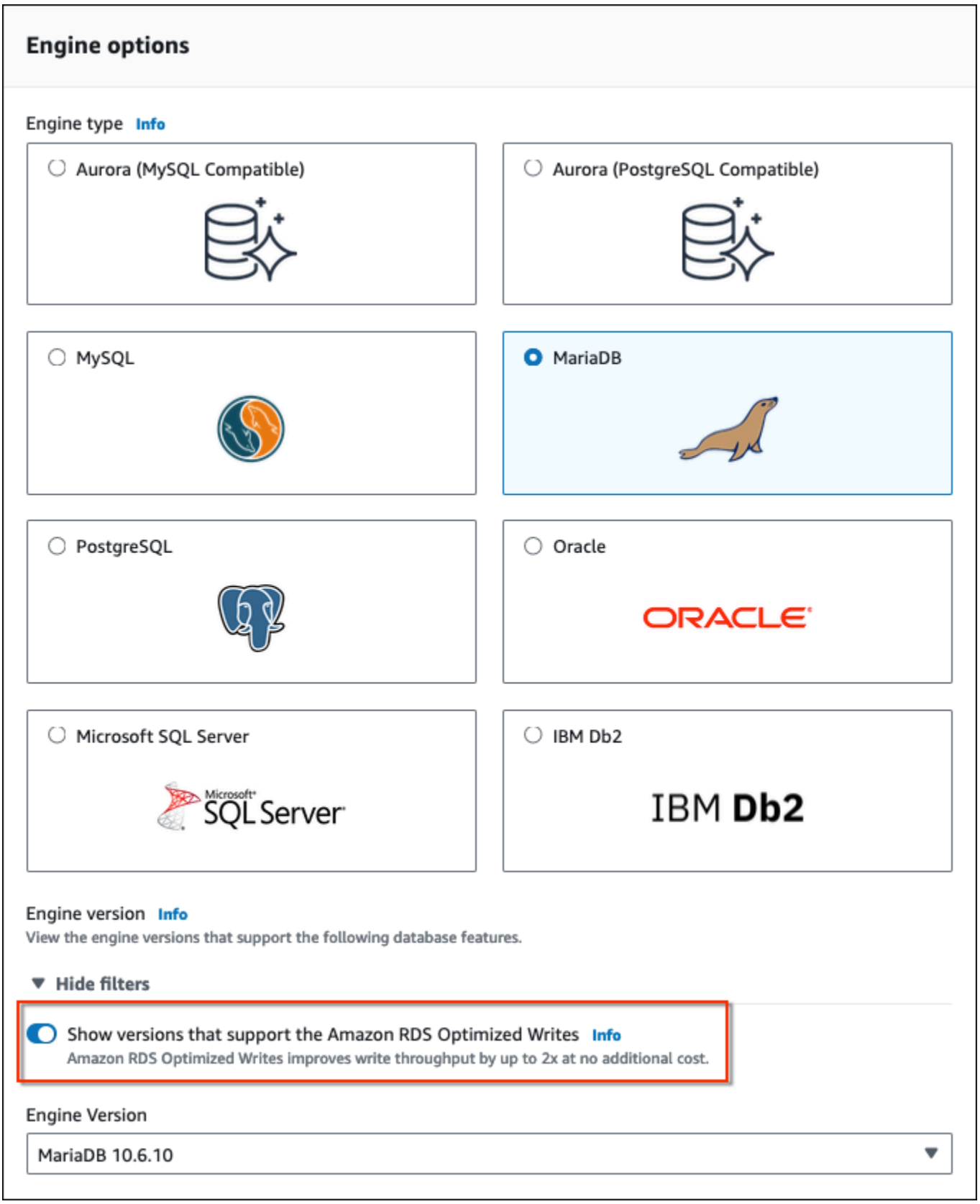
If you want to use a DB engine version and DB instance class that support RDS Optimized Writes, but you don't want to use this feature, then specify a custom parameter group when you create the database. In this parameter group, set the `rds.optimized_writes` parameter to `OFF`. If you want the database to use RDS Optimized Writes later, you can set the parameter to `AUTO` to turn it on. For information about creating custom parameter groups and setting parameters, see Parameter groups for Amazon RDS.

For information about creating a DB instance, see Creating an Amazon RDS DB instance.

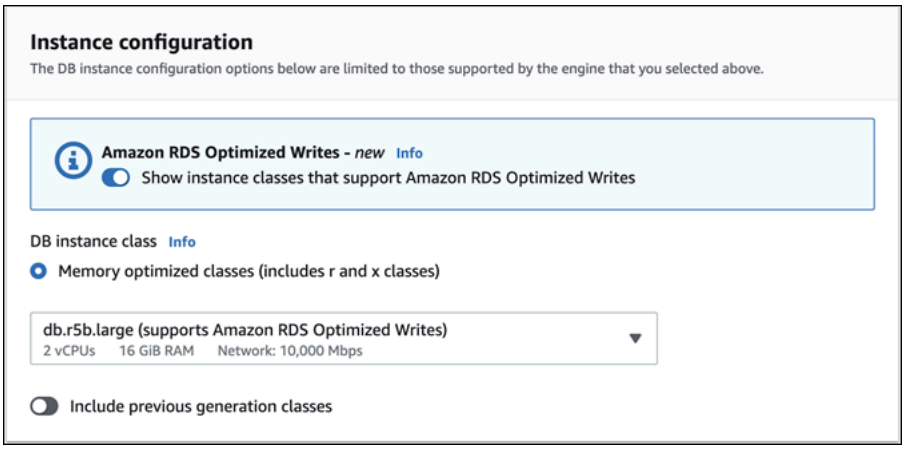
Console

When you use the RDS console to create an RDS for MariaDB database, you can filter for the DB engine versions and DB instance classes that support RDS Optimized Writes. After you turn on the filters, you can choose from the available DB engine versions and DB instance classes.

To choose a DB engine version that supports RDS Optimized Writes, filter for the RDS for MariaDB DB engine versions that support it in **Engine version**, and then choose a version.


Engine options
Engine type Info
Aurora (MySQL Compatible)
Aurora (PostgreSQL Compatible)
MySQL
MariaDB
PostgreSQL
Oracle
ORACLE
Microsoft SQL Server
Microsoft SQL Server
IBM Db2
IBM Db2
Engine version Info
View the engine versions that support the following database features.
Hide filters
Show versions that support the Amazon RDS Optimized Writes Info
Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.
Engine Version
MariaDB 10.6.10

In the **Instance configuration** section, filter for the DB instance classes that support RDS Optimized Writes, and then choose a DB instance class.



After you make these selections, you can choose other settings that meet your requirements and finish creating the RDS for MariaDB database with the console.

AWS CLI

To create a DB instance by using the AWS CLI, run the create-db-instance command. Make sure the `--engine-version` and `--db-instance-class` values support RDS Optimized Writes. In addition, make sure the parameter group associated with the DB instance has the `rds.optimized_writes` parameter set to AUTO. This example associates the default parameter group with the DB instance.

Example Creating a DB instance that uses RDS Optimized Writes

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --engine mariadb \
    --engine-version 10.6.10 \
    --db-instance-class db.r5b.large \
    --manage-master-user-password \
    --master-username admin \
    --allocated-storage 200
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
```

```
    --engine mariadb ^
    --engine-version 10.6.10 ^
    --db-instance-class db.r5b.large ^
    --manage-master-user-password ^
    --master-username admin ^
    --allocated-storage 200
```

RDS API

You can create a DB instance using the CreateDBInstance operation. When you use this operation, make sure the `EngineVersion` and `DBInstanceClass` values support RDS Optimized Writes. In addition, make sure the parameter group associated with the DB instance has the `rds.optimized_writes` parameter set to `AUTO`.

Enabling RDS Optimized Writes on an existing database

In order to modify an existing RDS for MariaDB database to turn on RDS Optimized Writes, the database must have been created with a supported DB engine version and DB instance class. In addition, the database must have been created *after* RDS Optimized Writes was released on March 7, 2023, as the required underlying file system configuration is incompatible with that of databases created before it was released. If these conditions are met, you can turn on RDS Optimized Writes by setting the `rds.optimized_writes` parameter to `AUTO`.

If your database was *not* created with a supported engine version, instance class, or file system configuration, you can use RDS Blue/Green Deployments to migrate to a supported configuration. While creating the blue/green deployment, do the following:

- Select **Enable Optimized Writes on green database**, then specify an engine version and DB instance class that supports RDS Optimized Writes. For a list of supported engine versions and instance classes, see the section called “Using with a new database”.
- Under **Storage**, choose **Upgrade storage file system configuration**. This option upgrades the database to a compatible underlying file system configuration.

When you create the blue/green deployment, if the `rds.optimized_writes` parameter is set to `AUTO`, RDS Optimized Writes will be automatically enabled on the green environment. You can then switch over the blue/green deployment, which promotes the green environment to be the new production environment.

For more information, see the section called “Creating a blue/green deployment”.

Limitations for RDS Optimized Writes

When you're restoring an RDS for MariaDB database from a snapshot, you can only turn on RDS Optimized Writes for the database if all of the following conditions apply:

- The snapshot was created from a database that supports RDS Optimized Writes.
- The snapshot was created from a database that was created *after* RDS Optimized Writes was released.
- The snapshot is restored to a database that supports RDS Optimized Writes.
- The restored database is associated with a parameter group that has the `rds.optimized_writes` parameter set to `AUTO`.

Upgrades of the MariaDB DB engine

When Amazon RDS supports a new version of a database engine, you can upgrade your DB instances to the new version. There are two kinds of upgrades for MariaDB DB instances: major version upgrades and minor version upgrades.

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, you must manually perform major version upgrades of your DB instances. You can initiate a major version upgrade by modifying your DB instance. However, before you perform a major version upgrade, we recommend that you follow the instructions in Major version upgrades for RDS for MariaDB.

In contrast, *minor version upgrades* include only changes that are backward-compatible with existing applications. You can initiate a minor version upgrade manually by modifying your DB instance. Or you can enable the **Auto minor version upgrade** option when creating or modifying a DB instance. Doing so means that your DB instance is automatically upgraded after Amazon RDS tests and approves the new version. For information about performing an upgrade, see Upgrading a DB instance engine version.

If your MariaDB DB instance is using read replicas, you must upgrade all of the read replicas before upgrading the source instance. If your DB instance is in a Multi-AZ deployment, both the writer and standby replicas are upgraded. Your DB instance might not be available until the upgrade is complete.

For more information about MariaDB supported versions and version management, see MariaDB on Amazon RDS versions.

Database engine upgrades require downtime. The duration of the downtime varies based on the size of your DB instance.

Amazon RDS also supports upgrade rollout policy to manage automatic minor version upgrades across multiple database resources and AWS accounts. For more information, see Using AWS Organizations upgrade rollout policy for automatic minor version upgrades.

> **Tip**
>
> You can minimize the downtime required for DB instance upgrade by using a blue/green deployment. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

Topics

- Considerations for MariaDB upgrades
- Finding valid upgrade targets
- MariaDB version numbers
- RDS version numbers in RDS for MariaDB
- Major version upgrades for RDS for MariaDB
- Upgrading a MariaDB DB instance
- Automatic minor version upgrades for RDS for MariaDB
- Using a read replica to reduce downtime when upgrading an RDS for MariaDB database
- Monitoring RDS for MariaDB DB engine upgrades with events

Considerations for MariaDB upgrades

Amazon RDS takes two or more DB snapshots during the upgrade process. Amazon RDS takes up to two snapshots of the DB instance *before* making any upgrade changes. If the upgrade doesn't work for your databases, you can restore one of these snapshots to create a DB instance running the old version. Amazon RDS takes another snapshot of the DB instance when the upgrade completes. Amazon RDS takes these snapshots regardless of whether AWS Backup manages the backups for the DB instance.

> **Note**
>
> Amazon RDS only takes DB snapshots if you have set the backup retention period for your DB instance to a number greater than 0. To change your backup retention period, see Modifying an Amazon RDS DB instance.

After the upgrade is complete, you can't revert to the previous version of the database engine. If you want to return to the previous version, restore the first DB snapshot taken to create a new DB instance.

You control when to upgrade your DB instance to a new version supported by Amazon RDS. This level of control helps you maintain compatibility with specific database versions and test new versions with your application before deploying in production. When you are ready, you can perform version upgrades at the times that best fit your schedule.

If your DB instance is using read replication, you must upgrade all of the Read Replicas before upgrading the source instance.

If your DB instance is in a Multi-AZ deployment, both the primary and standby DB instances are upgraded. The primary and standby DB instances are upgraded at the same time and you will experience an outage until the upgrade is complete. The time for the outage varies based on your database engine, engine version, and the size of your DB instance.

Finding valid upgrade targets

When you use the AWS Management Console to upgrade a DB instance, it shows the valid upgrade targets for the DB instance. You can also run the following AWS CLI command to identify the valid upgrade targets for a DB instance:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
  --engine mariadb \
  --engine-version version_number \
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
  --engine mariadb ^
  --engine-version version_number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For example, to identify the valid upgrade targets for a MariaDB version 10.5.17 DB instance, run the following AWS CLI command:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
  --engine mariadb \
  --engine-version 10.5.17 \
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
  --engine mariadb ^
  --engine-version 10.5.17 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

MariaDB version numbers

The version numbering sequence for the RDS for MariaDB database engine is either in the form of *major.minor.patch.YYYYMMDD* or *major.minor.patch*, for example, 10.11.5.R2.20231201 or 10.4.30. The format used depends on the MariaDB engine version.

major

The major version number is both the integer and the first fractional part of the version number, for example, 10.11. A major version upgrade increases the major part of the version number. For example, an upgrade from *10.5*.20 to 10.6.12 is a major version upgrade, where *10.5* and *10.6* are the major version numbers.

minor

The minor version number is the third part of the version number, for example, the 5 in 10.11.5.

patch

The patch is the fourth part of the version number, for example, the R2 in 10.11.5.R2. An RDS patch version includes important bug fixes added to a minor version after its release.

YYYYMMDD

The date is the fifth part of the version number, for example, the 20231201 in 10.11.5.R2.20231201. An RDS date version is a security patch that includes important security fixes added to a minor version after its release. It doesn't include any fixes that might change an engine's behavior.

The following table explains the naming scheme for RDS for MariaDB version 10.11.

10.11 minor version	Naming scheme
≥5	New DB instances use *major.minor.patch.YYMMDD*, for example, 10.11.5.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 10.11.5.R2, until your next major or minor version upgrade.
< 5	Existing DB instances use *major.minor.patch*, for example, 10.11.4.R2.

The following table explains the naming scheme for RDS for MariaDB version 10.6.

10.6 minor version	Naming scheme
≥ 14	New DB instances use *major.minor.patch.YYMMDD*, for example, 10.6.14.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 10.6.14.R2, until your next major or minor version upgrade.
< 14	Existing DB instances use *major.minor.patch*, for example, 10.6.13.R2.

The following table explains the naming scheme for RDS for MariaDB version 10.5.

10.5 minor version	Naming scheme
≥ 21	New DB instances use *major.minor.patch.YYMMDD*, for example, 10.5.21.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 10.5.21.R2, until your next major or minor version upgrade.

10.5 minor version	Naming scheme
< 21	Existing DB instances use *major.minor.patch*, for example, 10.5.20.R2.

The following table explains the naming scheme for RDS for MariaDB version 10.4.

10.4 minor version	Naming scheme
≥ 30	New DB instances use *major.minor.patch.YYMMDD*, for example, 10.4.30.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 10.4.30.R2, until your next major or minor version upgrade.
< 30	Existing DB instances use *major.minor.patch*, for example, 10.4.29.R2.

RDS version numbers in RDS for MariaDB

RDS version numbers use either the *major*.*minor*.*patch* or the *major*.*minor*.*patch*.*YYYYMMDD* naming scheme. An RDS patch version includes important bug fixes added to a minor version after its release. An RDS date version (*YYMMDD*) is a security patch. A security patch doesn't include any fixes that might change the engine's behavior.

To identify the Amazon RDS version number of your database, you must first create the `rds_tools` extension by using the following command:

```
CREATE EXTENSION rds_tools;
```

You can find out the RDS version number of your RDS for MariaDB database with the following SQL query:

```
mysql> select mysql.rds_version();
```

For example, querying an RDS for MariaDB 10.6.14 database returns the following output:

```
+---------------------+
| mysql.rds_version() |
+---------------------+
| 10.6.14.R2.20231201  |
+---------------------+
1 row in set (0.01 sec)
```

Major version upgrades for RDS for MariaDB

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, Amazon RDS doesn't apply major version upgrades automatically. You must manually modify your DB instance. We recommend that you thoroughly test any upgrade before applying it to your production instances.

> **Note**
>
> In MariaDB 11.8, the default value for `require_secure_transport` is now 1, requiring secure SSL/TLS connections. Set to `0` if non-secure connections are needed.

Amazon RDS supports the following in-place upgrades for major versions of the MariaDB database engine:

- Any MariaDB version to MariaDB 11.8
- Any MariaDB version to MariaDB 11.4
- Any MariaDB version to MariaDB 10.11
- Any MariaDB version to MariaDB 10.6
- MariaDB 10.4 to MariaDB 10.5

If you are using a custom parameter group, and you perform a major version upgrade, you must specify either a default parameter group for the new DB engine version or create your own custom parameter group for the new DB engine version. Associating the new parameter group with the DB instance requires a customer-initiated database reboot after the upgrade completes. The instance's parameter group status will show `pending-reboot` if the instance needs to be rebooted to apply the parameter group changes. An instance's parameter group status can be viewed in the AWS Management Console or by running a "describe" call such as `describe-db-instances`.

Upgrading a MariaDB DB instance

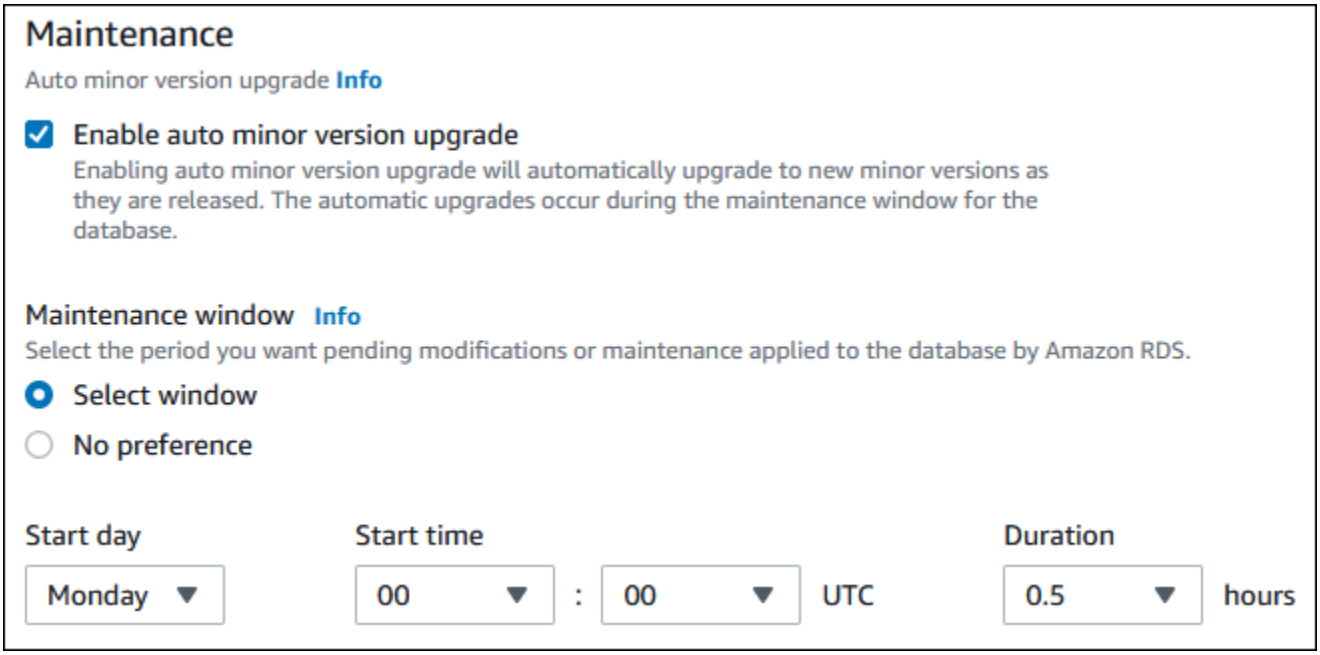
For information about manually or automatically upgrading a MariaDB DB instance, see Upgrading a DB instance engine version.

Automatic minor version upgrades for RDS for MariaDB

If you specify the following settings when creating or modifying a DB instance, you can have your DB instance automatically upgraded.

- The **Auto minor version upgrade** setting is enabled.
- The **Backup retention period** setting is greater than 0.

In the AWS Management Console, these settings are under **Additional configuration**. The following image shows the **Auto minor version upgrade** setting.



For more information about these settings, see Settings for DB instances.

For some RDS for MariaDB major versions in some AWS Regions, one minor version is designated by RDS as the automatic upgrade version. After a minor version has been tested and approved by Amazon RDS, the minor version upgrade occurs automatically during your maintenance window. RDS doesn't automatically set newer released minor versions as the automatic upgrade version. Before RDS designates a newer automatic upgrade version, several criteria are considered, such as the following:

- Known security issues
- Bugs in the MariaDB community version
- Overall fleet stability since the minor version was released

> **Note**
>
> Support for using TLS version 1.0 and 1.1 was removed starting with specific minor versions of MariaDB. For information about supported MariaDB minor versions, see the section called "SSL/TLS support for MariaDB".

You can run the following AWS CLI command to determine the current automatic minor upgrade target version for a specified MariaDB minor version in a specific AWS Region.

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mariadb \
--engine-version minor_version \
--region region \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mariadb ^
--engine-version minor_version ^
--region region ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output text
```

For example, the following AWS CLI command determines the automatic minor upgrade target for MariaDB minor version 10.5.16 in the US East (Ohio) AWS Region (us-east-2).

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
```

```
--engine mariadb \
--engine-version 10.5.16 \
--region us-east-2 \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output table
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mariadb ^
--engine-version 10.5.16 ^
--region us-east-2 ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output table
```

Your output is similar to the following.

```
---------------------------------
|    DescribeDBEngineVersions    |
+--------------+-----------------+
|  AutoUpgrade |  EngineVersion  |
+--------------+-----------------+
|  True        |  10.5.17        |
|  False       |  10.5.18        |
|  False       |  10.5.19        |
|  False       |  10.6.5         |
|  False       |  10.6.7         |
|  False       |  10.6.8         |
|  False       |  10.6.10        |
|  False       |  10.6.11        |
|  False       |  10.6.12        |
+--------------+-----------------+
```

In this example, the `AutoUpgrade` value is `True` for MariaDB version 10.5.17. So, the automatic minor upgrade target is MariaDB version 10.5.17, which is highlighted in the output.

A MariaDB DB instance is automatically upgraded during your maintenance window if the following criteria are met:

- The **Auto minor version upgrade** setting is enabled.

- The **Backup retention period** setting is greater than 0.
- The DB instance is running a minor DB engine version that is less than the current automatic upgrade minor version.

For more information, see Automatically upgrading the minor engine version.

Using a read replica to reduce downtime when upgrading an RDS for MariaDB database

In most cases, a blue/green deployment is the best option to reduce downtime when upgrading a MariaDB DB instance. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

If you can't use a blue/green deployment and your MariaDB DB instance is currently in use with a production application, you can use the following procedure to upgrade the database version for your DB instance. This procedure can reduce the amount of downtime for your application.

By using a read replica, you can perform most of the maintenance steps ahead of time and minimize the necessary changes during the actual outage. With this technique, you can test and prepare the new DB instance without making any changes to your existing DB instance.

The following procedure shows an example of upgrading from MariaDB version 10.5 to MariaDB version 10.6. You can use the same general steps for upgrades to other major versions.

To upgrade a MariaDB database while a DB instance is in use

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Create a read replica of your MariaDB 10.5 DB instance. This process creates an upgradable copy of your database. Other read replicas of the DB instance might also exist.
    a. In the console, choose **Databases**, and then choose the DB instance that you want to upgrade.
    b. For **Actions**, choose **Create read replica**.
    c. Provide a value for **DB instance identifier** for your read replica and ensure that the **DB instance class** and other settings match your MariaDB 10.5 DB instance.
    d. Choose **Create read replica**.

3. (Optional) When the read replica has been created and **Status** shows **Available**, convert the read replica into a Multi-AZ deployment and enable backups.

   By default, a read replica is created as a Single-AZ deployment with backups disabled. Because the read replica ultimately becomes the production DB instance, it is a best practice to configure a Multi-AZ deployment and enable backups now.

   a. In the console, choose **Databases**, and then choose the read replica that you just created.

   b. Choose **Modify**.

   c. For **Multi-AZ deployment**, choose **Create a standby instance**.

   d. For **Backup Retention Period**, choose a positive nonzero value, such as 3 days, and then choose **Continue**.

   e. For **Scheduling of modifications**, choose **Apply immediately**.

   f. Choose **Modify DB instance**.

4. When the read replica **Status** shows **Available**, upgrade the read replica to MariaDB 10.6.

   a. In the console, choose **Databases**, and then choose the read replica that you just created.

   b. Choose **Modify**.

   c. For **DB engine version**, choose the MariaDB 10.6 version to upgrade to, and then choose **Continue**.

   d. For **Scheduling of modifications**, choose **Apply immediately**.

   e. Choose **Modify DB instance** to start the upgrade.

5. When the upgrade is complete and **Status** shows **Available**, verify that the upgraded read replica is up-to-date with the source MariaDB 10.5 DB instance. To verify, connect to the read replica and run the `SHOW REPLICA STATUS` command. If the `Seconds_Behind_Master` field is `0`, then replication is up-to-date.

   > **Note**
   >
   > Previous versions of MariaDB used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MariaDB version before 10.6, then use `SHOW SLAVE STATUS`.

6. (Optional) Create a read replica of your read replica.

If you want the DB instance to have a read replica after it is promoted to a standalone DB instance, you can create the read replica now.

a. In the console, choose **Databases**, and then choose the read replica that you just upgraded.

b. For **Actions**, choose **Create read replica**.

c. Provide a value for **DB instance identifier** for your read replica and ensure that the **DB instance class** and other settings match your MariaDB 10.5 DB instance.

d. Choose **Create read replica**.

7. (Optional) Configure a custom DB parameter group for the read replica.

   If you want the DB instance to use a custom parameter group after it is promoted to a standalone DB instance, you can create the DB parameter group now and associate it with the read replica.

   a. Create a custom DB parameter group for MariaDB 10.6. For instructions, see Creating a DB parameter group in Amazon RDS.

   b. Modify the parameters that you want to change in the DB parameter group you just created. For instructions, see Modifying parameters in a DB parameter group in Amazon RDS.

   c. In the console, choose **Databases**, and then choose the read replica.

   d. Choose **Modify**.

   e. For **DB parameter group**, choose the MariaDB 10.6 DB parameter group you just created, and then choose **Continue**.

   f. For **Scheduling of modifications**, choose **Apply immediately**.

   g. Choose **Modify DB instance** to start the upgrade.

8. Make your MariaDB 10.6 read replica a standalone DB instance.

   Important

   When you promote your MariaDB 10.6 read replica to a standalone DB instance, it is no longer a replica of your MariaDB 10.5 DB instance. We recommend that you promote your MariaDB 10.6 read replica during a maintenance window when your source MariaDB 10.5 DB instance is in read-only mode and all write operations are

suspended. When the promotion is completed, you can direct your write operations to the upgraded MariaDB 10.6 DB instance to ensure that no write operations are lost. In addition, we recommend that, before promoting your MariaDB 10.6 read replica, you perform all necessary data definition language (DDL) operations on your MariaDB 10.6 read replica. An example is creating indexes. This approach avoids negative effects on the performance of the MariaDB 10.6 read replica after it has been promoted. To promote a read replica, use the following procedure.

a. In the console, choose **Databases**, and then choose the read replica that you just upgraded.
b. For **Actions**, choose **Promote**.
c. Choose **Yes** to enable automated backups for the read replica instance. For more information, see Introduction to backups.
d. Choose **Continue**.
e. Choose **Promote Read Replica**.

9. You now have an upgraded version of your MariaDB database. At this point, you can direct your applications to the new MariaDB 10.6 DB instance.

Monitoring RDS for MariaDB DB engine upgrades with events

When you upgrade the engine version of an RDS for MariaDB database, Amazon RDS emits a specific event during each phase of the process. To track the progress of an upgrade, you can view or subscribe to these events.

For more information about RDS events, see Monitoring Amazon RDS events.

For detailed information about a specific Amazon RDS event that occurs during your engine upgrade, see Amazon RDS event categories and event messages.

Upgrading a MariaDB DB snapshot engine version

With Amazon RDS, you can create a storage volume DB snapshot of your MariaDB DB instance. When you create a DB snapshot, the snapshot is based on the engine version used by your DB instance. You can upgrade the engine version for your DB snapshots.

For RDS for MariaDB, you can upgrade to all available engine versions. You can upgrade encrypted or unencrypted DB snapshots.

To view the available engine versions for your RDS for MariaDB DB snapshot, use the following AWS CLI example.

```
aws rds describe-db-engine-versions --engine mariadb --include-all --engine-
version example-engine-version --query "DBEngineVersions[*].ValidUpgradeTarget[*].
{EngineVersion:EngineVersion}" --output text
```

If you don't see results for your snapshot, your engine version might be deprecated. If your engine version is deprecated, we recommend that you upgrade to the newest major version upgrade target or to one of the other available upgrade targets for that version. For more information, see Upgrade options for DB snapshots with unsupported engine versions for RDS for MariaDB.

After restoring a DB snapshot upgraded to a new engine version, make sure to test that the upgrade was successful. For more information about a major version upgrade, see the section called “Upgrades of the MariaDB DB engine”. To learn how to restore a DB snapshot, see the section called “Restoring to a DB instance”.

> **Note**
>
> You can't upgrade automated DB snapshots that were created during the automated backup process.

You can upgrade a DB snapshot using the AWS Management Console, AWS CLI, or RDS API.

Console

To upgrade a DB snapshot engine version using the AWS Management Console, use the following procedure.

To upgrade a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the snapshot that you want to upgrade.
4. For **Actions**, choose **Upgrade snapshot**. The **Upgrade snapshot** page appears.
5. Choose the **New engine version** to upgrade to.
6. Choose **Save changes** to upgrade the snapshot.

   During the upgrade process, all snapshot actions are disabled for this DB snapshot. Also, the DB snapshot status changes from **Available** to **Upgrading**, and then changes to **Active** upon completion. If the DB snapshot can't be upgraded because of snapshot corruption issues, the status changes to **Unavailable**. You can't recover the snapshot from this state.

   > **Note**
   >
   > If the DB snapshot upgrade fails, the snapshot is rolled back to the original state with the original version.

AWS CLI

To upgrade a DB snapshot to a new database engine version, run the AWS CLI modify-db-snapshot command.

Options

- `--db-snapshot-identifier` – The identifier of the DB snapshot to upgrade. The identifier must be a unique Amazon Resource Name (ARN). For more information, see Amazon Resource Names (ARNs) in Amazon RDS.
- `--engine-version` – The engine version to upgrade the DB snapshot to.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-snapshot \
```

```
    --db-snapshot-identifier my_db_snapshot \
    --engine-version new_version
```

For Windows:

```
aws rds modify-db-snapshot ^
    --db-snapshot-identifier my_db_snapshot ^
    --engine-version new_version
```

Amazon RDS API

To upgrade a DB snapshot to a new database engine version, call the RDS API ModifyDBSnapshot operation.

Parameters

- `DBSnapshotIdentifier` – The identifier of the DB snapshot to upgrade. The identifier must be a unique Amazon Resource Name (ARN). For more information, see Amazon Resource Names (ARNs) in Amazon RDS.
- `EngineVersion` – The engine version to upgrade the DB snapshot to.

Upgrade options for DB snapshots with unsupported engine versions for RDS for MariaDB

The following table shows which engine versions you can upgrade to from an unsupported engine version for RDS for MariaDB DB snapshots.

> **Note**
>
> You might have to upgrade your DB snapshot more than once to upgrade to your chosen engine version.

DB snapshot engine version	Engine versions available for upgrade
10.0.17, 10.0.24, 10.0.28, 10.0.31, 10.0.32, 10.0.34, 10.0.35	10.0.38, 10.1.48, 10.6.19, 10.6.20, 10.11.9, 10.11.10, 11.4.3, 11.4.4, 11.8.3
10.1.14, 10.1.16, 10.1.19, 10.1.23, 10.1.26, 10.1.31, 10.1.34	10.1.48, 10.2.44, 10.6.19, 10.6.20, 10.11.9, 10.11.10, 11.4.3, 11.4.4, 11.8.3
10.2.11, 10.2.12, 10.2.15, 10.2.21, 10.2.32, 10.2.37, 10.2.39, 10.2.40, 10.2.41, 10.2.43	10.2.44, 10.3.39, 10.6.19, 10.6.20, 10.6.21, 10.11.9, 10.11.10, 10.11.11, 11.4.3, 11.4.4, 11.8.3
10.3.8, 10.3.13, 10.3.20, 10.3.23, 10.3.28, 10.3.31, 10.3.32	10.3.39, 10.4.34, 10.6.19, 10.6.20 10.6.21, 10.11.9, 10.11.10, 10.11.11, 11.4.3, 11.4.4, 11.8.3
10.4.8	10.4.34, 10.5.16, 10.5.17, 10.5.18, 10.5.20, 10.5.21, 10.5.22, 10.5.23, 10.5.24, 10.5.25, 10.5.26, 10.5.27, 10.5.28, 10.6.8, 10.6.10, 10.6.11, 10.6.13, 10.6.14, 10.6.15, 10.6.16, 10.6.18, 10.6.19, 10.6.20, 10.6.21, 10.11.4, 10.6.17, 10.11.5, 10.11.6, 10.11.7, 10.11.8, 10.11.9, 10.11.10, 10.11.11

Importing data into an Amazon RDS for MariaDB DB instance

You can use several different techniques to import data into an RDS for MariaDB DB instance. The best approach depends on a number of factors:

- Source of the data
- Amount of data
- One-time import or ongoing
- Amount of downtime

If you are also migrating an application with the data, the amount of downtime is important to consider.

The following table lists techniques to importing data into an RDS for MariaDB DB instance:

> **Note**
>
> Amazon RDS doesn't support `mariadb-backup` or importing from Amazon S3 for MariaDB.

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
Existing MariaDB database on premises or on Amazon EC2	Any	Ongoing	Minimal	Configure replication with an existing MariaDB database as the replication source. You can configure replication into a MariaDB DB instance using MariaDB global transaction identifiers (GTIDs) when the external instance is MariaDB version 10.0.24 or higher, or using binary log coordinates for MariaDB instances on earlier versions than 10.0.24. MariaDB GTIDs are implemented differently than	Configuri ng binary log file position replicati on with an external source instance

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
				MySQL GTIDs, which aren't supported by Amazon RDS.	Importing data to an Amazon RDS for MariaDB DB instance with reduced downtime

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
Any existing database	Any	One time or ongoing	Minimal	Use AWS Database Migration Service to migrate the database with minimal downtime and, for many database DB engines, continue ongoing replication.	What is AWS Database Migration Service and Using a MySQL-com patible database as a target for AWS DMS in the *AWS Database Migration Service User Guide*
Existing MariaDB DB instance	Any	One time or ongoing	Minimal	Create a read replica for ongoing replication. Promote the read replica for one-time creation of a new DB instance.	Working with DB instance read replicas

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
Existing MariaDB database	Small	One time	Some	Copy the data directly to your MariaDB DB instance using a command-line utility.	Importing data from an external MariaDB database to an Amazon RDS for MariaDB DB instance
Data not stored in an existing database	Medium	One time	Some	Create flat files and import them using MariaDB `LOAD DATA LOCAL INFILE` statements.	Importing data from any source to an Amazon RDS for MariaDB DB instance

Note

The `mysql` system database contains authentication and authorization information required to log in to your DB instance and access your data. Dropping, altering, renaming, or truncating tables, data, or other contents of the `mysql` database in your DB instance can result in errors and might render the DB instance and your data inaccessible. If this occurs, you can restore the DB instance from a snapshot using the AWS CLI restore-db-instance-

from-db-snapshot command. You can recover the DB instance using restore-db-instance-to-point-in-time command.

Importing data considerations for MariaDB

The following content contains technical information related to loading data into MariaDB. This content is aimed at users who are familiar with the MariaDB server architecture.

Binary logging

Enabling binary logging reduces data load performance and requires up to four times additional disk space compared to disabled logging. The transaction size used to load the data directly affects system performance and disk space needs—larger transactions require more resources.

Transaction size

Transaction size influences the following aspects of MariaDB data loads:

- Resource consumption
- Disk space utilization
- Resume process
- Time to recover
- Input format (flat files or SQL)

This section describes how transaction size affects binary logging and makes the case for disabling binary logging during large data loads. You can enable and disable binary logging by setting the Amazon RDS automated backup retention period. Non-zero values enable binary logging, and zero disables it. For more information, see Backup retention period.

This section also describes the impact of large transactions on InnoDB and why it's important to keep transaction sizes small.

Small transactions

For small transactions, binary logging doubles the number of disk writes required to load the data. This effect can severely degrade performance for other database sessions and increase the time required to load the data. The degradation experienced depends in part on the following factors:

- Upload rate
- Other database activity taking place during the load
- Capacity of your Amazon RDS DB instance

The binary logs also consume disk space roughly equal to the amount of data loaded until the logs are backed up and removed. Amazon RDS minimizes this by frequently backing up and removing binary logs.

Large transactions

For large transactions, binary logging triples IOPS and disk usage for the following reasons:

- The binary log cache stores transaction data temporarily on disk.
- This cache grows with the transaction size, which consumes disk space.
- When the transaction (commit or rollback) completes, the system copies the cache to the binary log.

This process creates three copies of the data:

- The original data
- The cache on disk
- The final binary log entry

Each write operation incurs additional IO, further impacting performance.

Because of this, binary logging requires triple the disk space compared to disabled logging. For example, loading 10 GiB of data as a single transaction creates three copies:

- 10 GiB for the table data
- 10 GiB for the binary log cache
- 10 GiB for the binary log file

The total temporary disk space required is 30 GiB.

Important disk space considerations:

- The cache file persists until either the session ends or a new transaction creates another cache.
- The binary log remains until it's backed up, potentially holding 20 GiB (cache and log) for an extended period.

If you use `LOAD DATA LOCAL INFILE` to load the data, data recovery creates a fourth copy in case the database has to be recovered from a backup made before the load. During recovery, MariaDB extracts the data from the binary log into a flat file. MariaDB then runs `LOAD DATA LOCAL INFILE`. Building on the preceding example, this recovery requires a total temporary disk space of 40 GiB, or 10 GiB each for table, cache, log, and local file. Without at least 40 GiB of free disk space, recovery fails.

Optimizing large data loads

For large data loads, disable binary logging to reduce overhead and disk space requirements. You can disable binary logging by setting the backup retention period to 0. After loading completes, restore the backup retention period to the appropriate non-zero value. For more information, see Modifying an Amazon RDS DB instance and Backup retention period in the settings table.

> **Note**
>
> If the DB instance is a source DB instance for read replicas, then you can't set the backup retention period to 0.

Before loading the data, we recommend that you create a DB snapshot. For more information, see Managing manual backups.

InnoDB

The following information about undo logging and recovery options supports keeping InnoDB transactions small to optimize database performance.

Understanding InnoDB undo logging

Undo is a logging mechanism that enables transaction rollback and supports multi-version concurrency control (MVCC).

For MariaDB 10.11 and lower versions, undo logs are stored in the InnoDB system tablespace (usually ibdata1) and are retained until the purge thread removes them. As a result, large data load

transactions can cause the system tablespace to become quite large and consume disk space that you can't reclaim unless you recreate the database.

For all MariaDB versions, the purge thread must wait to remove any undo logs until the oldest active transaction either commits or rolls back. If the database is processing other transactions during the load, their undo logs also accumulate and can't be removed, even if the transactions commit and no other transaction needs the undo logs for MVCC. In this situation, all transactions—including read-only transactions—slow down. This slowdown occurs because all transactions access all rows that any transaction—not just the load transaction—changes. In effect, transactions must scan through undo logs that long-running load transactions prevented from being purged during an undo log cleanup. This affects performance for any operation accessing modified rows.

InnoDB transaction recovery options

Although InnoDB optimizes commit operations, large transaction rollbacks are slow. For faster recovery, perform a point-in-time recovery or restore a DB snapshot. For more information, see Point-in-time recovery and Restoring to a DB instance.

Data import formats

MariaDB supports two data import formats: flat files and SQL. Review the information about each format to determine the best option for your needs.

Flat files

For small transactions, load flat files with `LOAD DATA LOCAL INFILE`. This data import format can provide the following benefits over using SQL:

- Less network traffic
- Lower data transmission costs
- Decreased database processing overhead
- Faster processing

`LOAD DATA LOCAL INFILE` loads the entire flat file as one transaction. Keep the size of the individual files small for the following advantages:

- **Resume capability** – You can keep track of which files have been loaded. If a problem arises during the load, you can pick up where you left off. You might need to retransmit some data to Amazon RDS, but with small files, the amount retransmitted is minimal.

- **Parallel data loading** – If you have sufficient IOPS and network bandwidth for a single file load, loading in parallel could save time.
- **Load rate control** – If your data load has a negative impact on other processes, you can control the load rate by increasing the interval between files.

Large transactions reduce the benefits of using `LOAD DATA LOCAL INFILE` to import data. When you can't break a large amount of data into smaller files, consider using SQL.

SQL

SQL has one main advantage over flat files: you can easily keep transaction sizes small. However, SQL can take significantly longer to load than flat files. Also, after a failure, it can be difficult to determine where to resume—you can't restart mariadb-dump files. If a failure occurs while loading mariadb-dump file, you must modify or replace the file before the load can resume. Or, alternatively, after you correct the cause of the failure, you can restore to the point in time before the load and resend the file. For more information, see Point-in-time recovery.

Using Amazon RDS DB snapshots for database checkpoints

If you load data over long durations—such as hours or days—without binary logging, use DB snapshots to provide periodic checkpoints for data safety. Each DB snapshot creates a consistent copy of your database instance that serves as a recovery point during system failures or data corruption events. Because DB snapshots are fast, frequent checkpointing has minimal impact on load performance. You can delete previous DB snapshots without impacting database durability or recovery capabilities. For more information about DB snapshots, see Managing manual backups.

Reducing database load times

The following items are additional tips to reduce load times:

- Create all secondary indexes before loading data into MariaDB databases. Unlike other database systems, MariaDB rebuilds the entire table when adding or modifying secondary indexes. This process creates a new table with index changes, copies all data, and drops the original table.
- Load data in primary key order. For InnoDB tables, this can reduce load times by 75%–80% and reduce data file size by 50%.
- Disable foreign key constraints by setting `foreign_key_checks` to `0`. This is often required for flat files loaded with `LOAD DATA LOCAL INFILE`. For any load, disabling foreign key

checks accelerates data loading. After loading completes, re-enable constraints by setting `foreign_key_checks` to 1 and verify the data.

- Load data in parallel unless approaching a resource limit. To enable concurrent loading across multiple table segments, use partitioned tables when appropriate.
- To reduce SQL execution overhead, combine multiple `INSERT` statements into single multi-value `INSERT` operations. `mariadb-dump` implements this optimization automatically.
- Reduce InnoDB log IO operations by setting `innodb_flush_log_at_trx_commit` to `0`. After loading completes, restore `innodb_flush_log_at_trx_commit` to 1.

> **Warning**
>
> Setting `innodb_flush_log_at_trx_commit` to `0` causes InnoDB to flush its logs every second instead of at each commit. This setting increases performance but can risk transaction loss during system failures.

- If you are loading data into a DB instance that doesn't have read replicas, set `sync_binlog` to `0`. After loading completes, restore `sync_binlog parameter`to 1.
- Load data into a Single-AZ DB instance before converting the DB instance to a Multi-AZ deployment. If the DB instance already uses a Multi-AZ deployment, we don't recommend switching to a Single-AZ deployment for data loading. Doing so only provides marginal improvements

Importing data from an external MariaDB database to an Amazon RDS for MariaDB DB instance

You can import data from an existing MariaDB database to an RDS for MariaDB DB instance. You do so by copying the database with mysqldump or mariadb-dump, and piping the database directly into the RDS for MariaDB DB instance. The `mysqldump` or `mariadb-dump` command line utility is commonly used to make backups and transfer data from one MariaDB server to another. It's included with MariaDB client software.

Starting with MariaDB 11.0.1, you must use `mariadb-dump` instead of `mysqldump`.

A typical `mysqldump` command to move data from an external database to an Amazon RDS DB instance looks similar to the following example. Replace values with your own information. For MariaDB 11.0.1 and higher versions, replace `mysqldump` with `mariadb-dump`.

```
mysqldump -u local_user \
    --databases database_name \
    --single-transaction \
    --compress \
    --order-by-primary  \
    --routines=0 \
    --triggers=0 \
    --events=0 \
    -plocal_password | mariadb -u RDS_user \
        --port=port_number \
        --host=host_name \
        -pRDS_password
```

Important

Make sure not to leave a space between the `-p` option and the entered password. As a security best practice, specify credentials other than the prompts shown in this example.

Make sure that you're aware of the following recommendations and considerations:

- Exclude the following schemas from the dump file:
  - `sys`
  - `performance_schema`
  - `information_schema`

  The `mysqldump` and `mariadb-dump` utility excludes these schemas by default.
- If you need to migrate users and privileges, consider using a tool that generates the data control language (DCL) for recreating them, such as the pt-show-grants utility.
- To perform the import, make sure the user doing so has access to the DB instance. For more information, see Controlling access with security groups.

The parameters used are as follows:

- `-u local_user` – Use to specify a user name. In the first usage of this parameter, specify the name of a user account on the local MariaDB database that you identify with the `--databases` parameter.

- `--databases` *database_name* – Use to specify the name of the database on the local MariaDB instance that you want to import into Amazon RDS.
- `--single-transaction` – Use to ensure that all of the data loaded from the local database is consistent with a single point in time. If there are other processes changing the data while `mysqldump` or `mariadb-dump` is reading it, using this parameter helps maintain data integrity.
- `--compress` – Use to reduce network bandwidth consumption by compressing the data from the local database before sending it to Amazon RDS.
- `--order-by-primary` – Use to reduce load time by sorting each table's data by its primary key.
- `--routines` – Use if routines such as stored procedures or functions exist in the database that you are copying. Set the parameter to `0`, which excludes the routines during the import process. Then later manually recreate the routines in the Amazon RDS database.
- `--triggers` – Use if triggers exist in the database that you are copying. Set the parameter to `0`, which excludes the triggers during the import process. Then later manually recreate the triggers in the Amazon RDS database.
- `--events` – Use if events exist in the database that you are copying. Set the parameter to `0`, which excludes the events during the import process. Then later manually recreate the events in the Amazon RDS database.
- `-p`*local_password* – Use to specify a password. In the first usage of this parameter, specify the password for the user account that you identify with the first `-u` parameter.
- `-u` *RDS_user* – Use to specify a user name. In the second usage of this parameter, specify the name of a user account on the default database for the MariaDB DB instance that you identify with the `--host` parameter.
- `--port` *port_number* – Use to specify the port for your MariaDB DB instance. By default, this is 3306 unless you changed the value when creating the DB instance.
- `--host` *host_name* – Use to specify the Domain Name System (DNS) name from the Amazon RDS DB instance endpoint, for example, `myinstance.123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the DB instance details in the Amazon RDS console.
- `-p`*RDS_password* – Use to specify a password. In the second usage of this parameter, you specify the password for the user account identified by the second `-u` parameter.

Make sure to create any stored procedures, triggers, functions, or events manually in your Amazon RDS database. If you have any of these objects in the database that you are copying, then exclude

them when you run `mysqldump` or `mariadb-dump`. To do so, include the following parameters with your `mysqldump` or `mariadb-dump` command:

- `--routines=0`
- `--triggers=0`
- `--events=0`

Example

The following example copies the `world` sample database on the local host to an RDS for MariaDB DB instance. Replace values with your own information.

For Linux, macOS, or Unix:

```
sudo mariadb-dump -u local_user \
    --databases world \
    --single-transaction \
    --compress \
    --order-by-primary  \
    --routines=0 \
    --triggers=0 \
    --events=0 \
    -plocal_password | mariadb -u rds_user \
        --port=3306 \
        --host=my_instance.123456789012.us-east-1.rds.amazonaws.com \
        -pRDS_password
```

For Windows:

Run the following command in a command prompt that has been opened by right-clicking **Command Prompt** on the Windows programs menu and choosing **Run as administrator**. Replace values with your own information.

```
mariadb-dump -u local_user ^
    --databases world ^
    --single-transaction ^
    --compress ^
    --order-by-primary  ^
    --routines=0 ^
    --triggers=0 ^
```

```
    --events=0 ^
    -plocal_password | mariadb -u RDS_user ^
        --port=3306 ^
        --host=my_instance.123456789012.us-east-1.rds.amazonaws.com ^
        -pRDS_password
```

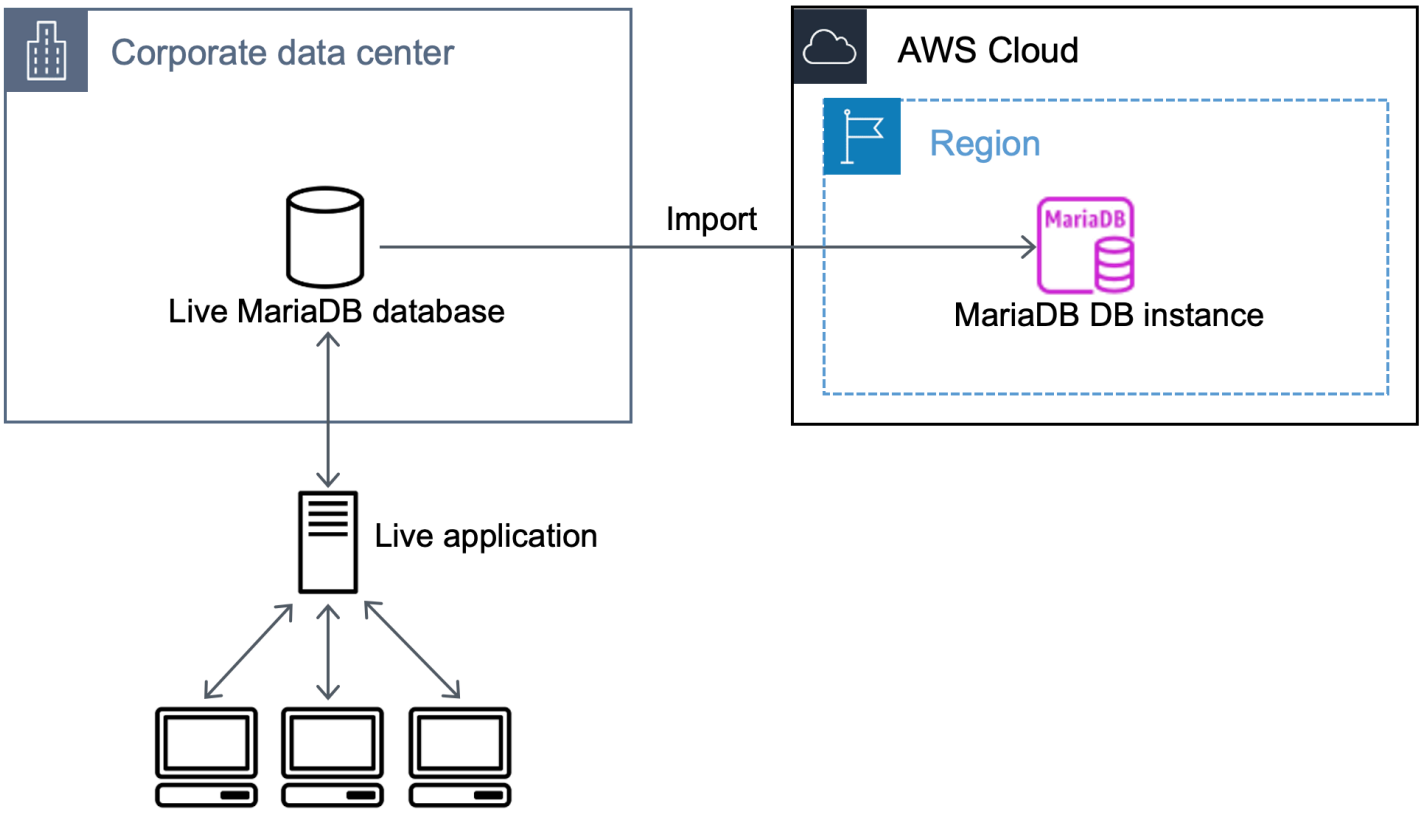
> **Note**
>
> As a security best practice, specify credentials other than the prompts shown in the example.

Importing data to an Amazon RDS for MariaDB DB instance with reduced downtime

In some cases, you might need to import data from an external MariaDB database that supports a live application to an RDS for MariaDB DB instance. Use the following procedure to minimize the impact on availability of applications. This procedure can also help if you are working with a very large database. Using this procedure, you can reduce the cost of the import by reducing the amount of data that is passed across the network to AWS.

In this procedure, you transfer a copy of your database data to an Amazon EC2 instance and import the data into a new Amazon RDS database. You then use replication to bring the Amazon RDS database up-to-date with your live external instance, before redirecting your application to the Amazon RDS database. If the external instance is MariaDB 10.0.24 or higher and the target instance is RDS for MariaDB, configure MariaDB replication based on global transaction identifiers (GTIDs). Otherwise, configure replication based on binary log coordinates. We recommend GTID-based replication if your external database supports it because GTID-based replication is a more reliable method. For more information, see Global transaction ID in the MariaDB documentation.

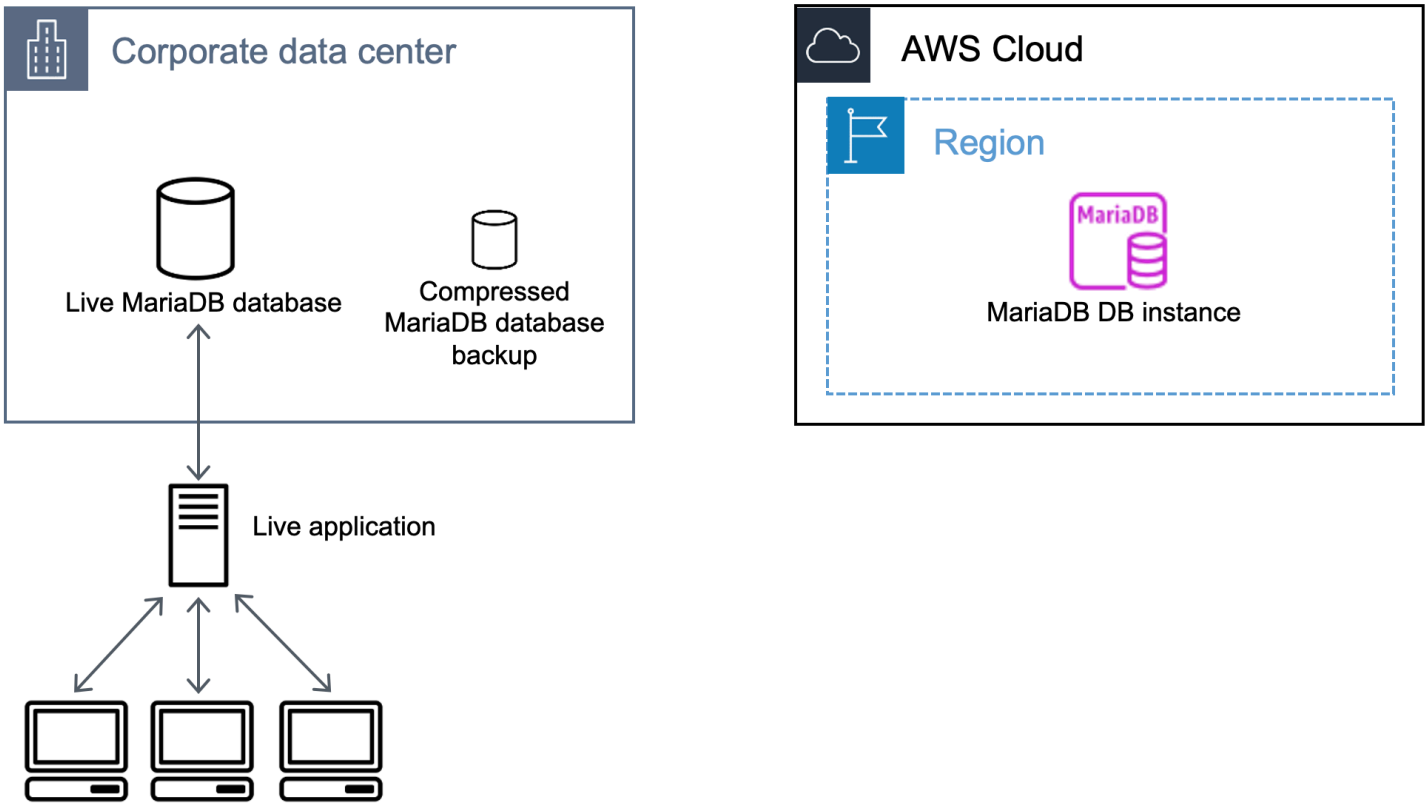
The following diagram shows importing an external MariaDB database into a MariaDB database on Amazon RDS.



Task 1: Create a copy of your existing database

The first step in the process of migrating a large amount of data to an RDS for MariaDB database with minimal downtime is to create a copy of the source data.

The following diagram shows creating a backup of the MariaDB database.



You can use the `mysqldump` or `mariadb-dump` utility to create a database backup in either SQL or delimited-text format. In MariaDB 10.5, the client is called mariadb-dump. Starting with MariaDB 11.0.1, you must use `mariadb-dump` instead of `mysqldump`. We recommend that you do a test run with each format in a non-production environment to see which method minimizes the amount of time that `mysqldump` or `mariadb-dump` runs.

We also recommend that you weigh `mysqldump` or `mariadb-dump` performance against the benefit offered by using the delimited-text format for loading. A backup using delimited-text format creates a tab-separated text file for each table being dumped. To reduce the amount of time required to import your database, you can load these files in parallel using the `LOAD DATA LOCAL INFILE` command. For more information, see Step 5: Load the data in the Importing data from any source procedure.

Before you start the backup operation, make sure to set the replication options on the MariaDB database that you are copying to Amazon RDS. The replication options include turning on binary logging and setting a unique server ID. Setting these options causes your server to start logging database transactions and prepares it to be a source replication instance later in this process.

Make sure that you're aware of the following recommendations and considerations:

- Use the `--single-transaction` option with `mysqldump` or `mariadb-dump` because it dumps a consistent state of the database. To ensure a valid dump file, don't run data definition language (DDL) statements while `mysqldump` or `mariadb-dump` is running. You can schedule a maintenance window for these operations.
- Exclude the following schemas from the dump file:
  - `sys`
  - `performance_schema`
  - `information_schema`

  The `mysqldump` and `mariadb-dump` utilities exclude these schemas by default.
- If you need to migrate users and privileges, consider using a tool that generates the data control language (DCL) for recreating them, such as the pt-show-grants utility.

To set replication options

1. Edit the `my.cnf` file. This file is usually located under `/etc`.

   ```
   sudo vi /etc/my.cnf
   ```

   Add the `log_bin` and `server_id` options to the `[mysqld]` section. The `log_bin` option provides a file name identifier for binary log files. The `server_id` option provides a unique identifier for the server in source-replica relationships.

   The following example shows the updated `[mariadb]` section of a `my.cnf` file:

   ```
   [mariadb]
   log-bin
   server-id=1
   log-basename=master1
   binlog-format=mixed
   ```

   For more information, see Setting the Replication Source Configuration in the MariaDB documentation.
2. For replication with a Multi-AZ DB cluster, enable `gtid_strict_mode`. For more information, see gtid_strict_mode in the MariaDB documentation.

   Enabling `gtid_strict_mode` isn't required for replication with a DB instance.

3. Restart the `mariadb` service.

```
sudo service mariadb restart
```

To create a backup copy of your existing database

1. Create a backup of your data using the `mysqldump` or `mariadb-dump` utility, specifying either SQL or delimited-text format.

   To improve performance and ensure data integrity, use the `--order-by-primary` and `--single-transaction` options for `mysqldump` and `mariadb-dump`.

   To avoid including the MySQL system database in the backup, don't use the `--all-databases` option with `mysqldump` or `mariadb-dump`. For more information, see Creating a Data Snapshot Using mysqldump in the MySQL documentation.

   Use chmod, if necessary, to make sure that the directory where the backup file is being created is writeable.

   > **Important**
   >
   > On Windows, run the command window as an administrator.

   - To produce SQL output, use the following command. For MariaDB 10.11 versions and lower, replace `mariadb-dump` with `mysqldump`.

     For Linux, macOS, or Unix:

     ```
     sudo mariadb-dump \
         --databases database_name \
         --master-data=2  \
         --single-transaction \
         --order-by-primary \
         -r backup.sql \
         -u local_user \
         -ppassword
     ```

> **Note**
>
> As a security best practice, specify credentials other than the prompts shown in the example.

For Windows:

```
mariadb-dump ^
    --databases database_name ^
    --master-data=2  ^
    --single-transaction ^
    --order-by-primary ^
    -r backup.sql ^
    -u local_user ^
    -ppassword
```

> **Note**
>
> As a security best practice, specify credentials other than the prompts shown in the example.

- To produce delimited-text output, use the following command. For MariaDB 11.01 and higher versions, replace `mysqldump` with `mariadb-dump`.

  For Linux, macOS, or Unix:

```
sudo mysqldump \
    --tab=target_directory \
    --fields-terminated-by ',' \
    --fields-enclosed-by '"' \
    --lines-terminated-by 0x0d0a \
    database_name \
    --master-data=2 \
    --single-transaction \
    --order-by-primary \
    -ppassword
```

  For Windows:

```
mysqldump ^
    --tab=target_directory ^
    --fields-terminated-by "," ^
    --fields-enclosed-by """ ^
    --lines-terminated-by 0x0d0a ^
    database_name ^
    --master-data=2 ^
    --single-transaction ^
    --order-by-primary ^
    -ppassword
```

Note

As a security best practice, specify credentials other than the prompts shown in the example.

Make sure to create any stored procedures, triggers, functions, or events manually in your Amazon RDS database. If you have any of these objects in the database that you are copying, then exclude them when you run `mysqldump` or `mariadb-dump`. To do so, include the following arguments with your `mysqldump` or `mariadb-dump` command:

- `--routines=0`
- `--triggers=0`
- `--events=0`

When you run `mysqldump` and specify the delimited-text format, a `CHANGE MASTER TO` comment is returned. This comment contains the master log file name and position. If the external instance is a MariaDB 10.0.23 or lower version, note the values for `MASTER_LOG_FILE` and `MASTER_LOG_POS`. You need these values when setting up replication.

The following output is returned for MariaDB versions.

```
-- Position to start replication or point-in-time recovery from
--
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031',
 MASTER_LOG_POS=107;
```

2. If the external instance you are using is MariaDB version 10.0.24 or higher, use GTID-based replication. Run `SHOW MASTER STATUS` on the external MariaDB instance to get the binary log file name and position, and then convert them to a GTID by running `BINLOG_GTID_POS` on the external MariaDB instance.

   ```
   SELECT BINLOG_GTID_POS('binary_log_file_name', binary_log_file_position);
   ```

   Note the GTID returned. You need the GTID to configure replication.

3. Compress the copied data to reduce the amount of network resources needed to copy your data to the Amazon RDS database. Note the size of the backup file. You need this information when determining how large an Amazon EC2 instance to create. When you are done, compress the backup file using GZIP or your preferred compression utility.

   - To compress SQL output, use the following command:

     ```
     gzip backup.sql
     ```

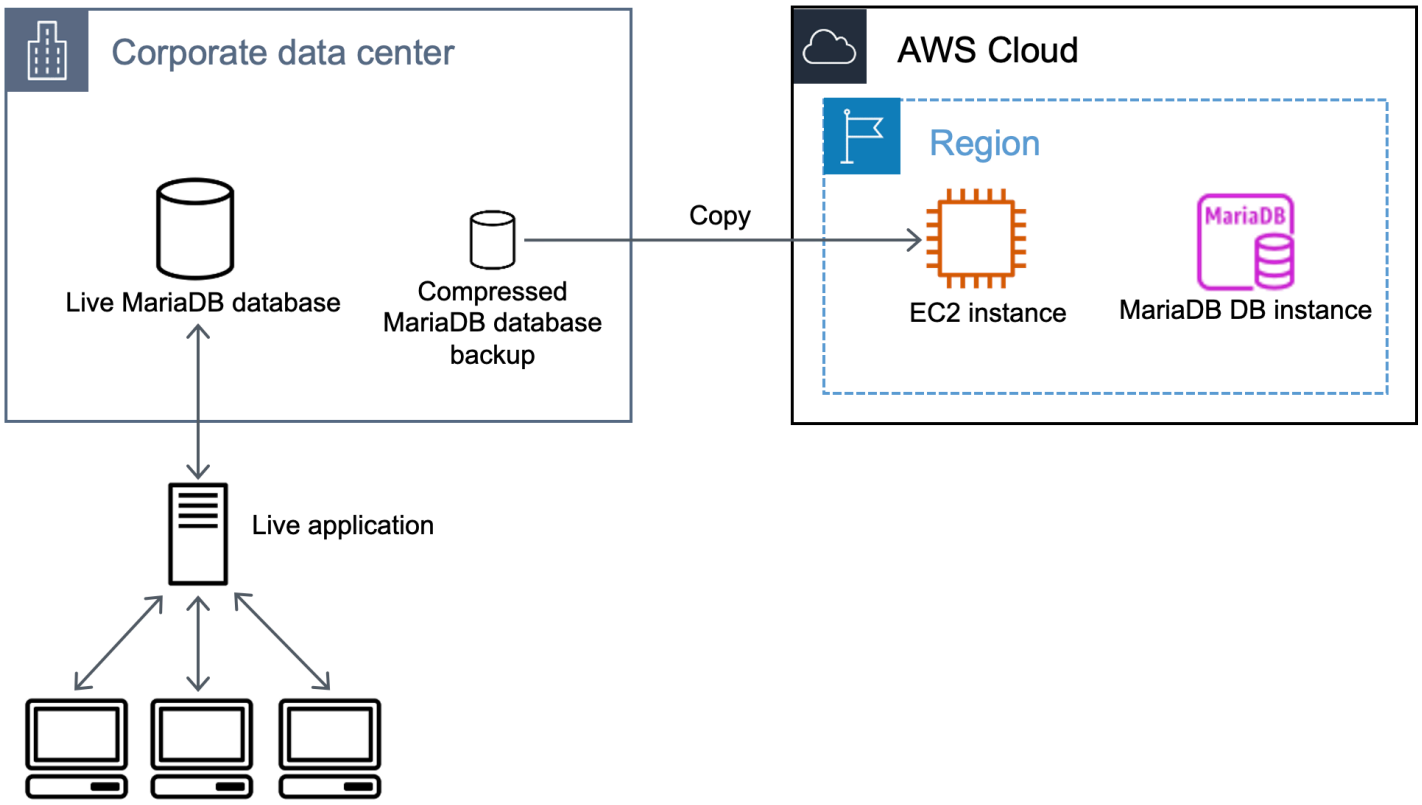
   - To compress delimited-text output, use the following command:

     ```
     tar -zcvf backup.tar.gz target_directory
     ```

Task 2: Create an Amazon EC2 instance and copy the compressed database

Copying your compressed database backup file to an Amazon EC2 instance takes fewer network resources than doing a direct copy of uncompressed data between database instances. After your data is in Amazon EC2, you can copy it from there directly to your MariaDB database. For you to save on the cost of network resources, your Amazon EC2 instance must be in the same AWS Region as your Amazon RDS DB instance. Having the Amazon EC2 instance in the same AWS Region as your Amazon RDS database also reduces network latency during the import.

The following diagram shows copying the database backup to an Amazon EC2 instance.



To create an Amazon EC2 instance and copy your data

1. In the AWS Region where you plan to create the Amazon RDS database, create a virtual private cloud (VPC), a VPC security group, and a VPC subnet. Ensure that the inbound rules for your VPC security group allow the IP addresses required for your application to connect to AWS. You can specify a range of IP addresses—for example, `203.0.113.0/24`—or another VPC security group. You can use the Amazon VPC console to create and manage VPCs, subnets, and security groups. For more information, see Getting started with Amazon VPC in the *Amazon Virtual Private Cloud User Guide*.
2. Open the Amazon EC2 console and choose the AWS Region to contain both your Amazon EC2 instance and your Amazon RDS database. Launch an Amazon EC2 instance using the VPC, subnet, and security group that you created in Step 1. Ensure that you select an instance type with enough storage for your database backup file when it is uncompressed. For details on Amazon EC2 instances, see Getting started with Amazon EC2 in the *Amazon Elastic Compute Cloud User Guide*.
3. To connect to your Amazon RDS database from your Amazon EC2 instance, edit your VPC security group. Add an inbound rule specifying the private IP address of your EC2 instance. You can find the private IP address on the **Details** tab of the **Instance** pane in the EC2 console

window. To edit the VPC security group and add an inbound rule, choose **Security Groups** in the EC2 console navigation pane, choose your security group, and then add an inbound rule for MySQL or Aurora specifying the private IP address of your EC2 instance. To learn how to add an inbound rule to a VPC security group, see Security group rules in the *Amazon Virtual Private Cloud User Guide*.

4. Copy your compressed database backup file from your local system to your Amazon EC2 instance. Use `chmod`, if necessary, to make sure that you have write permission for the target directory of the Amazon EC2 instance. You can use `scp` or a Secure Shell (SSH) client to copy the file. The following command is an example `scp` command:

```
scp -r -i key pair.pem backup.sql.gz ec2-user@EC2 DNS:/target_directory/backup.sql.gz
```

> **Important**
>
> When copying sensitive data, be sure to use a secure network transfer protocol.

5. Connect to your Amazon EC2 instance and install the latest updates and the MariaDB client tools using the following commands:

```
sudo yum update -y
sudo yum install mariadb1011-client-utils -y
```

For more information, see Connect to your instance for Linux instances in the *Amazon Elastic Compute Cloud User Guide* and MariaDB Connectors in the MariaDB documentation.

6. While connected to your Amazon EC2 instance, decompress your database backup file. The following commands are examples.

- To decompress SQL output, use the following command:

```
gzip backup.sql.gz -d
```

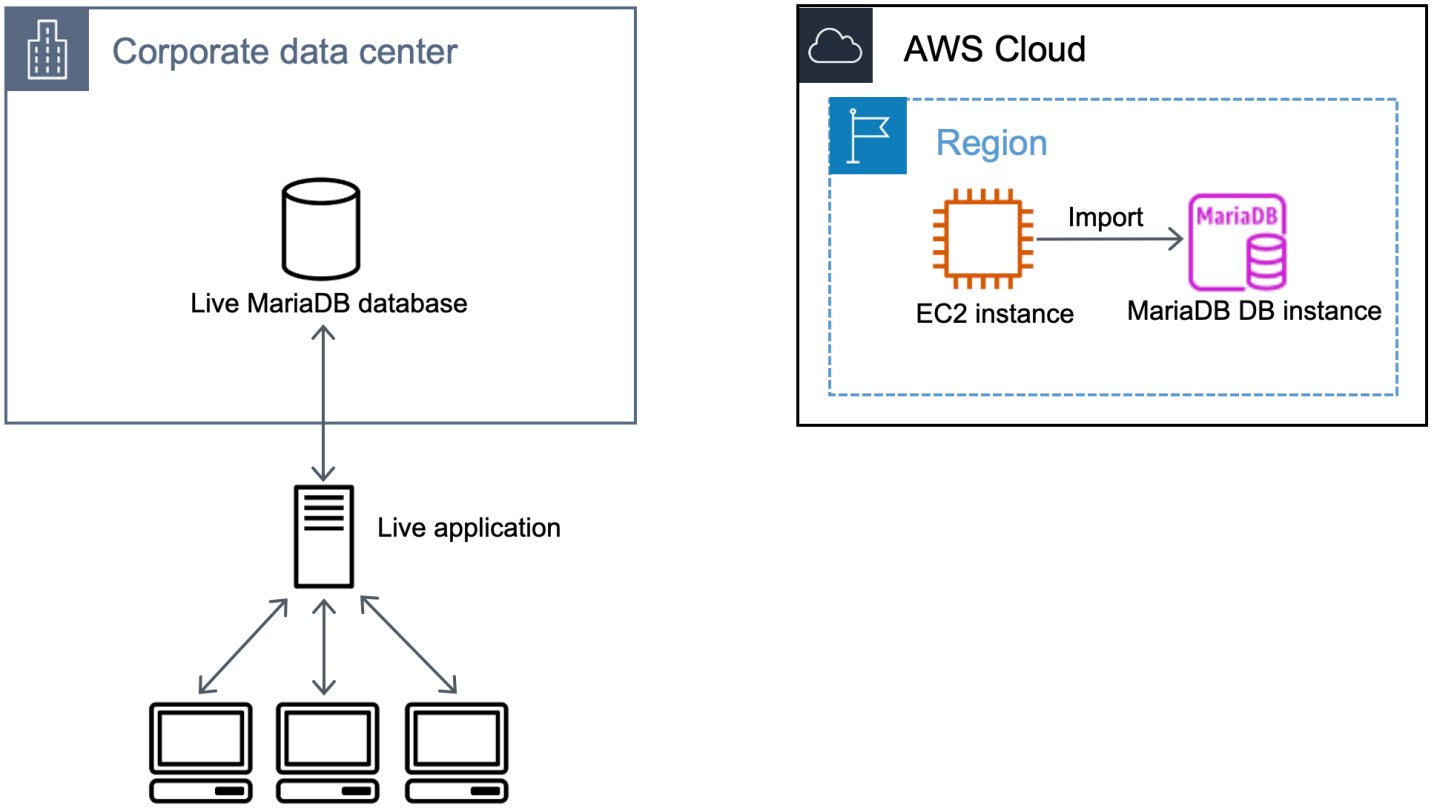
- To decompress delimited-text output, use the following command:

```
tar xzvf backup.tar.gz
```

Task 3: Create a MariaDB database and import data from your Amazon EC2 instance

By creating an RDS for MariaDB DB instance in the same AWS Region as your Amazon EC2 instance, you can import the database backup file from Amazon EC2 faster than over the internet.

The following diagram shows importing the backup from an Amazon EC2 instance into a MariaDB database.



To create a MariaDB database and import your data

1. Determine which DB instance class and what amount of storage space is required to support the expected workload for this Amazon RDS database. As part of this process, decide what is sufficient space and processing capacity for your data load procedures. Also, decide what is required to handle the production workload. You can estimate this based on the size and resources of the source MariaDB database. For more information, see DB instance classes.
2. Create a DB instance in the AWS Region that contains your Amazon EC2 instance. Follow the instructions in Creating an Amazon RDS DB instance and use the following guidelines:
   - Specify a DB engine version that is compatible with your source DB instance.

- Specify the same virtual private cloud (VPC) and VPC security group as for your Amazon EC2 instance. This approach ensures that your Amazon EC2 instance and your Amazon RDS instance are visible to each other over the network. Make sure your DB instance is publicly accessible. To set up replication with your source database as described in a following section, your DB instance must be publicly accessible.
- Don't configure multiple Availability Zones, backup retention, or read replicas until after you have imported the database backup. When that import is completed, you can configure Multi-AZ and backup retention for the production instance.

3. Review the default configuration options for the Amazon RDS database. If the default parameter group for the database doesn't have the configuration options that you want, find a different one that does or create a new parameter group. For more information about creating a parameter group, see Parameter groups for Amazon RDS.
4. Connect to the new Amazon RDS database as the master user. Create the users required to support the administrators, applications, and services that need to access the DB instance. The hostname for the Amazon RDS database is the **Endpoint** value for this DB instance without the port number, for example, `mysampledb.123456789012.us-west-2.rds.amazonaws.com`. You can find the endpoint value in the database details in the Amazon RDS console.
5. Connect to your Amazon EC2 instance. For more information, see Connect to your instance for Linux instances in the *Amazon Elastic Compute Cloud User Guide*.
6. Connect to your Amazon RDS database as a remote host from your Amazon EC2 instance using the `mysql` command. The following command is an example:

   ```
   mysql -h host_name -P 3306 -u db_master_user -p
   ```

   The *host_name* is the Amazon RDS database endpoint.
7. At the `mysql` prompt, run the `source` command and pass it the name of your database dump file. This command loads the data into the Amazon RDS DB instance.

   - For SQL format, use the following command:

     ```
     MariaDB [(none)]> source backup.sql;
     ```

   - For delimited-text format, first create the database, if it isn't the default database that you created when setting up the Amazon RDS database.

```
MariaDB [(none)]> create database database_name;
MariaDB [(none)]> use database_name;
```

Then create the tables.

```
MariaDB [(none)]> source table1.sql
MariaDB [(none)]> source table2.sql
etc...
```

Then import the data.

```
MariaDB [(none)]> LOAD DATA LOCAL INFILE 'table1.txt' INTO TABLE table1 FIELDS
 TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '0x0d0a';
MariaDB [(none)]> LOAD DATA LOCAL INFILE 'table2.txt' INTO TABLE table2 FIELDS
 TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '0x0d0a';
etc...
```

To improve performance, you can perform these operations in parallel from multiple connections so that all of your tables are created and then loaded at the same time.

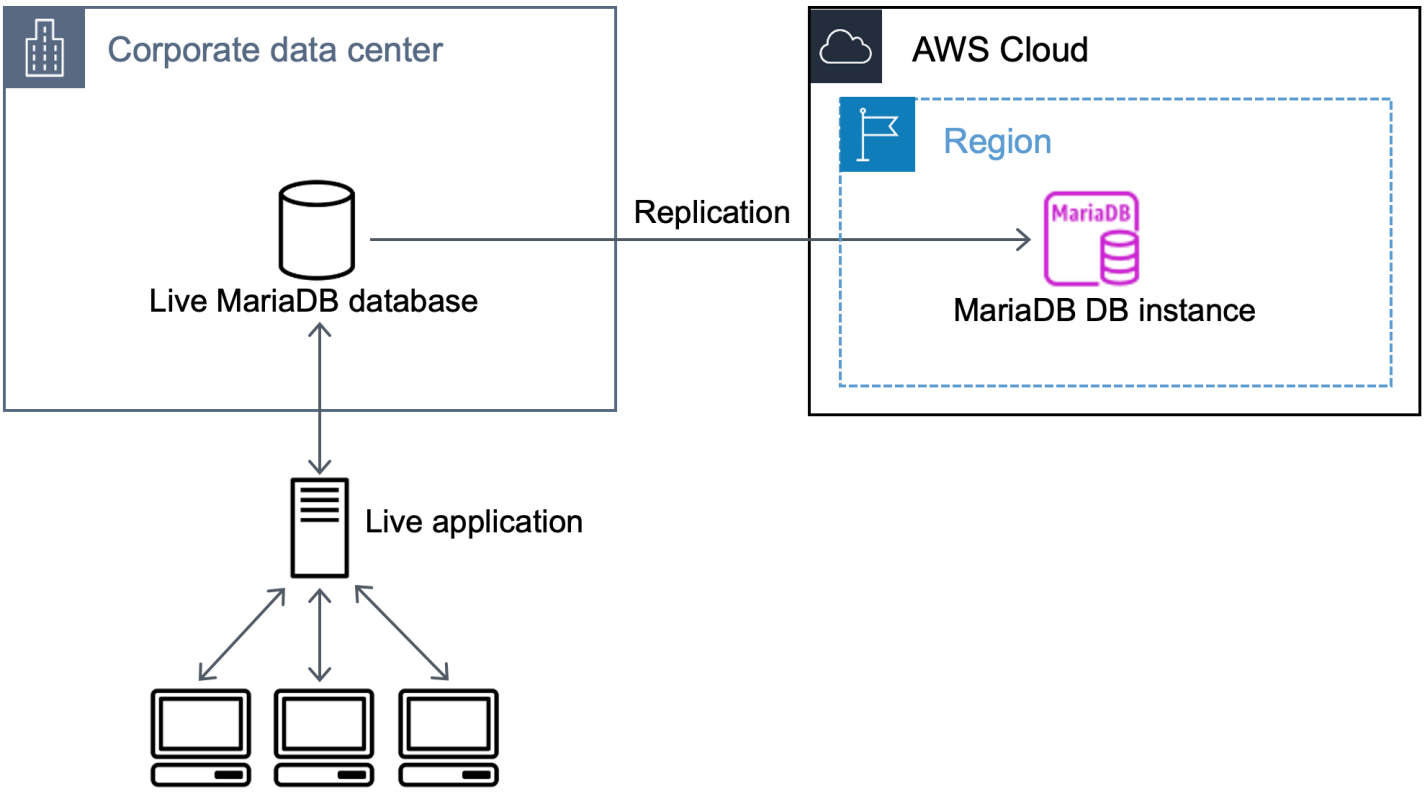
> **Note**
>
> If you used any data-formatting options with `mysqldump` or `mariadb-dump` when you initially dumped the table, make sure to use the same options with `LOAD DATA LOCAL INFILE` to ensure proper interpretation of the data file contents.

8. Run a simple `SELECT` query against one or two of the tables in the imported database to verify that the import was successful.

If you no longer need the Amazon EC2 instance used in this procedure, terminate the EC2 instance to reduce your AWS resource usage. To terminate an EC2 instance, see Terminate an instance in the *Amazon Elastic Compute Cloud User Guide*.

Task 4: Replicate data from your external database to your new Amazon RDS database

Your source database was likely updated during the time that it took to copy and transfer the data to the MariaDB database. You can use replication to bring the copied database up-to-date with the source database.



The permissions required to start replication on an Amazon RDS database are restricted and aren't available to your Amazon RDS master user. Because of this, use the appropriate Amazon RDS stored procedure:

- mysql.rds_set_external_master
- mysql.rds_set_external_master_gtid to configure replication and mysql.rds_start_replication to start replication

To start replication

In Task 1, when you set replication options, you turned on binary logging and set a unique server ID for your source database. Now, you can set up your Amazon RDS database as a replica with your live database as the source replication instance.

1. In the Amazon RDS console, add the IP address of the server that hosts the source database to the VPC security group for the Amazon RDS database. For more information on configuring a VPC security group, see Configure security group rules in the *Amazon Virtual Private Cloud User Guide*.

   You might also need to configure your local network to permit connections from the IP address of your Amazon RDS database so that it can communicate with your source instance. To find the IP address of the Amazon RDS database, use the `host` command:

   ```
   host host_name
   ```

   The *host_name* is the DNS name from the Amazon RDS database endpoint, for example `myinstance.123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the DB instance details in the Amazon RDS console.

2. Using the client of your choice, connect to the source instance and create a user to be used for replication. This account is used solely for replication and must be restricted to your domain to improve security. The following command is an example:

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   > **Note**
   >
   > Specify credentials other than the prompts shown here as a security best practice.

3. For the source instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. For example, to grant the `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the '`repl_user`' user for your domain, issue the following command:

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

4. If you used SQL format to create your backup file and the external instance isn't MariaDB 10.0.24 or higher, look at the contents of that file by running the following command:

   ```
   cat backup.sql
   ```

The file includes a `CHANGE MASTER TO` comment that contains the master log file name and position. This comment is included in the backup file when you use the `--master-data` option with `mysqldump`. Note the values for `MASTER_LOG_FILE` and `MASTER_LOG_POS`.

```
--
-- Position to start replication or point-in-time recovery from
--

-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;
```

If you used delimited text format to create your backup file and the external instance isn't MariaDB 10.0.24 or higher, you should already have binary log coordinates from Step 1 in Task 1 when you created a backup copy of your existing database.

If the external instance is MariaDB 10.0.24 or higher, you should already have the GTID from which to start replication from Step 2 in Task 1 when you created a backup copy of your existing database.

5. Make the Amazon RDS database the replica. If the external instance isn't MariaDB 10.0.24 or higher, connect to the Amazon RDS database as the master user and identify the source database as the source replication instance by using the mysql.rds_set_external_master stored procedure.

   If you have a SQL format backup file, use the master log file name and master log position that you determined in Step 4. If you used delimited-text format, use the name and position that you determined when creating the backup files. The following command is an example:

   ```
   CALL mysql.rds_set_external_master ('myserver.mydomain.com', 3306,
       'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
   ```

   > **Note**
   >
   > Specify credentials other than the prompts shown here as a security best practice.

   If the external instance is MariaDB 10.0.24 or higher, connect to the Amazon RDS database as the master user and identify the source database as the source replication instance by using the mysql.rds_set_external_master_gtid stored procedure. Use the GTID that you determined in Step

2 in Task 1 when you created a backup copy of your existing database. The following command is an example:

```
CALL mysql.rds_set_external_master_gtid ('source_server_ip_address', 3306,
 'ReplicationUser', 'password', 'GTID', 1);
```

The *source_server_ip_address* is the IP address of source replication instance. An EC2 private DNS address isn't currently supported.

> **Note**
>
> Specify credentials other than the prompts shown here as a security best practice.

6. On the Amazon RDS database, to start replication, run the following command that uses the mysql.rds_start_replication stored procedure:

```
CALL mysql.rds_start_replication;
```

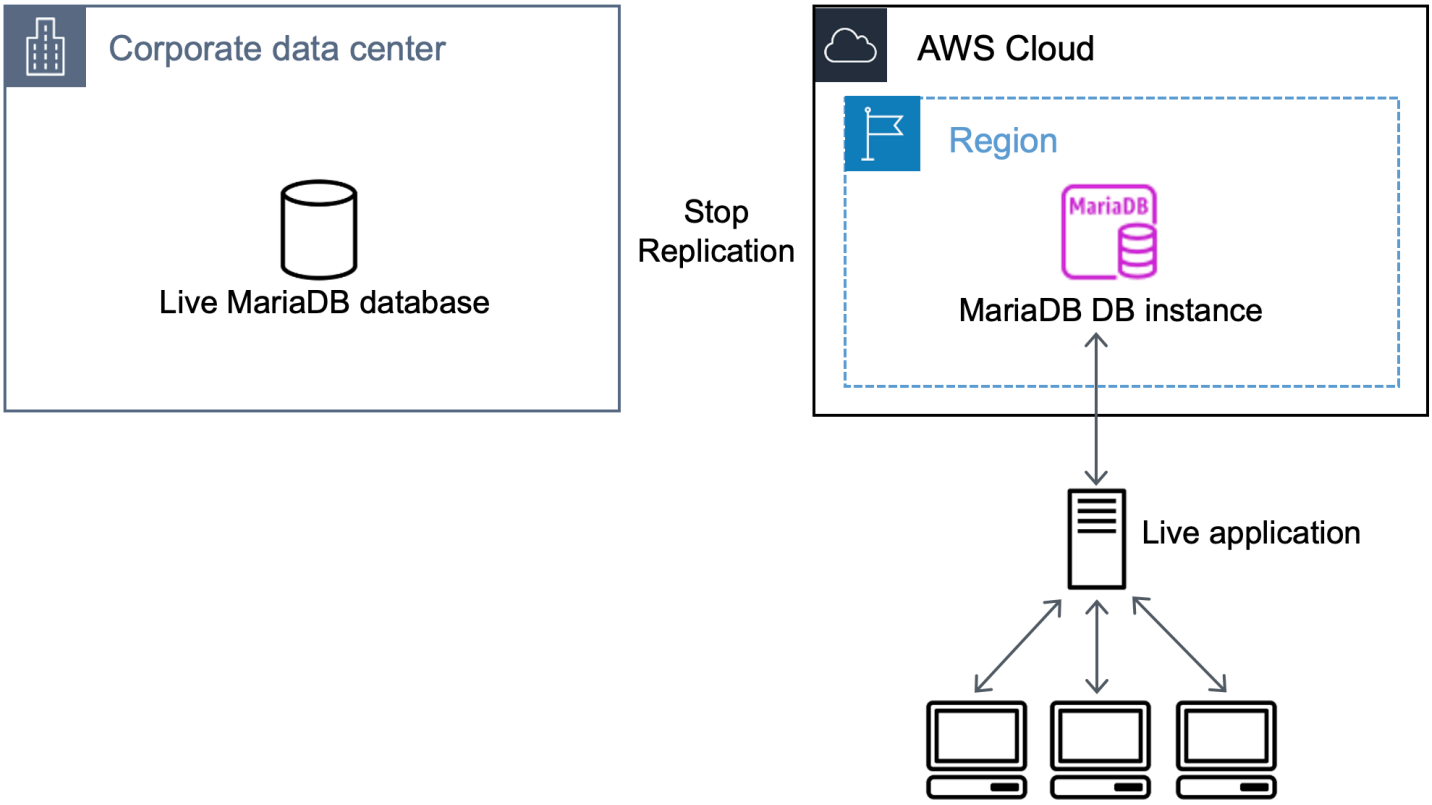
7. On the Amazon RDS database, to determine when the replica is up-to-date with the source replication instance, run the SHOW REPLICA STATUS command. The results of the `SHOW REPLICA STATUS` command include the `Seconds_Behind_Master` field. When the `Seconds_Behind_Master` field returns 0, then the replica is up-to-date with the source replication instance.

   For a MariaDB 10.5, 10.6, 10.11, 11.4, or 11.8 DB instance, use the mysql.rds_replica_status stored procedure instead of running the MySQL command.

8. After the Amazon RDS database is up-to-date, turn on automated backups so you can restore that database if needed. You can turn on or modify automated backups for your Amazon RDS database by using the Amazon RDS console. For more information, see Introduction to backups.

Task 5: Redirect your live application to your Amazon RDS instance

After the MariaDB database is up-to-date with the source replication instance, you can now update your live application to use the Amazon RDS instance.



To redirect your live application to your MariaDB database and stop replication

1. To add the VPC security group for the Amazon RDS database, add the IP address of the server that hosts the application. For more information on modifying a VPC security group, see Configure security group rules in the *Amazon Virtual Private Cloud User Guide*.
2. Verify that the `Seconds_Behind_Master` field in the SHOW REPLICA STATUS command results is 0, which indicates that the replica is up-to-date with the source replication instance.

   ```
   SHOW REPLICA STATUS;
   ```

   For a MariaDB 10.5, 10.6, 10.11, 11.4, or 11.8 DB instance, use the mysql.rds_replica_status procedure instead of running the MySQL command.
3. Close all connections to the source when their transactions complete.
4. Update your application to use the Amazon RDS database. This update typically involves changing the connection settings to identify the hostname and port of the Amazon RDS database, the user account and password to connect with, and the database to use.
5. Connect to the DB instance.

6. Stop replication for the Amazon RDS instance by running the following command that uses the mysql.rds_stop_replication stored procedure:

```
CALL mysql.rds_stop_replication;
```

7. Reset the replication configuration so this instance is no longer identified as a replica by running the following command that uses the mysql.rds_reset_external_master stored procedure on your Amazon RDS database:

```
CALL mysql.rds_reset_external_master;
```

8. Turn on additional Amazon RDS features such as Multi-AZ support and read replicas. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS and Working with DB instance read replicas.

Importing data from any source to an Amazon RDS for MariaDB DB instance

With Amazon RDS, you can migrate existing MariaDB data from any source to an RDS for MariaDB DB instance. You can transfer data from on-premises databases, other cloud providers, or existing RDS for MariaDB DB instances to your target RDS for MariaDB DB instance. With this functionality, you can consolidate databases, implement disaster recovery solutions, or transition from self-managed databases. Common scenarios include moving from self-hosted MariaDB servers to fully managed Amazon RDS DB instances, consolidating multiple MariaDB databases into a single DB instance, or creating test environments with production data. The following sections provide step-by-step instructions for importing your MariaDB data using methods such as `mariadb-dump`, backup files, or replication.

Step 1: Create flat files containing the data to be loaded

Use a common format, such as comma-separated values (CSV), to store the data to be loaded. Each table must have its own file—you can't combine data for multiple tables in the same file. Give each file the same name as the table it corresponds to. The file extension can be anything you like. For example, if the table name is `sales`, the file name could be `sales.csv` or `sales.txt`.

If possible, order the data by the primary key of the table being loaded. Doing this drastically improves load times and minimizes disk storage requirements.

The speed and efficiency of this procedure depends on keeping the size of the files small. If the uncompressed size of any individual file is larger than 1 GiB, split it into multiple files and load each one separately.

On Unix-like systems (including Linux), use the `split` command. For example, the following command splits the `sales.csv` file into multiple files of less than 1 GiB, splitting only at line breaks (-C 1024m). The names of the new files include ascending numerical suffixes. The following command produces files with names such as `sales.part_00` and `sales.part_01`.

```
split -C 1024m -d sales.csv sales.part_
```

Similar utilities are available for other operating systems.

You can store the flat files anywhere. However, when you load the data in Step 5, you must invoke the `mysql` shell from the same location where the files exist, or use the absolute path for the files when you run `LOAD DATA LOCAL INFILE`.

Step 2: Stop any applications from accessing the target DB instance

Before starting a large load, stop all application activity from accessing the target DB instance that you plan to load to. We recommend this particularly if other sessions will be modifying the tables being loaded or tables that they reference. Doing this reduces the risk of constraint violations occurring during the load and improves load performance. It also makes it possible to restore the DB instance to the point just before the load without losing changes made by processes not involved in the load.

Of course, this might not be possible or practical. If you can't stop applications from accessing the DB instance before the load, take steps to ensure the availability and integrity of your data. The specific steps required vary greatly depending upon specific use cases and site requirements.

Step 3: Create a DB snapshot

If you plan to load data into a new DB instance that contains no data, you can skip this step. Otherwise, we recommend that you create DB snapshots of the target Amazon RDS DB instance both before and after the data load. Amazon RDS DB snapshots are complete backups of your DB instance that you can use to restore your DB instance to a known state. When you initiate a DB snapshot, I/O operations to your DB instance are momentarily suspended while your database is backed up.

Creating a DB snapshot immediately before the load makes it possible for you to restore the database to its state before the load, if you need to. A DB snapshot taken immediately after the load protects you from having to load the data again in case of a mishap. You can also use DB snapshots after the load to import data into new database instances.

The following example runs the AWS CLI create-db-snapshot command to create a DB snapshot of the AcmeRDS instance and give the DB snapshot the identifier "preload".

For Linux, macOS, or Unix:

```
aws rds create-db-snapshot \
    --db-instance-identifier AcmeRDS \
    --db-snapshot-identifier preload
```

For Windows:

```
aws rds create-db-snapshot ^
    --db-instance-identifier AcmeRDS ^
    --db-snapshot-identifier preload
```

You can also use the restore from DB snapshot functionality to create test DB instances for dry runs or to undo changes made during the load.

Keep in mind that restoring a database from a DB snapshot creates a new DB instance that, like all DB instances, has a unique identifier and endpoint. To restore the DB instance without changing the endpoint, first delete the DB instance so that you can reuse the endpoint.

For example, to create a DB instance for dry runs or other testing, you give the DB instance its own identifier. In the example, AcmeRDS-2" is the identifier. The example connects to the DB instance using the endpoint associated with AcmeRDS-2. For more information, see restore-db-instance-from-db-snapshot.

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier AcmeRDS-2 \
    --db-snapshot-identifier preload
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^
    --db-instance-identifier AcmeRDS-2 ^
    --db-snapshot-identifier preload
```

To reuse the existing endpoint, first delete the DB instance and then give the restored database the same identifier. For more information, see delete-db-instance.

The following example also takes a final DB snapshot of the DB instance before deleting it. This is optional but recommended.

For Linux, macOS, or Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier AcmeRDS \
    --final-db-snapshot-identifier AcmeRDS-Final

aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier AcmeRDS \
    --db-snapshot-identifier preload
```

For Windows:

```
aws rds delete-db-instance ^
    --db-instance-identifier AcmeRDS ^
    --final-db-snapshot-identifier AcmeRDS-Final

aws rds restore-db-instance-from-db-snapshot ^
    --db-instance-identifier AcmeRDS ^
    --db-snapshot-identifier preload
```

Step 4 (Optional): Turn off Amazon RDS automated backups

> **Warning**
>
> Don't turn off automated backups if you need to perform point-in-time recovery.

Turning off automated backups is a performance optimization and isn't required for data loads. Turning off automated backups erases all existing backups. As a result, after you turn off

automated backups, point-in-time recovery isn't possible. Manual DB snapshots aren't affected by turning off automated backups. All existing manual DB snapshots are still available for restore.

Turning off automated backups reduces load time by about 25 percent and reduces the amount of storage space required during the load. If you plan to load data into a new DB instance that contains no data, turning off backups is an easy way to speed up the load and avoid using the additional storage needed for backups. However, in some cases you might plan to load into a DB instance that already contains data. If so, weigh the benefits of turning off backups against the impact of losing the ability to perform point-in-time-recovery.

DB instances have automated backups turned on by default (with a one day retention period). To turn off automated backups, set the backup retention period to zero. After the load, you can turn backups back on by setting the backup retention period to a nonzero value. To turn on or turn off backups, Amazon RDS shuts the DB instance down and then restarts it to turn MariaDB logging on or off.

Run the AWS CLI `modify-db-instance` command to set the backup retention to zero and apply the change immediately. Setting the retention period to zero requires a DB instance restart, so wait until the restart has completed before proceeding. For more information, see modify-db-instance.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier AcmeRDS \
    --apply-immediately \
    --backup-retention-period 0
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier AcmeRDS ^
    --apply-immediately ^
    --backup-retention-period 0
```

You can check the status of your DB instance with the AWS CLI describe-db-instances command. The following example displays the DB instance status of the `AcmeRDS` DB instance:

```
aws rds describe-db-instances --db-instance-identifier AcmeRDS --query "*[].
{DBInstanceStatus:DBInstanceStatus}"
```

When the DB instance status is `available`, you're ready to proceed to the next step.

Step 5: Load the data

To read rows from your flat files into the database tables, use the MariaDB `LOAD DATA LOCAL INFILE` statement.

> **Note**
>
> You must invoke the `mariadb` shell from the same location where your flat files exist, or use the absolute path for the files when you run `LOAD DATA LOCAL INFILE`.

The following example shows how to load data from a file named `sales.txt` into a table named `Sales` in the database:

```
MariaDB [(none)]> LOAD DATA LOCAL INFILE 'sales.txt' INTO TABLE Sales FIELDS TERMINATED
 BY ' ' ENCLOSED BY '' ESCAPED BY '\\';
Query OK, 1 row affected (0.01 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

For more information about the `LOAD DATA` statement, see LOAD DATA INFILE in the MariaDB documentation.

Step 6: Turn back on Amazon RDS automated backups

If you turned off Amazon RDS automated backups in Step 4, after the load is finished, turn automated backups on by setting the backup retention period back to its preload value. As noted in Step 4, Amazon RDS restarts the DB instance, so be prepared for a brief outage.

The following example runs the AWS CLI modify-db-instance command to turn on automated backups for the `AcmeRDS` DB instance and set the retention period to one day:

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier AcmeRDS \
    --backup-retention-period 1 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier AcmeRDS ^
    --backup-retention-period 1 ^
    --apply-immediately
```

Working with MariaDB replication in Amazon RDS

You usually use read replicas to configure replication between Amazon RDS DB instances. For general information about read replicas, see Working with DB instance read replicas. For specific information about working with read replicas on Amazon RDS for MariaDB, see Working with MariaDB read replicas.

You can also configure replication based on binary log coordinates for a MariaDB DB instance. For MariaDB instances, you can also configure replication based on global transaction IDs (GTIDs), which provides better crash safety. For more information, see Configuring GTID-based replication with an external source instance.

The following are other replication options available with RDS for MariaDB:

- You can set up replication between an RDS for MariaDB DB instance and a MySQL or MariaDB instance that is external to Amazon RDS. For information about configuring replication with an external source, see Configuring binary log file position replication with an external source instance.
- You can configure replication to import databases from a MySQL or MariaDB instance that is external to Amazon RDS, or to export databases to such instances. For more information, see Importing data to an Amazon RDS for MariaDB DB instance with reduced downtime and Exporting data from a MySQL DB instance by using replication.

For any of these replication options, you can use either row-based replication, statement-based, or mixed replication. Row-based replication only replicates the changed rows that result from a SQL statement. Statement-based replication replicates the entire SQL statement. Mixed replication uses statement-based replication when possible, but switches to row-based replication when SQL statements that are unsafe for statement-based replication are run. In most cases, mixed replication is recommended. The binary log format of the DB instance determines whether replication is row-based, statement-based, or mixed. For information about setting the binary log format, see Configuring MariaDB binary logging.

For information about replication compatibility between MariaDB versions, see Replication Compatibility in the MariaDB documentation.

Topics

- Working with MariaDB read replicas

- Configuring GTID-based replication with an external source instance
- Configuring binary log file position replication with an external source instance

Working with MariaDB read replicas

Following, you can find specific information about working with read replicas on Amazon RDS for MariaDB. For general information about read replicas and instructions for using them, see Working with DB instance read replicas.

- Configuring replication filters with MariaDB
- Configuring delayed replication with MariaDB
- Updating read replicas with MariaDB
- Working with Multi-AZ read replica deployments with MariaDB
- Using cascading read replicas with RDS for MariaDB
- Monitoring MariaDB read replicas
- Starting and stopping replication with MariaDB read replicas
- Troubleshooting a MariaDB read replica problem

Configuring read replicas with MariaDB

Before a MariaDB DB instance can serve as a replication source, make sure to turn on automatic backups on the source DB instance by setting the backup retention period to a value other than 0. This requirement also applies to a read replica that is the source DB instance for another read replica.

You can create up to 15 read replicas from one DB instance within the same Region. For replication to operate effectively, each read replica should have as the same amount of compute and storage resources as the source DB instance. If you scale the source DB instance, also scale the read replicas.

RDS for MariaDB supports cascading read replicas. To learn how to configure cascading read replicas, see Using cascading read replicas with RDS for MariaDB.

You can run multiple read replica create and delete actions at the same time that reference the same source DB instance. When you perform these actions, stay within the limit of 15 read replicas for each source instance.

Configuring replication filters with MariaDB

You can use replication filters to specify which databases and tables are replicated with a read replica. Replication filters can include databases and tables in replication or exclude them from replication.

The following are some use cases for replication filters:

- To reduce the size of a read replica. With replication filtering, you can exclude the databases and tables that aren't needed on the read replica.
- To exclude databases and tables from read replicas for security reasons.
- To replicate different databases and tables for specific use cases at different read replicas. For example, you might use specific read replicas for analytics or sharding.
- For a DB instance that has read replicas in different AWS Regions, to replicate different databases or tables in different AWS Regions.

> **Note**
>
> You can also use replication filters to specify which databases and tables are replicated with a primary MariaDB DB instance that is configured as a replica in an inbound replication topology. For more information about this configuration, see Configuring binary log file position replication with an external source instance.

Topics

- Setting replication filtering parameters for RDS for MariaDB
- Replication filtering limitations for RDS for MariaDB
- Replication filtering examples for RDS for MariaDB
- Viewing the replication filters for a read replica

Setting replication filtering parameters for RDS for MariaDB

To configure replication filters, set the following replication filtering parameters on the read replica:

- `replicate-do-db` – Replicate changes to the specified databases. When you set this parameter for a read replica, only the databases specified in the parameter are replicated.
- `replicate-ignore-db` – Don't replicate changes to the specified databases. When the `replicate-do-db` parameter is set for a read replica, this parameter isn't evaluated.
- `replicate-do-table` – Replicate changes to the specified tables. When you set this parameter for a read replica, only the tables specified in the parameter are replicated. Also, when the `replicate-do-db` or `replicate-ignore-db` parameter is set, the database that includes the specified tables must be included in replication with the read replica.
- `replicate-ignore-table` – Don't replicate changes to the specified tables. When the `replicate-do-table` parameter is set for a read replica, this parameter isn't evaluated.
- `replicate-wild-do-table` – Replicate tables based on the specified database and table name patterns. The `%` and _ wildcard characters are supported. When the `replicate-do-db` or `replicate-ignore-db` parameter is set, make sure to include the database that includes the specified tables in replication with the read replica.
- `replicate-wild-ignore-table` – Don't replicate tables based on the specified database and table name patterns. The `%` and _ wildcard characters are supported. When the `replicate-do-table` or `replicate-wild-do-table` parameter is set for a read replica, this parameter isn't evaluated.

The parameters are evaluated in the order that they are listed. For more information about how these parameters work, see the MariaDB documentation.

By default, each of these parameters has an empty value. On each read replica, you can use these parameters to set, change, and delete replication filters. When you set one of these parameters, separate each filter from others with a comma.

You can use the `%` and _ wildcard characters in the `replicate-wild-do-table` and `replicate-wild-ignore-table` parameters. The `%` wildcard matches any number of characters, and the _ wildcard matches only one character.

The binary logging format of the source DB instance is important for replication because it determines the record of data changes. The setting of the `binlog_format` parameter determines whether the replication is row-based or statement-based. For more information, see Configuring MariaDB binary logging.

> **Note**
>
> All data definition language (DDL) statements are replicated as statements, regardless of the `binlog_format` setting on the source DB instance.

Replication filtering limitations for RDS for MariaDB

The following limitations apply to replication filtering for RDS for MariaDB:

- Each replication filtering parameter has a 2,000-character limit.
- Commas aren't supported in replication filters.
- The MariaDB `binlog_do_db` and `binlog_ignore_db` options for binary log filtering aren't supported.
- Replication filtering doesn't support XA transactions.

  For more information, see Restrictions on XA Transactions in the MySQL documentation.
- Replication filtering isn't supported for RDS for MariaDB version 10.2.

Replication filtering examples for RDS for MariaDB

To configure replication filtering for a read replica, modify the replication filtering parameters in the parameter group associated with the read replica.

> **Note**
>
> You can't modify a default parameter group. If the read replica is using a default parameter group, create a new parameter group and associate it with the read replica. For more information on DB parameter groups, see Parameter groups for Amazon RDS.

You can set parameters in a parameter group using the AWS Management Console, AWS CLI, or RDS API. For information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS. When you set parameters in a parameter group, all of the DB instances associated with the parameter group use the parameter settings. If you set the replication filtering parameters in a parameter group, make sure that the parameter group is associated only with read replicas. Leave the replication filtering parameters empty for source DB instances.

The following examples set the parameters using the AWS CLI. These examples set `ApplyMethod` to `immediate` so that the parameter changes occur immediately after the CLI command completes. If you want a pending change to be applied after the read replica is rebooted, set `ApplyMethod` to `pending-reboot`.

The following examples set replication filters:

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters
- Escaping wildcard characters in names
- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example Including databases in replication

The following example includes the `mydb1` and `mydb2` databases in replication. When you set `replicate-do-db` for a read replica, only the databases specified in the parameter are replicated.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "[{"ParameterName": "replicate-do-db", "ParameterValue": "mydb1,mydb2",
 "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-do-db", "ParameterValue": "mydb1,mydb2",
 "ApplyMethod":"immediate"}]"
```

Example Including tables in replication

The following example includes the `table1` and `table2` tables in database `mydb1` in replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "[{"ParameterName": "replicate-do-table", "ParameterValue":
 "mydb1.table1,mydb1.table2", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-do-table", "ParameterValue":
 "mydb1.table1,mydb1.table2", "ApplyMethod":"immediate"}]"
```

Example Including tables in replication using wildcard characters

The following example includes tables with names that begin with `orders` and `returns` in database `mydb` in replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "[{"ParameterName": "replicate-wild-do-table", "ParameterValue":
 "mydb.orders%,mydb.returns%", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-wild-do-table", "ParameterValue":
 "mydb.orders%,mydb.returns%", "ApplyMethod":"immediate"}]"
```

Example Escaping wildcard characters in names

The following example shows you how to use the escape character \ to escape a wildcard character that is part of a name.

Assume that you have several table names in database `mydb1` that start with `my_table`, and you want to include these tables in replication. The table names include an underscore, which is also a wildcard character, so the example escapes the underscore in the table names.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "[{"ParameterName": "replicate-wild-do-table", "ParameterValue": "my
\_table%", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-wild-do-table", "ParameterValue": "my
\_table%", "ApplyMethod":"immediate"}]"
```

Example Excluding databases from replication

The following example excludes the `mydb1` and `mydb2` databases from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "[{"ParameterName": "replicate-ignore-db", "ParameterValue":
 "mydb1,mydb2", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-ignore-db", "ParameterValue":
 "mydb1,mydb2", "ApplyMethod":"immediate"}]"
```

Example Excluding tables from replication

The following example excludes tables `table1` and `table2` in database `mydb1` from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
```

```
  --parameters "[{"ParameterName": "replicate-ignore-table", "ParameterValue":
 "mydb1.table1,mydb1.table2", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-ignore-table", "ParameterValue":
 "mydb1.table1,mydb1.table2", "ApplyMethod":"immediate"}]"
```

Example Excluding tables from replication using wildcard characters

The following example excludes tables with names that begin with `orders` and `returns` in database `mydb` from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "[{"ParameterName": "replicate-wild-ignore-table", "ParameterValue":
 "mydb.orders%,mydb.returns%", "ApplyMethod":"immediate"}]"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "[{"ParameterName": "replicate-wild-ignore-table", "ParameterValue":
 "mydb.orders%,mydb.returns%", "ApplyMethod":"immediate"}]"
```

Viewing the replication filters for a read replica

You can view the replication filters for a read replica in the following ways:

- Check the settings of the replication filtering parameters in the parameter group associated with the read replica.

  For instructions, see Viewing parameter values for a DB parameter group in Amazon RDS.
- In a MariaDB client, connect to the read replica and run the `SHOW REPLICA STATUS` statement.

  In the output, the following fields show the replication filters for the read replica:
  - `Replicate_Do_DB`

- `Replicate_Ignore_DB`
- `Replicate_Do_Table`
- `Replicate_Ignore_Table`
- `Replicate_Wild_Do_Table`
- `Replicate_Wild_Ignore_Table`

For more information about these fields, see Checking Replication Status in the MySQL documentation.

> **Note**
>
> Previous versions of MariaDB used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MariaDB version before 10.5, then use `SHOW SLAVE STATUS`.

Configuring delayed replication with MariaDB

You can use delayed replication as a strategy for disaster recovery. With delayed replication, you specify the minimum amount of time, in seconds, to delay replication from the source to the read replica. In the event of a disaster, such as a table deleted unintentionally, you complete the following steps to recover from the disaster quickly:

- Stop replication to the read replica before the change that caused the disaster is sent to it.

  To stop replication, use the mysql.rds_stop_replication stored procedure.
- Promote the read replica to be the new source DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

> **Note**
>
> - Delayed replication is supported for MariaDB 10.6 and higher.
> - Use stored procedures to configure delayed replication. You can't configure delayed replication with the AWS Management Console, the AWS CLI, or the Amazon RDS API.
> - You can use replication based on global transaction identifiers (GTIDs) in a delayed replication configuration.

Topics

- Configuring delayed replication during read replica creation
- Modifying delayed replication for an existing read replica
- Promoting a read replica

Configuring delayed replication during read replica creation

To configure delayed replication for any future read replica created from a DB instance, run the mysql.rds_set_configuration stored procedure with the `target delay` parameter.

To configure delayed replication during read replica creation

1. Using a MariaDB client, connect to the MariaDB DB instance to be the source for read replicas as the master user.
2. Run the mysql.rds_set_configuration stored procedure with the `target delay` parameter.

   For example, run the following stored procedure to specify that replication is delayed by at least one hour (3,600 seconds) for any read replica created from the current DB instance.

   ```
   call mysql.rds_set_configuration('target delay', 3600);
   ```

   > **Note**
   >
   > After running this stored procedure, any read replica you create using the AWS CLI or Amazon RDS API is configured with replication delayed by the specified number of seconds.

Modifying delayed replication for an existing read replica

To modify delayed replication for an existing read replica, run the mysql.rds_set_source_delay stored procedure.

To modify delayed replication for an existing read replica

1. Using a MariaDB client, connect to the read replica as the master user.
2. Use the mysql.rds_stop_replication stored procedure to stop replication.
3. Run the mysql.rds_set_source_delay stored procedure.

For example, run the following stored procedure to specify that replication to the read replica is delayed by at least one hour (3600 seconds).

```
call mysql.rds_set_source_delay(3600);
```

4. Use the mysql.rds_start_replication stored procedure to start replication.

Promoting a read replica

After replication is stopped, in a disaster recovery scenario, you can promote a read replica to be the new source DB instance. For information about promoting a read replica, see Promoting a read replica to be a standalone DB instance.

Updating read replicas with MariaDB

Read replicas are designed to support read queries, but you might need occasional updates. For example, you might need to add an index to speed the specific types of queries accessing the replica. You can enable updates by setting the `read_only` parameter to **0** in the DB parameter group for the read replica.

Working with Multi-AZ read replica deployments with MariaDB

You can create a read replica from either single-AZ or Multi-AZ DB instance deployments. You use Multi-AZ deployments to improve the durability and availability of critical data, but you can't use the Multi-AZ secondary to serve read-only queries. Instead, you can create read replicas from high-traffic Multi-AZ DB instances to offload read-only queries. If the source instance of a Multi-AZ deployment fails over to the secondary, any associated read replicas automatically switch to use the secondary (now primary) as their replication source. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

You can create a read replica as a Multi-AZ DB instance. Amazon RDS creates a standby of your replica in another Availability Zone for failover support for the replica. Creating your read replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ DB instance.

Using cascading read replicas with RDS for MariaDB

RDS for MariaDB supports cascading read replicas. With *cascading read replicas*, you can scale reads without adding overhead to your source RDS for MariaDB DB instance.

With cascading read replicas, your RDS for MariaDB DB instance sends data to the first read replica in the chain. That read replica then sends data to the second replica in the chain, and so on. The end result is that all read replicas in the chain have the changes from the RDS for MariaDB DB instance, but without the overhead solely on the source DB instance.

You can create a series of up to three read replicas in a chain from a source RDS for MariaDB DB instance. For example, suppose that you have an RDS for MariaDB DB instance, `mariadb-main`. You can do the following:

- Starting with `mariadb-main`, create the first read replica in the chain, `read-replica-1`.
- Next, from `read-replica-1`, create the next read replica in the chain, `read-replica-2`.
- Finally, from `read-replica-2`, create the third read replica in the chain, `read-replica-3`.

You can't create another read replica beyond this third cascading read replica in the series for `mariadb-main`. A complete series of instances from an RDS for MariaDB source DB instance through to the end of a series of cascading read replicas can consist of at most four DB instances.

For cascading read replicas to work, each source RDS for MariaDB DB instance must have automated backups turned on. To turn on automatic backups on a read replica, first create the read replica, and then modify the read replica to turn on automatic backups. For more information, see Creating a read replica.

As with any read replica, you can promote a read replica that's part of a cascade. Promoting a read replica from within a chain of read replicas removes that replica from the chain. For example, suppose that you want to move some of the workload from your `mariadb-main` DB instance to a new instance for use by the accounting department only. Assuming the chain of three read replicas from the example, you decide to promote `read-replica-2`. The chain is affected as follows:

- Promoting `read-replica-2` removes it from the replication chain.
  - It is now a full read/write DB instance.
  - It continues replicating to `read-replica-3`, just as it was doing before promotion.
- Your `mariadb-main` continues replicating to `read-replica-1`.

For more information about promoting read replicas, see Promoting a read replica to be a standalone DB instance.

Monitoring MariaDB read replicas

For MariaDB read replicas, you can monitor replication lag in Amazon CloudWatch by viewing the Amazon RDS `ReplicaLag` metric. The `ReplicaLag` metric reports the value of the `Seconds_Behind_Master` field of the `SHOW REPLICA STATUS` command.

> **Note**
>
> Previous versions of MariaDB used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MariaDB version before 10.5, then use `SHOW SLAVE STATUS`.

Common causes for replication lag for MariaDB are the following:

- A network outage.
- Writing to tables with indexes on a read replica. If the `read_only` parameter is not set to 0 on the read replica, it can break replication.
- Using a nontransactional storage engine such as MyISAM. Replication is only supported for the InnoDB storage engine on MariaDB.

When the `ReplicaLag` metric reaches 0, the replica has caught up to the source DB instance. If the `ReplicaLag` metric returns -1, then replication is currently not active. `ReplicaLag` = -1 is equivalent to `Seconds_Behind_Master` = `NULL`.

Starting and stopping replication with MariaDB read replicas

You can stop and restart the replication process on an Amazon RDS DB instance by calling the system stored procedures mysql.rds_stop_replication and mysql.rds_start_replication. You can do this when replicating between two Amazon RDS instances for long-running operations such as creating large indexes. You also need to stop and start replication when importing or exporting databases. For more information, see Importing data to an Amazon RDS for MariaDB DB instance with reduced downtime and Exporting data from a MySQL DB instance by using replication.

If replication is stopped for more than 30 consecutive days, either manually or due to a replication error, Amazon RDS ends replication between the source DB instance and all read replicas. It does so to prevent increased storage requirements on the source DB instance and long failover times. The read replica DB instance is still available. However, replication can't be resumed because the binary

logs required by the read replica are deleted from the source DB instance after replication is ended. You can create a new read replica for the source DB instance to reestablish replication.

Troubleshooting a MariaDB read replica problem

The replication technologies for MariaDB are asynchronous. Because they are asynchronous, occasional `BinLogDiskUsage` increases on the source DB instance and `ReplicaLag` on the read replica are to be expected. For example, a high volume of write operations to the source DB instance can occur in parallel. In contrast, write operations to the read replica are serialized using a single I/O thread, which can lead to a lag between the source instance and read replica. For more information about read-only replicas in the MariaDB documentation, go to Replication overview.

You can do several things to reduce the lag between updates to a source DB instance and the subsequent updates to the read replica, such as the following:

- Sizing a read replica to have a storage size and DB instance class comparable to the source DB instance.
- Ensuring that parameter settings in the DB parameter groups used by the source DB instance and the read replica are compatible. For more information and an example, see the discussion of the `max_allowed_packet` parameter later in this section.

Amazon RDS monitors the replication status of your read replicas and updates the `Replication State` field of the read replica instance to `Error` if replication stops for any reason. An example might be if DML queries run on your read replica conflict with the updates made on the source DB instance.

You can review the details of the associated error thrown by the MariaDB engine by viewing the `Replication Error` field. Events that indicate the status of the read replica are also generated, including RDS-EVENT-0045, RDS-EVENT-0046, and RDS-EVENT-0047. For more information about events and subscribing to events, see Working with Amazon RDS event notification. If a MariaDB error message is returned, review the error in the MariaDB error message documentation.

One common issue that can cause replication errors is when the value for the `max_allowed_packet` parameter for a read replica is less than the `max_allowed_packet` parameter for the source DB instance. The `max_allowed_packet` parameter is a custom parameter that you can set in a DB parameter group that is used to specify the maximum size of DML code that can be run on the database. In some cases, the `max_allowed_packet` parameter value in the DB parameter group associated with a source DB instance is smaller than

the `max_allowed_packet` parameter value in the DB parameter group associated with the source's read replica. In these cases, the replication process can throw an error (Packet bigger than 'max_allowed_packet' bytes) and stop replication. You can fix the error by having the source and read replica use DB parameter groups with the same `max_allowed_packet` parameter values.

Other common situations that can cause replication errors include the following:

- Writing to tables on a read replica. If you are creating indexes on a read replica, you need to have the `read_only` parameter set to **0** to create the indexes. If you are writing to tables on the read replica, it might break replication.
- Using a non-transactional storage engine such as MyISAM. read replicas require a transactional storage engine. Replication is only supported for the InnoDB storage engine on MariaDB.
- Using unsafe nondeterministic queries such as `SYSDATE()`. For more information, see Determination of safe and unsafe statements in binary logging.

If you decide that you can safely skip an error, you can follow the steps described in Skipping the current replication error for RDS for MySQL. Otherwise, you can delete the read replica and create an instance using the same DB instance identifier so that the endpoint remains the same as that of your old read replica. If a replication error is fixed, the `Replication State` changes to *replicating*.

For MariaDB DB instances, in some cases read replicas can't be switched to the secondary if some binary log (binlog) events aren't flushed during the failure. In these cases, manually delete and recreate the read replicas. You can reduce the chance of this happening by setting the following parameter values: `sync_binlog=1` and `innodb_flush_log_at_trx_commit=1`. These settings might reduce performance, so test their impact before implementing the changes in a production environment.

Configuring GTID-based replication with an external source instance

You can set up replication based on global transaction identifiers (GTIDs) from an external MariaDB instance of version 10.0.24 or higher into an RDS for MariaDB DB instance. Follow these guidelines when you set up an external source instance and a replica on Amazon RDS:

- Monitor failover events for the RDS for MariaDB DB instance that is your replica. If a failover occurs, then the DB instance that is your replica might be recreated on a new host with a different network address. For information on how to monitor failover events, see Working with Amazon RDS event notification.

- Maintain the binary logs (binlogs) on your source instance until you have verified that they have been applied to the replica. This maintenance ensures that you can restore your source instance in the event of a failure.
- Turn on automated backups on your MariaDB DB instance on Amazon RDS. Turning on automated backups ensures that you can restore your replica to a particular point in time if you need to resynchronize your source instance and replica. For information on backups and Point-In-Time Restore, see Backing up, restoring, and exporting data.

> **Note**
>
> The permissions required to start replication on a MariaDB DB instance are restricted and not available to your Amazon RDS master user. Because of this, you must use the Amazon RDS mysql.rds_set_external_master_gtid and mysql.rds_start_replication commands to set up replication between your live database and your RDS for MariaDB database.

To start replication between an external source instance and a MariaDB DB instance on Amazon RDS, use the following procedure.

To start replication

1. Make the source MariaDB instance read-only:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   mysql> SET GLOBAL read_only = ON;
   ```

2. Get the current GTID of the external MariaDB instance. You can do this by using `mysql` or the query editor of your choice to run `SELECT @@gtid_current_pos;`.

   The GTID is formatted as `<domain-id>-<server-id>-<sequence-id>`. A typical GTID looks something like **`0-1234510749-1728`**. For more information about GTIDs and their component parts, see Global transaction ID in the MariaDB documentation.

3. Copy the database from the external MariaDB instance to the MariaDB DB instance using `mysqldump`. For very large databases, you might want to use the procedure in Importing data to an Amazon RDS for MariaDB DB instance with reduced downtime.

   For Linux, macOS, or Unix:

```
mysqldump \
    --databases database_name \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -plocal_password | mysql \
        --host=hostname \
        --port=3306 \
        -u RDS_user_name \
        -pRDS_password
```

For Windows:

```
mysqldump ^
    --databases database_name ^
    --single-transaction ^
    --compress ^
    --order-by-primary \
    -u local_user \
    -plocal_password | mysql ^
        --host=hostname ^
        --port=3306 ^
        -u RDS_user_name ^
        -pRDS_password
```

> **Note**
>
> Make sure that there isn't a space between the `-p` option and the entered password. Specify a password other than the prompt shown here as a security best practice.

Use the `--host`, `--user (-u)`, `--port` and `-p` options in the `mysql` command to specify the host name, user name, port, and password to connect to your MariaDB DB instance. The host name is the DNS name from the MariaDB DB instance endpoint, for example `myinstance.123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the instance details in the Amazon RDS Management Console.

4. Make the source MariaDB instance writeable again.

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

5. In the Amazon RDS Management Console, add the IP address of the server that hosts the external MariaDB database to the VPC security group for the MariaDB DB instance. For more information on modifying a VPC security group, go to Security groups for your VPC in the *Amazon Virtual Private Cloud User Guide*.

   The IP address can change when the following conditions are met:

   - You are using a public IP address for communication between the external source instance and the DB instance.
   - The external source instance was stopped and restarted.

   If these conditions are met, verify the IP address before adding it.

   You might also need to configure your local network to permit connections from the IP address of your MariaDB DB instance, so that it can communicate with your external MariaDB instance. To find the IP address of the MariaDB DB instance, use the `host` command.

   ```
   host db_instance_endpoint
   ```

   The host name is the DNS name from the MariaDB DB instance endpoint.

6. Using the client of your choice, connect to the external MariaDB instance and create a MariaDB user to be used for replication. This account is used solely for replication and must be restricted to your domain to improve security. The following is an example.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   > **Note**
   >
   > Specify a password other than the prompt shown here as a security best practice.

7. For the external MariaDB instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. For example, to grant the `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the '`repl_user`' user for your domain, issue the following command.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

8. Make the MariaDB DB instance the replica. Connect to the MariaDB DB instance as the master user and identify the external MariaDB database as the replication source instance by using the mysql.rds_set_external_master_gtid command. Use the GTID that you determined in Step 2. The following is an example.

```
CALL mysql.rds_set_external_master_gtid ('mymasterserver.mydomain.com', 3306,
 'repl_user', 'password', 'GTID', 1);
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

9. On the MariaDB DB instance, issue the mysql.rds_start_replication command to start replication.

```
CALL mysql.rds_start_replication;
```

Configuring binary log file position replication with an external source instance

You can set up replication between an RDS for MySQL or MariaDB DB instance and a MySQL or MariaDB instance that is external to Amazon RDS using binary log file replication.

Topics

- Before you begin
- Configuring binary log file position replication with an external source instance

Before you begin

You can configure replication using the binary log file position of replicated transactions.

The permissions required to start replication on an Amazon RDS DB instance are restricted and not available to your Amazon RDS master user. Because of this, make sure that you use the Amazon

RDS mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) or mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher), and mysql.rds_start_replication commands to set up replication between your live database and your Amazon RDS database.

To set the binary logging format for a MySQL or MariaDB database, update the `binlog_format` parameter. If your DB instance uses the default DB instance parameter group, create a new DB parameter group to modify the `binlog_format` parameter. In MariaDB and MySQL 8.0 and lower versions, `binlog_format` defaults to `MIXED`. However, you can also set `binlog_format` to `ROW` or `STATEMENT` if you need a specific binary log (binlog) format. Reboot your DB instance for the change to take effect. In MySQL 8.4 and higher versions, `binlog_format` defaults to `ROW`.

For information about setting the `binlog_format` parameter, see Configuring RDS for MySQL binary logging for Single-AZ databases. For information about the implications of different MySQL replication types, see Advantages and disadvantages of statement-based and row-based replication in the MySQL documentation.

Configuring binary log file position replication with an external source instance

Follow these guidelines when you set up an external source instance and a replica on Amazon RDS:

- Monitor failover events for the Amazon RDS DB instance that is your replica. If a failover occurs, then the DB instance that is your replica might be recreated on a new host with a different network address. For information on how to monitor failover events, see Working with Amazon RDS event notification.
- Maintain the binlogs on your source instance until you have verified that they have been applied to the replica. This maintenance makes sure that you can restore your source instance in the event of a failure.
- Turn on automated backups on your Amazon RDS DB instance. Turning on automated backups makes sure that you can restore your replica to a particular point in time if you need to re-synchronize your source instance and replica. For information on backups and point-in-time restore, see Backing up, restoring, and exporting data.

To configure binary log file replication with an external source instance

1. Make the source MySQL or MariaDB instance read-only.

```
mysql> FLUSH TABLES WITH READ LOCK;
```

```
mysql> SET GLOBAL read_only = ON;
```

2. Run the `SHOW MASTER STATUS` command on the source MySQL or MariaDB instance to determine the binlog location.

   You receive output similar to the following example.

```
File                        Position
-------------------------------------
 mysql-bin-changelog.000031      107
-------------------------------------
```

3. Copy the database from the external instance to the Amazon RDS DB instance using `mysqldump`. For very large databases, you might want to use the procedure in [Importing data to an Amazon RDS for MySQL database with reduced downtime].

   For Linux, macOS, or Unix:

```
mysqldump --databases database_name \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -plocal_password | mysql \
        --host=hostname \
        --port=3306 \
        -u RDS_user_name \
        -pRDS_password
```

   For Windows:

```
mysqldump --databases database_name ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -plocal_password | mysql ^
        --host=hostname ^
        --port=3306 ^
        -u RDS_user_name ^
        -pRDS_password
```

> **Note**
>
> Make sure that there isn't a space between the `-p` option and the entered password.

To specify the host name, user name, port, and password to connect to your Amazon RDS DB instance, use the `--host`, `--user (-u)`, `--port` and `-p` options in the `mysql` command. The host name is the Domain Name Service (DNS) name from the Amazon RDS DB instance endpoint, for example `myinstance.123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the instance details in the AWS Management Console.

4. Make the source MySQL or MariaDB instance writeable again.

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

For more information on making backups for use with replication, see the MySQL documentation.

5. In the AWS Management Console, add the IP address of the server that hosts the external database to the virtual private cloud (VPC) security group for the Amazon RDS DB instance. For more information on modifying a VPC security group, see Security groups for your VPC in the *Amazon Virtual Private Cloud User Guide*.

   The IP address can change when the following conditions are met:

   - You are using a public IP address for communication between the external source instance and the DB instance.
   - The external source instance was stopped and restarted.

   If these conditions are met, verify the IP address before adding it.

   You might also need to configure your local network to permit connections from the IP address of your Amazon RDS DB instance. You do this so that your local network can communicate with your external MySQL or MariaDB instance. To find the IP address of the Amazon RDS DB instance, use the `host` command.

```
host db_instance_endpoint
```

The host name is the DNS name from the Amazon RDS DB instance endpoint.

6. Using the client of your choice, connect to the external instance and create a user to use for replication. Use this account solely for replication and restrict it to your domain to improve security. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

7. For the external instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. For example, to grant the `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the '`repl_user`' user for your domain, issue the following command.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

8. Make the Amazon RDS DB instance the replica. To do so, first connect to the Amazon RDS DB instance as the master user. Then identify the external MySQL or MariaDB database as the source instance by using the mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher) or mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) command. Use the master log file name and master log position that you determined in step 2. The following commands are examples.

MySQL 8.4

```
CALL mysql.rds_set_external_source ('mysourceserver.mydomain.com', 3306,
 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

MariaDB and MySQL 8.0 and 5.7

```
CALL mysql.rds_set_external_master ('mymasterserver.mydomain.com', 3306,
 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

> **Note**
>
> On RDS for MySQL, you can choose to use delayed replication by running the mysql.rds_set_external_source_with_delay (RDS for MySQL major versions 8.4 and higher) or mysql.rds_set_external_master_with_delay (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) stored procedure instead. On RDS for MySQL, one reason to use delayed replication is to turn on disaster recovery with the mysql.rds_start_replication_until stored procedure. Currently, RDS for MariaDB supports delayed replication but doesn't support the `mysql.rds_start_replication_until` procedure.

9. On the Amazon RDS DB instance, issue the mysql.rds_start_replication command to start replication.

```
CALL mysql.rds_start_replication;
```

Options for MariaDB database engine

Following, you can find descriptions for options, or additional features, that are available for Amazon RDS instances running the MariaDB DB engine. To turn on these options, you add them to a custom option group, and then associate the option group with your DB instance. For more information about working with option groups, see Working with option groups.

Amazon RDS supports the following options for MariaDB:

Option ID	Engine versions
MARIADB_AUDIT_PLUGIN	MariaDB 10.3 and higher

MariaDB Audit Plugin support

Amazon RDS supports using the MariaDB Audit Plugin on MariaDB database instances. The MariaDB Audit Plugin records database activity such as users logging on to the database, queries run against the database, and more. The record of database activity is stored in a log file.

Audit Plugin option settings

Amazon RDS supports the following settings for the MariaDB Audit Plugin option.

> **Note**
>
> If you don't configure an option setting in the RDS console, RDS uses the default setting.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_FILE_PATH	/rdsdbdata/log/audit/	/rdsdbdata/log/audit/	The location of the log file. The log file contains the record of the activity specified in SERVER_AUDIT_EVENTS . For more information, see Viewing and listing database log files and MariaDB database log files.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_FILE_ROTATE_SIZE	1–1000000000	1000000	The size in bytes that when reached, causes the file to rotate. For more information, see Log rotation and retention for MariaDB.
SERVER_AUDIT_FILE_ROTATIONS	0–100	9	The number of log rotations to save when `server_audit_output_type=file` . If set to 0, then the log file never rotates. For more information, see Log rotation and retention for MariaDB and Downloading a database log file.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_EVENTS	CONNECT, QUERY, TABLE, QUERY_DDL , QUERY_DML , QUERY_DML_NO_SELECT , QUERY_DCL	CONNECT, QUERY	The types of activity to record in the log. Installing the MariaDB Audit Plugin is itself logged. • CONNECT: Log successful and unsuccessful connections to the database, and disconnections from the database. • QUERY: Log the text of all queries run against the database. • TABLE: Log tables affected by queries when the queries are run against the database. • QUERY_DDL : Similar to the QUERY event, but returns only data definition language (DDL) queries (CREATE, ALTER, and so on). • QUERY_DML : Similar to the QUERY event, but returns only data manipulation language (DML) queries (INSERT, UPDATE, and so on, and also SELECT). • QUERY_DML_NO_SELECT : Similar to the QUERY_DML event, but doesn't log SELECT queries. • QUERY_DCL : Similar to the QUERY event, but returns only data control language (DCL) queries (GRANT, REVOKE, and so on).
SERVER_AUDIT_INCL_USERS	Multiple comma-separated values	None	Include only activity from the specified users. By default, activity is recorded for all users. SERVER_AUDIT_INCL_USERS and SERVER_AUDIT_EXCL_USERS are mutually exclusive. If you add values to SERVER_AUDIT_INCL_USERS , make sure no values are added to SERVER_AUDIT_EXCL_USERS .

Option setting	Valid values	Default value	Description
SERVER_AUDIT_EXCL_USERS	Multiple comma-separated values	None	Exclude activity from the specified users. By default, activity is recorded for all users. SERVER_AUDIT_INCL_USERS and SERVER_AUDIT_EXCL_USERS are mutually exclusive. If you add values to SERVER_AUDIT_EXCL_USERS, make sure no values are added to SERVER_AUDIT_INCL_USERS. The rdsadmin user queries the database every second to check the health of the database. Depending on your other settings, this activity can possibly cause the size of your log file to grow very large, very quickly. If you don't need to record this activity, add the rdsadmin user to the SERVER_AUDIT_EXCL_USERS list. **Note** CONNECT activity is always recorded for all users, even if the user is specified for this option setting.
SERVER_AUDIT_LOGGING	ON	ON	Logging is active. The only valid value is ON. Amazon RDS does not support deactivating logging. If you want to deactivate logging, remove the MariaDB Audit Plugin. For more information, see Removing the MariaDB Audit Plugin.
SERVER_AUDIT_QUERY_LOG_LIMIT	0–2147483647	1024	The limit on the length of the query string in a record.

Adding the MariaDB Audit Plugin

The general process for adding the MariaDB Audit Plugin to a DB instance is the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with the DB instance.

After you add the MariaDB Audit Plugin, you don't need to restart your DB instance. As soon as the option group is active, auditing begins immediately.

To add the MariaDB Audit Plugin

1. Determine the option group you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group. Choose **mariadb** for **Engine**, and choose **10.3** or higher for **Major engine version**. For more information, see Creating an option group.
2. Add the **MARIADB_AUDIT_PLUGIN** option to the option group, and configure the option settings. For more information about adding options, see Adding an option to an option group. For more information about each setting, see Audit Plugin option settings.
3. Apply the option group to a new or existing DB instance.
   - For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, you apply the option group by modifying the DB instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

Viewing and downloading the MariaDB Audit Plugin log

After you enable the MariaDB Audit Plugin, you access the results in the log files the same way you access any other text-based log files. The audit log files are located at `/rdsdbdata/log/audit/`. For information about viewing the log file in the console, see Viewing and listing database log files. For information about downloading the log file, see Downloading a database log file.

Modifying MariaDB Audit Plugin settings

After you enable the MariaDB Audit Plugin, you can modify settings for the plugin. For more information about how to modify option settings, see Modifying an option setting. For more information about each setting, see Audit Plugin option settings.

Removing the MariaDB Audit Plugin

Amazon RDS doesn't support turning off logging in the MariaDB Audit Plugin. However, you can remove the plugin from a DB instance. When you remove the MariaDB Audit Plugin, the DB instance is restarted automatically to stop auditing.

To remove the MariaDB Audit Plugin from a DB instance, do one of the following:

- Remove the MariaDB Audit Plugin option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group
- Modify the DB instance and specify a different option group that doesn't include the plugin. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Parameters for MariaDB

By default, a MariaDB DB instance uses a DB parameter group that is specific to a MariaDB database. This parameter group contains some but not all of the parameters contained in the Amazon RDS DB parameter groups for the MySQL database engine. It also contains a number of new, MariaDB-specific parameters. For information about working with parameter groups and setting parameters, see Parameter groups for Amazon RDS.

Viewing MariaDB parameters

RDS for MariaDB parameters are set to the default values of the storage engine that you have selected. For more information about MariaDB parameters, see the MariaDB documentation. For more information about MariaDB storage engines, see Supported storage engines for MariaDB on Amazon RDS.

You can view the parameters available for a specific RDS for MariaDB version using the RDS console or the AWS CLI. For information about viewing the parameters in a MariaDB parameter group in the RDS console, see Viewing parameter values for a DB parameter group in Amazon RDS.

Using the AWS CLI, you can view the parameters for an RDS for MariaDB version by running the `describe-engine-default-parameters` command. Specify one of the following values for the `--db-parameter-group-family` option:

- `mariadb11.8`
- `mariadb11.4`
- `mariadb10.11`
- `mariadb10.6`
- `mariadb10.5`
- `mariadb10.4`
- `mariadb10.3`

For example, to view the parameters for RDS for MariaDB version 10.6, run the following command.

```
aws rds describe-engine-default-parameters --db-parameter-group-family mariadb10.6
```

Your output looks similar to the following.

```
{
    "EngineDefaults": {
        "Parameters": [
            {
                "ParameterName": "alter_algorithm",
                "Description": "Specify the alter table algorithm.",
                "Source": "engine-default",
                "ApplyType": "dynamic",
                "DataType": "string",
                "AllowedValues": "DEFAULT,COPY,INPLACE,NOCOPY,INSTANT",
                "IsModifiable": true
            },
            {
                "ParameterName": "analyze_sample_percentage",
                "Description": "Percentage of rows from the table ANALYZE TABLE will
 sample to collect table statistics.",
                "Source": "engine-default",
                "ApplyType": "dynamic",
                "DataType": "float",
                "AllowedValues": "0-100",
                "IsModifiable": true
            },
            {
                "ParameterName": "aria_block_size",
                "Description": "Block size to be used for Aria index pages.",
                "Source": "engine-default",
                "ApplyType": "static",
                "DataType": "integer",
                "AllowedValues": "1024-32768",
                "IsModifiable": false
            },
            {
                "ParameterName": "aria_checkpoint_interval",
                "Description": "Interval in seconds between automatic checkpoints.",
                "Source": "engine-default",
                "ApplyType": "dynamic",
                "DataType": "integer",
                "AllowedValues": "0-4294967295",
                "IsModifiable": true
            },
        ...
```

To list only the modifiable parameters for RDS for MariaDB version 10.6, run the following command.

For Linux, macOS, or Unix:

```
aws rds describe-engine-default-parameters --db-parameter-group-family mariadb10.6 \
   --query 'EngineDefaults.Parameters[?IsModifiable==`true`]'
```

For Windows:

```
aws rds describe-engine-default-parameters --db-parameter-group-family mariadb10.6 ^
   --query "EngineDefaults.Parameters[?IsModifiable==`true`]"
```

MySQL parameters that aren't available

The following MySQL parameters are not available in MariaDB-specific DB parameter groups:

- bind_address
- binlog_error_action
- binlog_gtid_simple_recovery
- binlog_max_flush_queue_time
- binlog_order_commits
- binlog_row_image
- binlog_rows_query_log_events
- binlogging_impossible_mode
- block_encryption_mode
- core_file
- default_tmp_storage_engine
- div_precision_increment
- end_markers_in_json
- enforce_gtid_consistency
- eq_range_index_dive_limit
- explicit_defaults_for_timestamp
- gtid_executed

- gtid-mode
- gtid_next
- gtid_owned
- gtid_purged
- log_bin_basename
- log_bin_index
- log_bin_use_v1_row_events
- log_slow_admin_statements
- log_slow_slave_statements
- log_throttle_queries_not_using_indexes
- master-info-repository
- optimizer_trace
- optimizer_trace_features
- optimizer_trace_limit
- optimizer_trace_max_mem_size
- optimizer_trace_offset
- relay_log_info_repository
- rpl_stop_slave_timeout
- slave_parallel_workers
- slave_pending_jobs_size_max
- slave_rows_search_algorithms
- storage_engine
- table_open_cache_instances
- timed_mutexes
- transaction_allow_batching
- validate-password
- validate_password_dictionary_file
- validate_password_length
- validate_password_mixed_case_count
- validate_password_number_count

- validate_password_policy
- validate_password_special_char_count

For more information on MySQL parameters, see the MySQL documentation.

Migrating data from a MySQL DB snapshot to a MariaDB DB instance

You can migrate an RDS for MySQL DB snapshot to a new DB instance running MariaDB using the AWS Management Console, the AWS CLI, or Amazon RDS API. You must use a DB snapshot that was created from an Amazon RDS DB instance running MySQL 5.6 or 5.7. To learn how to create an RDS for MySQL DB snapshot, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Migrating the snapshot doesn't affect the original DB instance from which the snapshot was taken. You can test and validate the new DB instance before diverting traffic to it as a replacement for the original DB instance.

After you migrate from MySQL to MariaDB, the MariaDB DB instance is associated with the default DB parameter group and option group. After you restore the DB snapshot, you can associate a custom DB parameter group with the new DB instance. However, a MariaDB parameter group has a different set of configurable system variables. For information about the differences between MySQL and MariaDB system variables, see System Variable Differences between MariaDB and MySQL. To learn about DB parameter groups, see Parameter groups for Amazon RDS. To learn about option groups, see Working with option groups.

Performing the migration

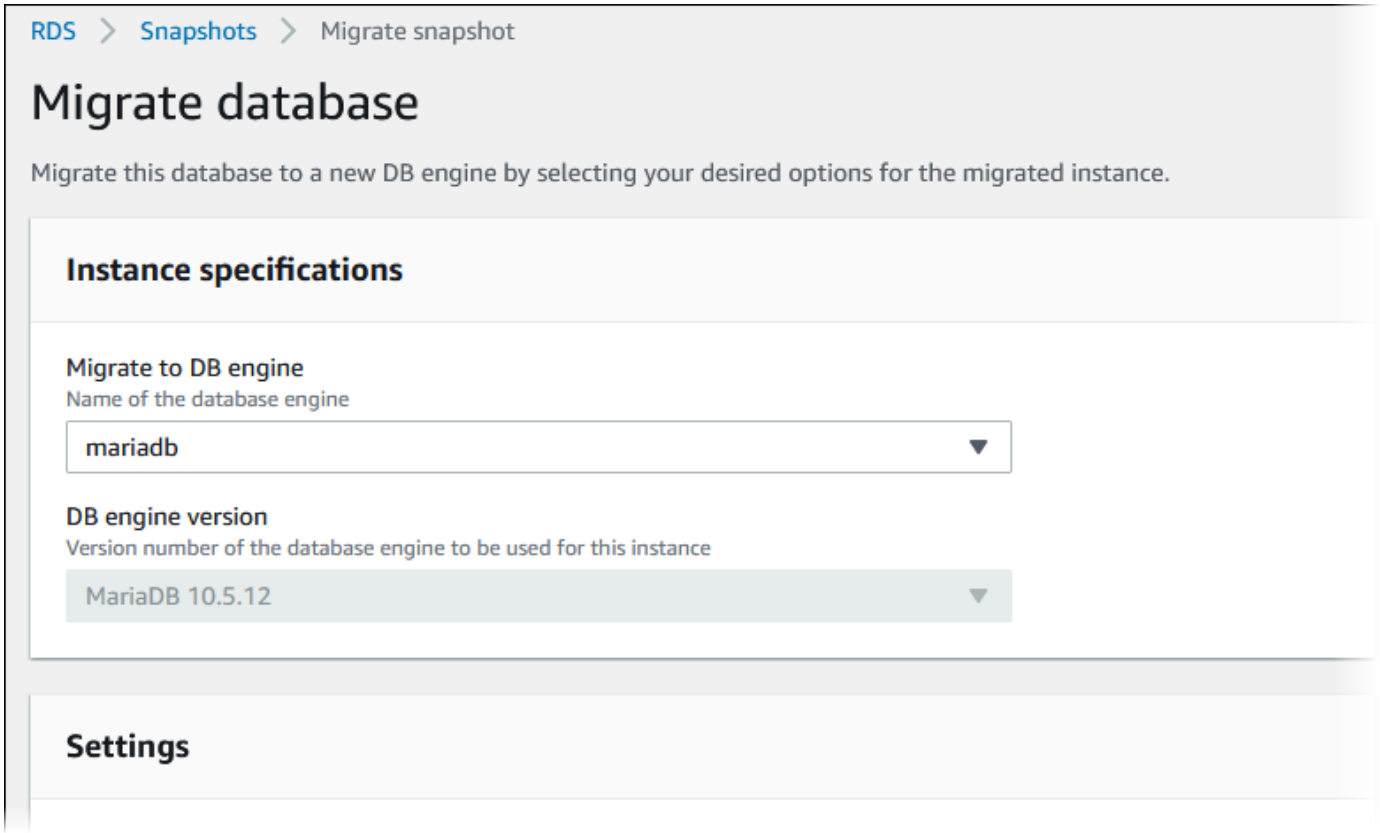
You can migrate an RDS for MySQL DB snapshot to a new MariaDB DB instance using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To migrate a MySQL DB snapshot to a MariaDB DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**, and then select the MySQL DB snapshot you want to migrate.
3. For **Actions**, choose **Migrate snapshot**. The **Migrate database** page appears.
4. For **Migrate to DB Engine**, choose **mariadb**.

   Amazon RDS selects the **DB engine version** automatically. You can't change the DB engine version.



5. For the remaining sections, specify your DB instance settings. For information about each setting, see Settings for DB instances.
6. Choose **Migrate**.

AWS CLI

To migrate data from a MySQL DB snapshot to a MariaDB DB instance, run the AWS CLI `restore-db-instance-from-db-snapshot` command with the following options:

- --db-instance-identifier – Name of the DB instance to create from the DB snapshot.
- --db-snapshot-identifier – The identifier for the DB snapshot to restore from.
- --engine – The database engine to use for the new instance.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier newmariadbinstance \
```

```
    --db-snapshot-identifier mysqlsnapshot \
    --engine mariadb
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^
    --db-instance-identifier newmariadbinstance ^
    --db-snapshot-identifier mysqlsnapshot ^
    --engine mariadb
```

API

To migrate data from a MySQL DB snapshot to a MariaDB DB instance, call the Amazon RDS API operation `RestoreDBInstanceFromDBSnapshot`.

Incompatibilities between MariaDB and MySQL

Incompatibilities between MySQL and MariaDB include the following:

- You can't migrate a DB snapshot created with MySQL 8.0 to MariaDB.
- If the source MySQL database uses a SHA256 password hash, make sure to reset user passwords that are SHA256 hashed before you connect to the MariaDB database. The following code shows how to reset a password that is SHA256 hashed.

  ```
  SET old_passwords = 0;
  UPDATE mysql.user SET plugin = 'mysql_native_password',
  Password = PASSWORD('new_password')
  WHERE (User, Host) = ('master_user_name', %);
  FLUSH PRIVILEGES;
  ```

- If your RDS master user account uses the SHA-256 password hash, make sure to reset the password using the AWS Management Console, the `modify-db-instance` AWS CLI command, or the ModifyDBInstance RDS API operation. For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.
- MariaDB doesn't support the Memcached plugin. However, the data used by the Memcached plugin is stored as InnoDB tables. After you migrate a MySQL DB snapshot, you can access the data used by the Memcached plugin using SQL. For more information about the innodb_memcache database, see InnoDB memcached Plugin Internals.

MariaDB on Amazon RDS SQL reference

Following, you can find descriptions of system stored procedures that are available for Amazon RDS instances running the MariaDB DB engine.

You can use the system stored procedures that are available for MySQL DB instances and MariaDB DB instances. These stored procedures are documented at RDS for MySQL stored procedure reference. MariaDB DB instances support all of the stored procedures, except for `mysql.rds_start_replication_until` and `mysql.rds_start_replication_until_gtid`.

Additionally, the following system stored procedures are supported only for Amazon RDS DB instances running MariaDB:

- mysql.rds_replica_status
- mysql.rds_set_external_master_gtid
- mysql.rds_kill_query_id
- mysql.rds_execute_operation

mysql.rds_replica_status

Shows the replication status of a MariaDB read replica.

Call this procedure on the read replica to show status information on essential parameters of the replica threads.

Syntax

```
CALL mysql.rds_replica_status;
```

Usage notes

This procedure is only supported for MariaDB DB instances running MariaDB version 10.5 and higher.

This procedure is the equivalent of the `SHOW REPLICA STATUS` command. This command isn't supported for MariaDB version 10.5 and higher DB instances.

In prior versions of MariaDB, the equivalent `SHOW SLAVE STATUS` command required the `REPLICATION SLAVE` privilege. In MariaDB version 10.5 and higher, it requires the `REPLICATION REPLICA ADMIN` privilege. To protect the RDS management of MariaDB 10.5 and higher DB instances, this new privilege isn't granted to the RDS master user.

Examples

The following example shows the status of a MariaDB read replica:

```
call mysql.rds_replica_status;
```

The response is similar to the following:

```
*************************** 1. row ***************************
               Replica_IO_State: Waiting for master to send event
                    Source_Host: XX.XX.XX.XXX
                    Source_User: rdsrepladmin
                    Source_Port: 3306
                  Connect_Retry: 60
                Source_Log_File: mysql-bin-changelog.003988
            Read_Source_Log_Pos: 405
                 Relay_Log_File: relaylog.011024
                  Relay_Log_Pos: 657
          Relay_Source_Log_File: mysql-bin-changelog.003988
             Replica_IO_Running: Yes
            Replica_SQL_Running: Yes
                Replicate_Do_DB:
            Replicate_Ignore_DB:
             Replicate_Do_Table:
         Replicate_Ignore_Table:
mysql.rds_sysinfo,mysql.rds_history,mysql.rds_replication_status
        Replicate_Wild_Do_Table:
    Replicate_Wild_Ignore_Table:
                     Last_Errno: 0
                     Last_Error:
                   Skip_Counter: 0
            Exec_Source_Log_Pos: 405
                Relay_Log_Space: 1016
                Until_Condition: None
                 Until_Log_File:
                  Until_Log_Pos: 0
             Source_SSL_Allowed: No
```

```
              Source_SSL_CA_File:
              Source_SSL_CA_Path:
                 Source_SSL_Cert:
               Source_SSL_Cipher:
                  Source_SSL_Key:
          Seconds_Behind_Master: 0
  Source_SSL_Verify_Server_Cert: No
                  Last_IO_Errno: 0
                  Last_IO_Error:
                 Last_SQL_Errno: 0
                 Last_SQL_Error:
    Replicate_Ignore_Server_Ids:
               Source_Server_Id: 807509301
                 Source_SSL_Crl:
             Source_SSL_Crlpath:
                     Using_Gtid: Slave_Pos
                   Gtid_IO_Pos: 0-807509301-3980
        Replicate_Do_Domain_Ids:
    Replicate_Ignore_Domain_Ids:
                  Parallel_Mode: optimistic
                      SQL_Delay: 0
            SQL_Remaining_Delay: NULL
      Replica_SQL_Running_State: Reading event from the relay log
             Replica_DDL_Groups: 15
Replica_Non_Transactional_Groups: 0
    Replica_Transactional_Groups: 3658
1 row in set (0.000 sec)

Query OK, 0 rows affected (0.000 sec)
```

mysql.rds_set_external_master_gtid

Configures GTID-based replication from a MariaDB instance running external to Amazon RDS to a MariaDB DB instance. This stored procedure is supported only where the external MariaDB instance is version 10.0.24 or higher. When setting up replication where one or both instances do not support MariaDB global transaction identifiers (GTIDs), use mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower).

Using GTIDs for replication provides crash-safety features not offered by binary log replication, so we recommend it in cases where the replicating instances support it.

Syntax

```
CALL mysql.rds_set_external_master_gtid(
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , gtid
  , ssl_encryption
);
```

Parameters

host_name

String. The host name or IP address of the MariaDB instance running external to Amazon RDS that will become the source instance.

host_port

Integer. The port used by the MariaDB instance running external to Amazon RDS to be configured as the source instance. If your network configuration includes SSH port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

String. The ID of a user with `REPLICATION SLAVE` permissions in the MariaDB DB instance to be configured as the read replica.

replication_user_password

String. The password of the user ID specified in `replication_user_name`.

gtid

String. The global transaction ID on the source instance that replication should start from.

You can use `@@gtid_current_pos` to get the current GTID if the source instance has been locked while you are configuring replication, so the binary log doesn't change between the points when you get the GTID and when replication starts.

Otherwise, if you are using `mysqldump` version 10.0.13 or greater to populate the replica instance prior to starting replication, you can get the GTID position in the output by using the `--master-data` or `--dump-slave` options. If you are not using `mysqldump` version

10.0.13 or greater, you can run the `SHOW MASTER STATUS` or use those same `mysqldump` options to get the binary log file name and position, then convert them to a GTID by running `BINLOG_GTID_POS` on the external MariaDB instance:

```
SELECT BINLOG_GTID_POS('<binary log file name>', <binary log file position>);
```

For more information about the MariaDB implementation of GTIDs, go to Global transaction ID in the MariaDB documentation.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The `MASTER_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

Usage notes

The `mysql.rds_set_external_master_gtid` procedure must be run by the master user. It must be run on the MariaDB DB instance that you are configuring as the replica of a MariaDB instance running external to Amazon RDS. Before running `mysql.rds_set_external_master_gtid`, you must have configured the instance of MariaDB running external to Amazon RDS as a source instance. For more information, see Importing data into an Amazon RDS for MariaDB DB instance.

> **Warning**
>
> Do not use `mysql.rds_set_external_master_gtid` to manage replication between two Amazon RDS DB instances. Use it only when replicating with a MariaDB instance running external to RDS. For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas.

After calling `mysql.rds_set_external_master_gtid` to configure an Amazon RDS DB instance as a read replica, you can call mysql.rds_start_replication on the replica to start the

replication process. You can call mysql.rds_reset_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) to remove the read replica configuration.

When `mysql.rds_set_external_master_gtid` is called, Amazon RDS records the time, user, and an action of "set master" in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

Examples

When run on a MariaDB DB instance, the following example configures it as the replica of an instance of MariaDB running external to Amazon RDS.

```
call mysql.rds_set_external_master_gtid
 ('Sourcedb.some.com',3306,'ReplicationUser','SomePassW0rd','0-123-456',0);
```

mysql.rds_kill_query_id

Ends a query running against the MariaDB server in order to terminate long-running or problematic queries. You can identify the query ID and effectively stop a specific query to address performance issues and maintain optimal database operation.

Syntax

```
CALL mysql.rds_kill_query_id(queryID);
```

Parameters

queryID

Integer. The identity of the query to be ended.

Usage notes

To stop a query running against the MariaDB server, use the `mysql.rds_kill_query_id` procedure and pass in the ID of that query. To obtain the query ID, query the MariaDB Information schema PROCESSLIST table, as shown following:

```
SELECT USER, HOST, COMMAND, TIME, STATE, INFO, QUERY_ID FROM
                INFORMATION_SCHEMA.PROCESSLIST WHERE USER = '<user name>';
```

The connection to the MariaDB server is retained.

Examples

The following example ends a query with a query ID of 230040:

```
call mysql.rds_kill_query_id(230040);
```

mysql.rds_execute_operation

Executes InnoDB operations to manage buffer pool states and temporary tablespace. This procedure can be used to dynamically control InnoDB operations such as dumping and loading buffer pool states or truncating temporary tablespace.

Syntax

```
CALL mysql.rds_execute_operation(operation);
```

Parameters

operation

String. The InnoDB operations to execute. Valid values are:

- *innodb_buffer_pool_dump_now* - Operation that dumps the current state of the buffer pool.
- *innodb_buffer_pool_load_now* - Operation that loads the saved buffer pool state.
- *innodb_buffer_pool_load_abort* - Operation that aborts a buffer pool load operation.
- *innodb_truncate_temporary_tablespace_now* - Operation that truncates the temporary tablespace.

Usage notes

This procedure is only supported for MariaDB DB instances running MariaDB version 11.8 and higher.

During execution, binary logging is temporarily disabled to prevent replication of these administrative commands.

The procedure maintains an audit trail by logging all operations in the `mysql.rds_history` table.

Examples

The following example demonstrates temporary tablespace shrinking using `mysql.rds_execute_operation`:

To check current temporary tablespace size, run the following query:

```
SELECT FILE_SIZE FROM information_schema.innodb_sys_tablespaces WHERE name LIKE
 'innodb_temporary';
+------------+
| FILE_SIZE  |
+------------+
| 6723469312 |  -- 6.3 GB
+------------+
```

When you drop temporary tables, it doesn't reduce storage usage in the global tablespace. To reduce the size of the global tablespace, run the `mysql.rds_execute_operation` command to shrink the temporary tablespace.

```
CALL mysql.rds_execute_operation('innodb_truncate_temporary_tablespace_now');
Query OK, 2 rows affected (0.004 sec)
```

After you run the procedure, verify that the space was reclaimed.

```
SELECT FILE_SIZE FROM information_schema.innodb_sys_tablespaces WHERE name LIKE
 'innodb_temporary';
+-----------+
| FILE_SIZE |
+-----------+
|  12582912 |  -- 12 MB
+-----------+
```

> **Note**
>
> The shrink operation might take time, depending on the temporary tablespace size and current workload.

Important

The temporary tablespace shrinks only when all temporary tables that contributed to its size are no longer in use. We recommend that you run this procedure when there are no active temporary tablespaces on the instance.

Local time zone for MariaDB DB instances

By default, the time zone for a MariaDB DB instance is Universal Time Coordinated (UTC). You can set the time zone for your DB instance to the local time zone for your application instead.

To set the local time zone for a DB instance, set the `time_zone` parameter in the parameter group for your DB instance to one of the supported values listed later in this section. When you set the `time_zone` parameter for a parameter group, all DB instances and read replicas that are using that parameter group change to use the new local time zone. For information on setting parameters in a parameter group, see Parameter groups for Amazon RDS.

After you set the local time zone, all new connections to the database reflect the change. If you have any open connections to your database when you change the local time zone, you won't see the local time zone update until after you close the connection and open a new connection.

You can set a different local time zone for a DB instance and one or more of its read replicas. To do this, use a different parameter group for the DB instance and the replica or replicas and set the `time_zone` parameter in each parameter group to a different local time zone.

If you are replicating across AWS Regions, then the source DB instance and the read replica use different parameter groups (parameter groups are unique to an AWS Region). To use the same local time zone for each instance, you must set the `time_zone` parameter in the instance's and read replica's parameter groups.

When you restore a DB instance from a DB snapshot, the local time zone is set to UTC. You can update the time zone to your local time zone after the restore is complete. If you restore a DB instance to a point in time, then the local time zone for the restored DB instance is the time zone setting from the parameter group of the restored DB instance.

The Internet Assigned Numbers Authority (IANA) publishes new time zones at https://www.iana.org/time-zones several times a year. Every time RDS releases a new minor maintenance release of MariaDB, it ships with the latest time zone data at the time of the release. When you use the latest RDS for MariaDB versions, you have recent time zone data from RDS. To ensure that your DB instance has recent time zone data, we recommend upgrading to a higher DB engine version. Alternatively, you can modify the time zone tables in MariaDB DB instances manually. To do so, you can use SQL commands or run the mysql_tzinfo_to_sql tool in a SQL client. After updating the time zone data manually, reboot your DB instance so that the changes take effect. RDS doesn't modify or reset the time zone data of running DB instances. New time zone data is installed only when you perform a database engine version upgrade.

You can set your local time zone to one of the following values.

Zone	Time zone
Africa	Africa/Cairo, Africa/Casablanca, Africa/Harare, Africa/Monrovia, Africa/Nairobi, Africa/Tripoli, Africa/Windhoek
America	America/Araguaina, America/Asuncion, America/Bogota, America/Buenos_Aires, America/Caracas, America/Chihuahua, America/Cuiaba, America/Denver, America/Fortaleza, America/Guatemala, America/Halifax, America/Manaus, America/Matamoros, America/Monterrey, America/Montevideo, America/Phoenix, America/Santiago, America/Tijuana
Asia	Asia/Amman, Asia/Ashgabat, Asia/Baghdad, Asia/Baku, Asia/Bangkok, Asia/Beirut, Asia/Calcutta, Asia/Damascus, Asia/Dhaka, Asia/Irkutsk, Asia/Jerusalem, Asia/Kabul, Asia/Karachi, Asia/Kathmandu, Asia/Krasnoyarsk, Asia/Magadan, Asia/Muscat, Asia/Novosibirsk, Asia/Riyadh, Asia/Seoul, Asia/Shanghai, Asia/Singapore, Asia/Taipei, Asia/Tehran, Asia/Tokyo, Asia/Ulaanbaatar, Asia/Vladivostok, Asia/Yakutsk, Asia/Yerevan
Atlantic	Atlantic/Azores
Australia	Australia/Adelaide, Australia/Brisbane, Australia/Darwin, Australia/Hobart, Australia/Perth, Australia/Sydney
Brazil	Brazil/DeNoronha, Brazil/East
Canada	Canada/Newfoundland, Canada/Saskatchewan, Canada/Yukon
Europe	Europe/Amsterdam, Europe/Athens, Europe/Dublin, Europe/Helsinki, Europe/Istanbul, Europe/Kaliningrad Europe/Moscow, Europe/Paris, Europe/Prague, Europe/Sarajevo
Pacific	Pacific/Auckland, Pacific/Fiji, Pacific/Guam, Pacific/Honolulu, Pacific/Samoa
US	US/Alaska, US/Central, US/East-Indiana, US/Eastern, US/Pacific
UTC	UTC

Known issues and limitations for RDS for MariaDB

The following items are known issues and limitations when using RDS for MariaDB.

> **Note**
>
> This list is not exhaustive.

Topics

- MariaDB file size limits in Amazon RDS
- InnoDB reserved word
- Custom ports
- Performance Insights

MariaDB file size limits in Amazon RDS

For MariaDB DB instances, the maximum size of a table is 16 TB when using InnoDB file-per-table tablespaces. This limit also constrains the system tablespace to a maximum size of 16 TB. InnoDB file-per-table tablespaces (with tables each in their own tablespace) are set by default for MariaDB DB instances. This limit isn't related to the maximum storage limit for MariaDB DB instances. For more information about the storage limit, see Amazon RDS DB instance storage.

There are advantages and disadvantages to using InnoDB file-per-table tablespaces, depending on your application. To determine the best approach for your application, see File-per-table tablespaces in the MySQL documentation.

We don't recommend allowing tables to grow to the maximum file size. In general, a better practice is to partition data into smaller tables, which can improve performance and recovery times.

One option that you can use for breaking up a large table into smaller tables is partitioning. *Partitioning* distributes portions of your large table into separate files based on rules that you specify. For example, if you store transactions by date, you can create partitioning rules that distribute older transactions into separate files using partitioning. Then periodically, you can archive the historical transaction data that doesn't need to be readily available to your application. For more information, see Partitioning in the MySQL documentation.

To determine the size of all InnoDB tablespaces

- Use the following SQL command to determine if any of your tables are too large and are candidates for partitioning.

> **Note**
>
> For MariaDB 10.6 and higher, this query also returns the size of the InnoDB system tablespace.
> For MariaDB versions earlier than 10.6, you can't determine the size of the InnoDB system tablespace by querying the system tables. We recommend that you upgrade to a later version.

```
SELECT SPACE,NAME,ROUND((ALLOCATED_SIZE/1024/1024/1024), 2)
as "Tablespace Size (GB)"
FROM information_schema.INNODB_SYS_TABLESPACES ORDER BY 3 DESC;
```

To determine the size of non-InnoDB user tables

- Use the following SQL command to determine if any of your non-InnoDB user tables are too large.

```
SELECT TABLE_SCHEMA, TABLE_NAME, round(((DATA_LENGTH + INDEX_LENGTH+DATA_FREE)
/ 1024 / 1024/ 1024), 2) As "Approximate size (GB)" FROM information_schema.TABLES
WHERE TABLE_SCHEMA NOT IN ('mysql', 'information_schema', 'performance_schema')
and ENGINE<>'InnoDB';
```

To enable InnoDB file-per-table tablespaces

- Set the `innodb_file_per_table` parameter to 1 in the parameter group for the DB instance.

To disable InnoDB file-per-table tablespaces

- Set the `innodb_file_per_table` parameter to `0` in the parameter group for the DB instance.

For information on updating a parameter group, see Parameter groups for Amazon RDS.

When you have enabled or disabled InnoDB file-per-table tablespaces, you can issue an ALTER TABLE command. You can use this command to move a table from the global tablespace to its own tablespace. Or you can move a table from its own tablespace to the global tablespace. Following is an example.

```
ALTER TABLE table_name ENGINE=InnoDB, ALGORITHM=COPY;
```

InnoDB reserved word

InnoDB is a reserved word for RDS for MariaDB. You can't use this name for a MariaDB database.

Custom ports

Amazon RDS blocks connections to custom port 33060 for the MariaDB engine. Choose a different port for your MariaDB engine.

Performance Insights

InnoDB counters are not visible in Performance Insights for RDS for MariaDB version 10.11 because the MariaDB community no longer supports them.

Amazon RDS for Microsoft SQL Server

Amazon RDS supports several versions and editions of Microsoft SQL Server. The following table shows the most recent supported minor version of each major version. For the full list of supported versions, editions, and RDS engine versions, see Microsoft SQL Server versions on Amazon RDS.

Major version	Service Pack / GDR	Cumulative Update	Minor version	Knowledge Base Article	Release Date
SQL Server 2022	Not applicable	CU23	16.0.4236.2	KB5078297	January 29, 2026
SQL Server 2019	GDR	CU32 GDR	15.0.4455.2	KB5068404	November 11, 2025
SQL Server 2017	GDR	CU31 GDR	14.0.3515.1	KB5068402	November 11, 2025
SQL Server 2016	SP3 GDR	Not applicable	13.0.6475.1	KB5068401	November 11, 2025

For information about licensing for SQL Server, see Licensing Microsoft SQL Server on Amazon RDS. For information about SQL Server builds, see this Microsoft support article about Where to find information about the latest SQL Server builds.

With Amazon RDS, you can create DB instances and DB snapshots, point-in-time restores, and automated or manual backups. DB instances running SQL Server can be used inside a VPC. You can also use Secure Sockets Layer (SSL) to connect to a DB instance running SQL Server, and you can use transparent data encryption (TDE) to encrypt data at rest. Amazon RDS currently supports Multi-AZ deployments for SQL Server using SQL Server Database Mirroring (DBM) or Always On Availability Groups (AGs) as a high-availability, failover solution.

To deliver a managed service experience, Amazon RDS does not provide shell access to DB instances, and it restricts access to certain system procedures and tables that require advanced privileges. Amazon RDS supports access to databases on a DB instance using any standard SQL client application such as Microsoft SQL Server Management Studio. Amazon RDS does not allow

direct host access to a DB instance via Telnet, Secure Shell (SSH), or Windows Remote Desktop Connection. When you create a DB instance, the master user is assigned to the *db_owner* role for all user databases on that instance, and has all database-level permissions except for those that are used for backups. Amazon RDS manages backups for you.

Before creating your first DB instance, you should complete the steps in the setting up section of this guide. For more information, see Setting up your Amazon RDS environment.

Topics

- Common management tasks for Microsoft SQL Server on Amazon RDS
- Limitations for Microsoft SQL Server DB instances
- DB instance class support for Microsoft SQL Server
- Optimize CPUs for RDS for SQL Server license-included instances
- Microsoft SQL Server security
- Compliance program support for Microsoft SQL Server DB instances
- Microsoft SQL Server versions on Amazon RDS
- Microsoft SQL Server features on Amazon RDS
- Multi-AZ deployments using Microsoft SQL Server Database Mirroring or Always On availability groups
- Using Transparent Data Encryption to encrypt data at rest
- Functions and stored procedures for Amazon RDS for Microsoft SQL Server
- Local time zone for Microsoft SQL Server DB instances
- Licensing Microsoft SQL Server on Amazon RDS
- Connecting to your Microsoft SQL Server DB instance
- Working with SQL Server Developer Edition on RDS for SQL Server
- Working with Active Directory with RDS for SQL Server
- Upgrades of the Microsoft SQL Server DB engine
- Working with storage in RDS for SQL Server
- Importing and exporting SQL Server databases using native backup and restore
- Working with read replicas for Microsoft SQL Server in Amazon RDS
- Multi-AZ deployments for Amazon RDS for Microsoft SQL Server
- Additional features for Microsoft SQL Server on Amazon RDS

- Options for the Microsoft SQL Server database engine
- Common DBA tasks for Amazon RDS for Microsoft SQL Server

Common management tasks for Microsoft SQL Server on Amazon RDS

The following are the common management tasks you perform with an Amazon RDS for SQL Server DB instance, with links to relevant documentation for each task.

Task area	Description	Relevant documentation
Instance classes, storage, and PIOPS	If you are creating a DB instance for production purposes, you should understand how instance classes, storage types, and Provisioned IOPS work in Amazon RDS.	DB instance class support for Microsoft SQL Server Amazon RDS storage types
Multi-AZ deployments	A production DB instance should use Multi-AZ deployments. Multi-AZ deployments provide increased availability, data durability, and fault tolerance for DB instances. Multi-AZ deployments for SQL Server are implemented using SQL Server's native DBM or AGs technology.	Configuring and managing a Multi-AZ deployment for Amazon RDS Multi-AZ deployments using Microsoft SQL Server Database Mirroring or Always On availability groups
Amazon Virtual Private Cloud (VPC)	If your AWS account has a default VPC, then your DB instance is automatically created inside the default VPC. If your account does	Working with a DB instance in a VPC

Task area	Description	Relevant documentation
	not have a default VPC, and you want the DB instance in a VPC, you must create the VPC and subnet groups before you create the DB instance.	
Security groups	By default, DB instances are created with a firewall that prevents access to them. You therefore must create a security group with the correct IP addresses and network configuration to access the DB instance.	Controlling access with security groups
Parameter groups	If your DB instance is going to require specific database parameters, you should create a parameter group before you create the DB instance.	Parameter groups for Amazon RDS
Option groups	If your DB instance is going to require specific database options, you should create an option group before you create the DB instance.	Options for the Microsoft SQL Server database engine
Connecting to your DB instance	After creating a security group and associating it to a DB instance, you can connect to the DB instance using any standard SQL client application such as Microsoft SQL Server Management Studio.	Connecting to your Microsoft SQL Server DB instance

Task area	Description	Relevant documentation
Backup and restore	When you create your DB instance, you can configure it to take automated backups. You can also back up and restore your databases manually by using full backup files (.bak files).	Introduction to backups Importing and exporting SQL Server databases using native backup and restore
Monitoring	You can monitor your SQL Server DB instance by using CloudWatch Amazon RDS metrics, events, and enhanced monitoring.	Viewing metrics in the Amazon RDS console Viewing Amazon RDS events
Log files	You can access the log files for your SQL Server DB instance.	Monitoring Amazon RDS log files Amazon RDS for Microsoft SQL Server database log files

There are also advanced administrative tasks for working with SQL Server DB instances. For more information, see the following documentation:

- Common DBA tasks for Amazon RDS for Microsoft SQL Server.
- Working with AWS Managed Active Directory with RDS for SQL Server
- Accessing the tempdb database

Limitations for Microsoft SQL Server DB instances

The Amazon RDS implementation of Microsoft SQL Server on a DB instance has some limitations that you should be aware of:

- The maximum number of databases supported on a DB instance depends on the instance class type and the availability mode—Single-AZ, Multi-AZ Database Mirroring (DBM), or Multi-AZ

Availability Groups (AGs). The Microsoft SQL Server system databases don't count toward this limit.

The following table shows the maximum number of supported databases for each instance class type and availability mode. Use this table to help you decide if you can move from one instance class type to another, or from one availability mode to another. If your source DB instance has more databases than the target instance class type or availability mode can support, modifying the DB instance fails. You can see the status of your request in the **Events** pane.

Instance class type	Allowed vCPU range	Single-AZ	Multi-AZ with DBM	Multi-AZ with Always On AGs
db.*.micro to db.*.medium	N/A	30	N/A	N/A
db.*.large	N/A	30	30	30
db.*.xlarge to db.*.16xlarge	4 vCPUs - 64 vCPUs	100	50	75
db.*.24xlarge to db.*.32xlarge	4 vCPUs - 64 vCPUs	100	50	100
db.*.24xlarge to db.*.32xlarge	96 vCPUs - 128 vCPUs	100	50	100

* Represents the different instance class types.

For example, let's say that your DB instance runs on a db.*.16xlarge with Single-AZ and that it has 76 databases. You modify the DB instance to upgrade to using Multi-AZ Always On AGs. This upgrade fails, because your DB instance contains more databases than your target configuration can support. If you upgrade your instance class type to db.*.24xlarge instead, the modification succeeds.

If the upgrade fails, you see events and messages similar to the following:

- Unable to modify database instance class. The instance has 76 databases, but after conversion it would only support 75.

- Unable to convert the DB instance to Multi-AZ: The instance has 76 databases, but after conversion it would only support 75.

  If the point-in-time restore or snapshot restore fails, you see events and messages similar to the following:

  - Database instance put into incompatible-restore. The instance has 76 databases, but after conversion it would only support 75.
- The following ports are reserved for Amazon RDS, and you can't use them when you create a DB instance: `1234, 1434, 3260, 3343, 3389, 47001,` and `49152-49156`.
- Client connections from IP addresses within the range 169.254.0.0/16 are not permitted. This is the Automatic Private IP Addressing Range (APIPA), which is used for local-link addressing.
- SQL Server Standard Edition uses only a subset of the available processors if the DB instance has more processors than the software limits (24 cores, 4 sockets, and 128GB RAM). Examples of this are the db.m5.24xlarge and db.r5.24xlarge instance classes.

  For more information, see the table of scale limits under Editions and supported features of SQL Server 2019 (15.x) in the Microsoft documentation.
- Amazon RDS for SQL Server doesn't support importing data into the msdb database.
- You can't rename databases on a DB instance in a SQL Server Multi-AZ deployment.
- Make sure that you use these guidelines when setting the following DB parameters on RDS for SQL Server:
  - `max server memory (mb)` >= 256 MB
  - `max worker threads` >= (number of logical CPUs * 7)

  For more information on setting DB parameters, see Parameter groups for Amazon RDS.
- The maximum storage size for SQL Server DB instances is the following:
  - General Purpose (SSD) storage – 16 TiB for all editions
  - Provisioned IOPS storage – 64 TiB for all editions
  - Magnetic storage – 1 TiB for all editions

  If you have a scenario that requires a larger amount of storage, you can use sharding across multiple DB instances to get around the limit. This approach requires data-dependent routing logic in applications that connect to the sharded system. You can use an existing sharding framework, or you can write custom code to enable sharding. If you use an existing framework, the framework can't install any components on the same server as the DB instance.

- The minimum storage size for SQL Server DB instances is the following:
  - General Purpose (SSD) storage – 20 GiB for Enterprise, Standard, Web, and Express Editions
  - Provisioned IOPS storage – 20 GiB for Enterprise, Standard, Web, and Express Editions
  - Magnetic storage – 20 GiB for Enterprise, Standard, Web, and Express Editions
- Amazon RDS doesn't support running these services on the same server as your RDS DB instance:
  - Data Quality Services
  - Master Data Services

  To use these features, we recommend that you install SQL Server on an Amazon EC2 instance, or use an on-premises SQL Server instance. In these cases, the EC2 or SQL Server instance acts as the Master Data Services server for your SQL Server DB instance on Amazon RDS. You can install SQL Server on an Amazon EC2 instance with Amazon EBS storage, pursuant to Microsoft licensing policies.
- Because of limitations in Microsoft SQL Server, restoring to a point in time before successfully running `DROP DATABASE` might not reflect the state of that database at that point in time. For example, the dropped database is typically restored to its state up to 5 minutes before the `DROP DATABASE` command was issued. This type of restore means that you can't restore the transactions made during those few minutes on your dropped database. To work around this, you can reissue the `DROP DATABASE` command after the restore operation is completed. Dropping a database removes the transaction logs for that database.
- For SQL Server, you create your databases after you create your DB instance. Database names follow the usual SQL Server naming rules with the following differences:
  - Database names can't start with `rdsadmin`.
  - They can't start or end with a space or a tab.
  - They can't contain any of the characters that create a new line.
  - They can't contain a single quote (`'`).
- SQL Server Web Edition only allows you to use the **Dev/Test** template when creating a new RDS for SQL Server DB instance.
- SQL Server Web Edition is designed for web hosters and web VAPs to host public and internet-accessible web pages, websites, web applications, and web services. For more information, see Licensing Microsoft SQL Server on Amazon RDS.

DB instance class support for Microsoft SQL Server

The computation and memory capacity of a DB instance is determined by its DB instance class. The DB instance class you need depends on your processing power and memory requirements. For more information, see DB instance classes.

The following list of DB instance classes supported for Microsoft SQL Server is provided here for your convenience. For the most current list, see the RDS console: https://console.aws.amazon.com/rds/.

Not all DB instance classes are available on all supported SQL Server minor versions. For example, some newer DB instance classes such as db.r6i aren't available on older minor versions. You can use the describe-orderable-db-instance-options AWS CLI command to find out which DB instance classes are available for your SQL Server edition and version.

SQL Server edition	2022 support range	2019 support range	2017 and 2016 support range
Enterprise Edition	db.t3.xlarge –db.t3.2xlarge	db.t3.xlarge –db.t3.2xlarge	db.t3.xlarge –db.t3.2xlarge
	db.r5.xlarge –db.r5.24xlarge	db.r5.xlarge –db.r5.24xlarge	db.r5.xlarge –db.r5.24xlarge
	db.r5b.xlarge –db.r5b.24xlarge	db.r5b.xlarge –db.r5b.24xlarge	db.r5b.xlarge –db.r5b.24xlarge
	db.r5d.xlarge –db.r5d.24xlarge	db.r5d.xlarge –db.r5d.24xlarge	db.r5d.xlarge –db.r5d.24xlarge
	db.r6i.xlarge –db.r6i.32xlarge db.r7i.xlarge –db.r7i.48xlarge	db.r6i.xlarge –db.r6i.32xlarge	db.r6i.xlarge –db.r6i.32xlarge

SQL Server edition	2022 support range	2019 support range	2017 and 2016 support range
	db.m5.xlarge –db.m5.24xlarge db.m5d.xlarge –db.m5d.24xlarge db.m6i.xlarge –db.m6i.32xlarge db.m7i.xlarge –db.m7i.48xlarge db.x2iedn.xlarge –db.x2iedn.32xlarge db.z1d.xlarge –db.z1d.12xlarge	db.r7i.xlarge –db.r7i.48xlarge db.m5.xlarge –db.m5.24xlarge db.m5d.xlarge –db.m5d.24xlarge db.m6i.xlarge –db.m6i.32xlarge db.m7i.xlarge –db.m7i.48xlarge db.x2iedn.xlarge –db.x2iedn.32xlarge db.z1d.xlarge –db.z1d.12xlarge	db.r7i.xlarge –db.r7i.48xlarge db.m5.xlarge –db.m5.24xlarge db.m5d.xlarge –db.m5d.24xlarge db.m6i.xlarge –db.m6i.32xlarge db.m7i.xlarge –db.m7i.48xlarge db.x2iedn.xlarge –db.x2iedn.32xlarge db.z1d.xlarge –db.z1d.12xlarge

SQL Server edition	2022 support range	2019 support range	2017 and 2016 support range
Standard Edition	db.t3.xlarge –db.t3.2xlarge db.r5.large –db.r5.24xlarge db.r5b.large –db.r5b.8xlarge db.r5d.large –db.r5d.24xlarge db.r6i.large –db.r6i.8xlarge db.r7i.large –db.r7i.12xlarge db.m5.large –db.m5.24xlarge db.m5d.large –db.m5d.24xlarge db.m6i.large –db.m6i.8xlarge db.m7i.large –db.m7i.12xlarge	db.t3.xlarge –db.t3.2xlarge db.r5.large –db.r5.24xlarge db.r5b.large –db.r5b.24xlarge db.r5d.large –db.r5d.24xlarge db.r6i.large –db.r6i.8xlarge db.r7i.large –db.r7i.12xlarge db.m5.large –db.m5.24xlarge db.m5d.large –db.m5d.24xlarge db.m6i.large –db.m6i.8xlarge db.m7i.large –db.m7i.12xlarge	db.t3.xlarge –db.t3.2xlarge db.r5.large –db.r5.24xlarge db.r5b.large –db.r5b.24xlarge db.r5d.large –db.r5d.24xlarge db.r6i.large –db.r6i.8xlarge db.r7i.large –db.r7i.12xlarge db.m5.large –db.m5.24xlarge db.m5d.large –db.m5d.24xlarge db.m6i.large –db.m6i.8xlarge db.m7i.large –db.m7i.12xlarge

SQL Server edition	2022 support range	2019 support range	2017 and 2016 support range
	db.x2iedn .xlarge –db.x2iedn .8xlarge db.z1d.la rge –db.z1d.12 xlarge	db.x2iedn .xlarge –db.x2iedn .32xlarge db.z1d.la rge –db.z1d.12 xlarge	db.x2iedn .xlarge –db.x2iedn .32xlarge db.z1d.la rge –db.z1d.12 xlarge

SQL Server edition	2022 support range	2019 support range	2017 and 2016 support range
Web Edition	db.t3.small –db.t3.xlarge db.r5.large –db.r5.4xlarge db.r5b.large –db.r5b.4xlarge db.r5d.large –db.r5d.4xlarge db.r6i.large –db.r6i.4xlarge db.r7i.large –db.r7i.4xlarge db.m5.large –db.m5.4xlarge db.m5d.large –db.m5d.4xlarge db.m6i.large –db.m6i.4xlarge db.m7i.large –db.m7i.4xlarge db.z1d.large –db.z1d.13xlarge	db.t3.small –db.t3.2xlarge db.r5.large –db.r5.4xlarge db.r5b.large –db.r5b.4xlarge db.r5d.large –db.r5d.4xlarge db.r6i.large –db.r6i.4xlarge db.r7i.large –db.r7i.4xlarge db.m5.large –db.m5.4xlarge db.m5d.large –db.m5d.4xlarge db.m6i.large –db.m6i.4xlarge db.m7i.large –db.m7i.4xlarge db.z1d.large –db.z1d.3xlarge	db.t3.small –db.t3.2xlarge db.r5.large –db.r5.4xlarge db.r5b.large –db.r5b.4xlarge db.r5d.large –db.r5d.4xlarge db.r6i.large –db.r6i.4xlarge db.r7i.large –db.r7i.4xlarge db.m5.large –db.m5.4xlarge db.m5d.large –db.m5d.4xlarge db.m6i.large –db.m6i.4xlarge db.m7i.large –db.m7i.4xlarge db.z1d.large –db.z1d.3xlarge
Express Edition	db.t3.micro –db.t3.xlarge	db.t3.micro –db.t3.xlarge	db.t3.micro –db.t3.xlarge

SQL Server edition	2022 support range	2019 support range	2017 and 2016 support range
Develop Edition	db.m6i.xlarge –db.m6i.32xlarge db.r6i.xlarge –db.r6i.32xlarge		

Note

- Starting with the 7th generation instance class, hyper-threading is disabled on RDS SQL Server for instance sizes 2xlarge and above. This results in the total number of vCPUs available being half of that supported by the corresponding EC2 instance. For example, the EC2 instance type `m7i.2xlarge` by default supports 4 cores and 2 threadsPerCore, resulting in a total of 8 vCPUs. In contrast, the RDS for SQL Server `db.m7i.2xlarge` instance, with hyper-threading disabled, results in 4 cores and 1 threadsPerCore, overall 4 vCPUs.
- Starting with the 7th generation instances, your billing provides a detailed breakdown of RDS DB instance and third-party licensing fees. For more details, refer to RDS SQL Server pricing.

Optimize CPUs for RDS for SQL Server license-included instances

With RDS for SQL Server, you can use Optimize CPU by specifying processor features to configure the vCPU count on your DB instance while maintaining the same memory and IOPS. You can achieve desired memory-to-CPU ratios for specific database workload requirements and reduce licensing costs for Microsoft Windows OS and SQL Server, which are based on vCPU count.

To specify processor feature, use the following parameters:

```
--processor-features "Name=coreCount,Value=value" \
 "Name=threadsPerCore,Value=value"
```

- **coreCount** – Specify the number of CPU cores for the DB instance, to optimize licensing costs for DB instances. See DB instance classes that support Optimize CPU to find the allowed values for core count for a selected instance type.
- **threadsPerCore** – Specify the threads per core to define the number of threads per CPU core. See DB instance classes that support Optimize CPU to find the allowed values for threads per core for a selected instance type.

Sample command to create an RDS for SQL Server instance with Optimize CPU settings:

```
aws rds create-db-instance \
    --engine sqlserver-ee \
    --engine-version 16.00 \
    --license-model license-included \
    --allocated-storage 300 \
    --master-username myuser \
    --master-user-password xxxxx \
    --no-multi-az \
    --vpc-security-group-ids myvpcsecuritygroup \
    --db-subnet-group-name mydbsubnetgroup \
    --db-instance-identifier my-rds-instance \
    --db-instance-class db.m7i.8xlarge \
    --processor-features "Name=coreCount,Value=8" "Name=threadsPerCore,Value=1"
```

In this example, you create a `db.m7i.8xlarge` instance, which by default has a coreCount of 16. By using Optimize CPU, you opt for a coreCount of 8, resulting in an effective vCPU count of 8.

If you create the instance without the `--processor-features` parameter, core count is set to 16 and threads per core is set to 1 by default, resulting in a default vCPU count of 16.

Some considerations to keep in mind while specifying processor features:

- **Create** – Specify both the `coreCount` and `threadsPerCore` for the `processor-features` parameter from the allowed values. See DB instance classes that support Optimize CPU.
- **Modify** – When modifying from one instance class configured with Optimize CPU settings to another one that supports Optimize CPU settings, you must specify the default processor

settings using the `--use-default-processor-features` parameter or explicitly define the options during the modify request.

> **Note**
>
> Changing the vCPU count can have implications for the licensing fee cost associated with the DB instance.

- **Snapshot restore** – When restoring a snapshot to the same instance type as source, the restored DB instance inherits the Optimize CPU settings from the snapshot. If restoring to a different instance type, you need to define the Optimize CPU settings for the target instance or specify the `--use-default-processor-features` parameter.
- **Point-in-time restore** – Point-in-time restore (PITR) involves restoring a specific snapshot based on the designated time for PITR and subsequently applying all transactional log backups to that snapshot, thereby bringing the instance to the specified point in time. For PITR, the Optimize CPU settings, `coreCount` and `threadsPerCore`, are derived from the source snapshot (not the point in time) unless custom values are specified during the PITR request. If the source snapshot being used is enabled with Optimize CPU settings and you are using a different instance type for PITR, you must define the Optimize CPU settings for the target instance or specify the `—-use-default-processor-features` parameter.

Limitations

The following limitations apply when using Optimize CPU:

- Optimize CPU is supported with Enterprise, Standard, and Web Editions only.
- Optimize CPU is available on select instances. See DB instance classes that support Optimize CPU.
- Customizing the number of CPU cores is supported on instance sizes of `2xlarge` and above. With these instance types, the minimum number of vCPCU supported for Optimize CPU is 4.
- Optimize CPU allows only 1 thread per core since Hyper-Threading is disabled for instances starting from 7th generation that support Optimize CPU.

DB instance classes that support Optimize CPU

RDS for SQL Server supports Optimize CPU beginning with the 7th Generation instance class type. Additionally, RDS provides a detailed billing breakdown of RDS DB instance and third-party licensing fees, starting from the 7th Generation instance class type, regardless of whether the Optimize CPU feature is enabled.

RDS for SQL Server provides support for Optimize CPU on specific instance sizes, with the smallest instance size supported being `2xlarge`. The minimum configuration supported is 4 vCPUs. The table below outlines the DB instance classes that support the Optimize CPU, including their default and valid values for CPU cores, CPU threads per core and vCPUs:

General purpose instances

Instan type	Default vCPUs	Default CPU cores	Valid CPU cores	Valid threads per core
m7i.l	2	1	1	2
m7i.x e	4	2	2	2
m7i.2 ge	4	4	1,2,3,4	1
m7i.4 ge	8	8	1,2,3,4,5,6,7,8	1
m7i.8 ge	16	16	1,2,3,4,5,6,7,8,9, 10,11,12,13,14,15, 16	1
m7i.1 rge	24	24	1,2,3,4,5,6,7,8,9, 10,11,12,13,14,15, 16,17,18,19,20,21, 22,23,24	1
m7i.1 rge	32	32	1,2,3,4,5,6,7,8,9, 10,11,12,13,14,15,	1

Instan type	Default vCPUs	Default CPU cores	Valid CPU cores	Valid threads per core
			16,17,18,19,20,21, 22,23,24,25,26,27, 28,29,30,31,32	

Memory optimized instances

Instan type	Default vCPUs	Default CPU cores	Valid CPU cores	Valid threads per core
r7i.l	2	1	1	2
r7i.x e	4	2	2	2
r7i.2 ge	4	4	1,2,3,4	1
r7i.4 ge	8	8	1,2,3,4,5,6,7,8	1
r7i.8 ge	16	16	1,2,3,4,5,6,7,8,9, 10,11,12,13,14,15, 16	1
r7i.1 rge	24	24	1,2,3,4,5,6,7,8,9, 10,11,12,13,14,15, 16,17,18,19,20,21, 22,23,24	1
r7i.1 rge	32	32	1,2,3,4,5,6,7,8,9, 10,11,12,13,14,15, 16,17,18,19,20,21, 22,23,24,25,26,27, 28,29,30,31,32	1

Setting the CPU cores and threads per CPU core for a DB instance class

You can configure the number of CPU cores and threads per core for the DB instance class when you perform the following operations:

- Creating an Amazon RDS DB instance
- Modifying an Amazon RDS DB instance
- Restoring to a DB instance
- Restoring a DB instance to a specified time for Amazon RDS

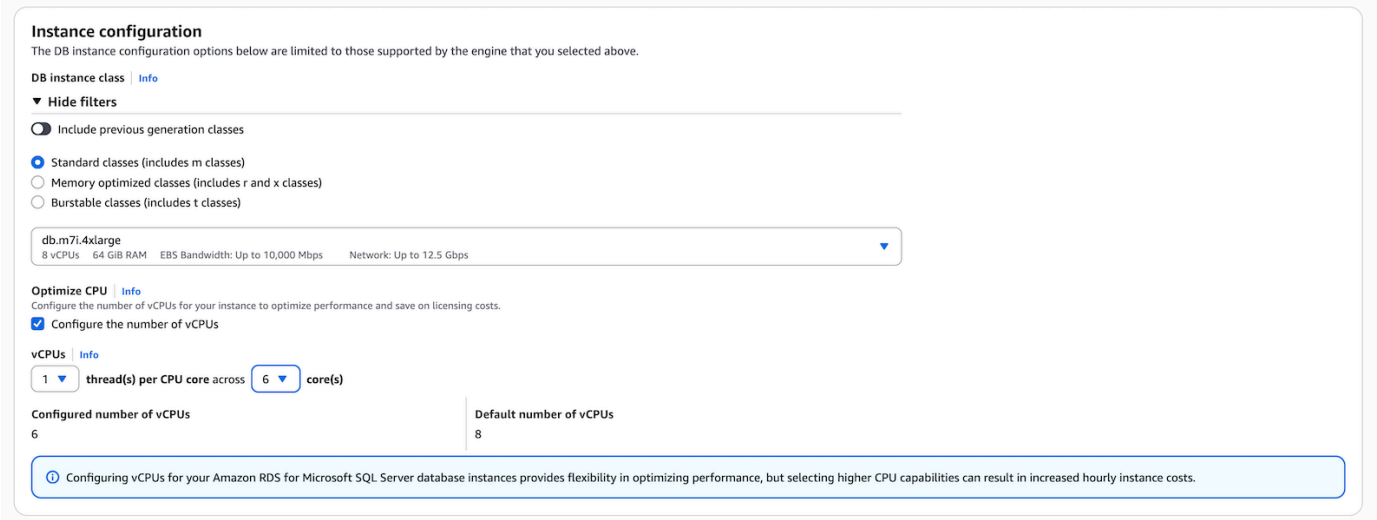
> **Note**
>
> When you modify a DB instance to configure the number of CPU cores or threads per core, there is a brief outage similar to when you modify the instance class.

Set the CPU cores by using the AWS Management Console, AWS CLI or the RDS API.

Console

To set the cores

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Create database**.
3. When setting the **Instance configuration** options:
    a. Choose the **Optimize CPU** option.
    b. Set your **vCPU** option by choosing the number of cores.



4. After completing other selections, select **Create database**.

AWS CLI

To set the cores

1. To configure Optimize CPU using the AWS CLI, include the `--processor-features` option in the command. Specify the number of CPU cores with the `coreCount` and `threadsPerCore` as 1.
2. Use the following syntax:

```
aws rds create-db-instance \
    --engine sqlserver-ee \
    --engine-version 16.00 \
    --license-model license-included \
    --allocated-storage 300 \
    --master-username myuser \
    --master-user-password xxxxx \
    --no-multi-az \
    --vpc-security-group-ids myvpcsecuritygroup \
    --db-subnet-group-name mydbsubnetgroup \
    --db-instance-identifier my-rds-instance \
    --db-instance-class db.m7i.4xlarge \
    --processor-features "Name=coreCount,Value=6" "Name=threadsPerCore,Value=1"
```

Example of viewing valid processor values for a DB instance class

Use the `describe-orderable-db-instance-options` command to show the default vCPUs, cores, and threads per core. For example, the output for the following command shows the processor options for the db.r7i.2xlarge instance class.

```
aws rds describe-orderable-db-instance-options --engine sqlserver-ee \
--db-instance-class db.r7i.2xlarge

Sample output:
-------------------------------------------------------------
|            DescribeOrderableDBInstanceOptions             |
+-----------------------------------------------------------+
||               OrderableDBInstanceOptions                ||
|+-----------------------------------+--------------------+|
||  DBInstanceClass                  |  db.r7i.2xlarge    ||
||  Engine                           |  sqlserver-ee      ||
||  EngineVersion                    |  13.00.6300.2.v1   ||
||  LicenseModel                     |  license-included  ||
||  MaxIopsPerDbInstance             |                    ||
||  MaxIopsPerGib                    |                    ||
||  MaxStorageSize                   |  64000             ||
||  MinIopsPerDbInstance             |                    ||
||  MinIopsPerGib                    |                    ||
||  MinStorageSize                   |  20                ||
||  MultiAZCapable                   |  True              ||
||  OutpostCapable                   |  False             ||
||  ReadReplicaCapable               |  True              ||
||  StorageType                      |  gp2               ||
||  SupportsClusters                 |  False             ||
||  SupportsDedicatedLogVolume       |  False             ||
||  SupportsEnhancedMonitoring       |  True              ||
||  SupportsGlobalDatabases          |  False             ||
||  SupportsIAMDatabaseAuthentication |  False             ||
||  SupportsIops                     |  False             ||
||  SupportsKerberosAuthentication   |  True              ||
||  SupportsPerformanceInsights      |  True              ||
||  SupportsStorageAutoscaling       |  True              ||
||  SupportsStorageEncryption        |  True              ||
||  SupportsStorageThroughput        |  False             ||
||  Vpc                              |  True              ||
|+-----------------------------------+--------------------+|
|||                  AvailabilityZones                    |||
```

```
||+-----------------------------------------------------+||
|||                        Name                         |||
||+-----------------------------------------------------+||
|||  us-west-2a                                         |||
|||  us-west-2b                                         |||
|||  us-west-2c                                         |||
||+-----------------------------------------------------+||
|||             AvailableProcessorFeatures              |||
||+-----------------+-----------------+-----------------+||
|||  AllowedValues  |  DefaultValue   |       Name      |||
||+-----------------+-----------------+-----------------+||
|||  1,2,3,4        |  4              |  coreCount      |||
|||  1              |  1              |  threadsPerCore |||
||+-----------------+-----------------+-----------------+||
```

In addition, you can run the following commands for DB instance class processor information:

- `describe-db-instances` – Shows the processor information for the specified DB instance
- `describe-db-snapshots` – Shows the processor information for the specified DB snapshot
- `describe-valid-db-instance-modifications` – Shows the valid modifications to the processor for the specified DB instance

In the output of the preceding command, the values for the processor features are `null` if Optimize CPU is not configured.

Example of setting the number of CPU cores for a DB instance

The following example modifies *`mydbinstance`* by setting the number of CPU cores to 4 threadsPerCore as 1. Apply the changes immediately by using `--apply-immediately`. If you want to apply the changes during the next scheduled maintenance window, omit `--apply-immediately option`.

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r7i.8xlarge \
    --processor-features "Name=coreCount,Value=4" "Name=threadsPerCore,Value=1" \
    --apply-immediately
```

Example of returning to default processor settings for a DB instance

The following example modifies *mydbinstance* by returning it to the default processor values for it.

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r7i.8xlarge \
    --use-default-processor-features \
    --apply-immediately
```

Microsoft SQL Server security

The Microsoft SQL Server database engine uses role-based security. The master user name that you specify when you create a DB instance is a SQL Server Authentication login that is a member of the `processadmin`, `public`, and `setupadmin` fixed server roles.

Any user who creates a database is assigned to the db_owner role for that database and has all database-level permissions except for those that are used for backups. Amazon RDS manages backups for you.

The following server-level roles aren't available in Amazon RDS for SQL Server:

- bulkadmin
- dbcreator
- diskadmin
- securityadmin
- serveradmin
- sysadmin

The following server-level permissions aren't available on RDS for SQL Server DB instances:

- ALTER ANY DATABASE
- ALTER ANY EVENT NOTIFICATION
- ALTER RESOURCES
- ALTER SETTINGS (you can use the DB parameter group API operations to modify parameters; for more information, see Parameter groups for Amazon RDS)

- AUTHENTICATE SERVER
- CONTROL_SERVER
- CREATE DDL EVENT NOTIFICATION
- CREATE ENDPOINT
- CREATE SERVER ROLE
- CREATE TRACE EVENT NOTIFICATION
- DROP ANY DATABASE
- EXTERNAL ACCESS ASSEMBLY
- SHUTDOWN (You can use the RDS reboot option instead)
- UNSAFE ASSEMBLY
- ALTER ANY AVAILABILITY GROUP
- CREATE ANY AVAILABILITY GROUP

SSL support for Microsoft SQL Server DB instances

You can use SSL to encrypt connections between your applications and your Amazon RDS DB instances running Microsoft SQL Server. You can also force all connections to your DB instance to use SSL. If you force connections to use SSL, it happens transparently to the client, and the client doesn't have to do any work to use SSL.

SSL is supported in all AWS Regions and for all supported SQL Server editions. For more information, see Using SSL with a Microsoft SQL Server DB instance.

Using SSL with a Microsoft SQL Server DB instance

You can use Secure Sockets Layer (SSL) to encrypt connections between your client applications and your Amazon RDS DB instances running Microsoft SQL Server. SSL support is available in all AWS regions for all supported SQL Server editions.

When you create a SQL Server DB instance, Amazon RDS creates an SSL certificate for it. The SSL certificate includes the DB instance endpoint as the Common Name (CN) for the SSL certificate to guard against spoofing attacks.

There are 2 ways to use SSL to connect to your SQL Server DB instance:

- Force SSL for all connections — this happens transparently to the client, and the client doesn't have to do any work to use SSL.

  > **Note**
  >
  > When you set `rds.force_ssl` to 1 and use SSMS version 19.3, 20.0, and 20.2, check for the following:
  >
  > - Enable **Trust Server Certificate** in SSMS.
  > - Import the certificate in your system.

- Encrypt specific connections — this sets up an SSL connection from a specific client computer, and you must do work on the client to encrypt connections.

For information about Transport Layer Security (TLS) support for SQL Server, see TLS 1.2 support for Microsoft SQL Server.

Forcing connections to your DB instance to use SSL

You can force all connections to your DB instance to use SSL. If you force connections to use SSL, it happens transparently to the client, and the client doesn't have to do any work to use SSL.

If you want to force SSL, use the `rds.force_ssl` parameter. By default, the `rds.force_ssl` parameter is set to `0 (off)`. Set the `rds.force_ssl` parameter to `1 (on)` to force connections to use SSL. The `rds.force_ssl` parameter is static, so after you change the value, you must reboot your DB instance for the change to take effect.

To force all connections to your DB instance to use SSL

1. Determine the parameter group that is attached to your DB instance:

   a. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

   b. In the top right corner of the Amazon RDS console, choose the AWS Region of your DB instance.

   c. In the navigation pane, choose **Databases**, and then choose the name of your DB instance to show its details.

   d. Choose the **Configuration** tab. Find the **Parameter group** in the section.

2. If necessary, create a new parameter group. If your DB instance uses the default parameter group, you must create a new parameter group. If your DB instance uses a nondefault parameter group, you can choose to edit the existing parameter group or to create a new parameter group. If you edit an existing parameter group, the change affects all DB instances that use that parameter group.

   To create a new parameter group, follow the instructions in Creating a DB parameter group in Amazon RDS.

3. Edit your new or existing parameter group to set the `rds.force_ssl` parameter to `true`. To edit the parameter group, follow the instructions in Modifying parameters in a DB parameter group in Amazon RDS.

4. If you created a new parameter group, modify your DB instance to attach the new parameter group. Modify the **DB Parameter Group** setting of the DB instance. For more information, see Modifying an Amazon RDS DB instance.

5. Reboot your DB instance. For more information, see Rebooting a DB instance.

Encrypting specific connections

You can force all connections to your DB instance to use SSL, or you can encrypt connections from specific client computers only. To use SSL from a specific client, you must obtain certificates for the client computer, import certificates on the client computer, and then encrypt the connections from the client computer.

> **Note**
>
> All SQL Server instances created after August 5, 2014, use the DB instance endpoint in the Common Name (CN) field of the SSL certificate. Prior to August 5, 2014, SSL certificate verification was not available for VPC-based SQL Server instances. If you have a VPC-based SQL Server DB instance that was created before August 5, 2014, and you want to use SSL certificate verification and ensure that the instance endpoint is included as the CN for the SSL certificate for that DB instance, then rename the instance. When you rename a DB instance, a new certificate is deployed and the instance is rebooted to enable the new certificate.

Obtaining certificates for client computers

To encrypt connections from a client computer to an Amazon RDS DB instance running Microsoft SQL Server, you need a certificate on your client computer.

To obtain that certificate, download the certificate to your client computer. You can download a root certificate that works for all regions. You can also download a certificate bundle that contains both the old and new root certificate. In addition, you can download region-specific intermediate certificates. For more information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

After you have downloaded the appropriate certificate, import the certificate into your Microsoft Windows operating system by following the procedure in the section following.

Importing certificates on client computers

You can use the following procedure to import your certificate into the Microsoft Windows operating system on your client computer.

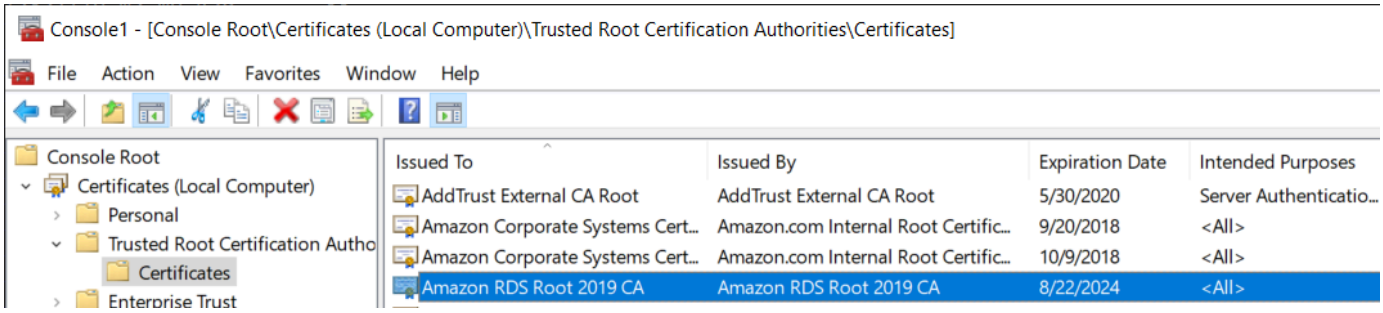
To import the certificate into your Windows operating system:

1. On the **Start** menu, type **Run** in the search box and press **Enter**.
2. In the **Open** box, type **`MMC`** and then choose **OK**.
3. In the MMC console, on the **File** menu, choose **Add/Remove Snap-in**.
4. In the **Add or Remove Snap-ins** dialog box, for **Available snap-ins**, select **`Certificates`**, and then choose **Add**.

5. In the **Certificates snap-in** dialog box, choose **Computer account**, and then choose **Next**.
6. In the **Select computer** dialog box, choose **Finish**.
7. In the **Add or Remove Snap-ins** dialog box, choose **OK**.
8. In the MMC console, expand **Certificates**, open the context (right-click) menu for **Trusted Root Certification Authorities**, choose **All Tasks**, and then choose **Import**.
9. On the first page of the Certificate Import Wizard, choose **Next**.
10. On the second page of the Certificate Import Wizard, choose **Browse**. In the browse window, change the file type to **All files (*.*)** because .pem is not a standard certificate extension. Locate the .pem file that you downloaded previously.

> **Note**
>
> When connecting from Windows clients such as SQL Server Management Studio (SSMS), we recommend using the PKCS#7 (.p7b) certificate format instead of the global-bundle.pem file. The .p7b format ensures the complete certificate chain — including Root and Intermediate Certificate Authorities (CAs) — is correctly imported into the Windows Certificate Store. This prevents connection failures that can occur when mandatory encryption is enabled, as .pem imports may not install the full chain properly.

11. Choose **Open** to select the certificate file, and then choose **Next**.
12. On the third page of the Certificate Import Wizard, choose **Next**.
13. On the fourth page of the Certificate Import Wizard, choose **Finish**. A dialog box appears indicating that the import was successful.
14. In the MMC console, expand **Certificates**, expand **Trusted Root Certification Authorities**, and then choose **Certificates**. Locate the certificate to confirm it exists, as shown here.



Encrypting connections to an Amazon RDS DB instance running Microsoft SQL Server

After you have imported a certificate into your client computer, you can encrypt connections from the client computer to an Amazon RDS DB instance running Microsoft SQL Server.

For SQL Server Management Studio, use the following procedure. For more information about SQL Server Management Studio, see Use SQL Server management studio.

To encrypt connections from SQL Server Management Studio

1. Launch SQL Server Management Studio.
2. For **Connect to server**, type the server information, login user name, and password.
3. Choose **Options**.
4. Select **Encrypt connection**.
5. Choose **Connect**.
6. Confirm that your connection is encrypted by running the following query. Verify that the query returns `true` for `encrypt_option`.

```
select ENCRYPT_OPTION from SYS.DM_EXEC_CONNECTIONS where SESSION_ID = @@SPID
```

For any other SQL client, use the following procedure.

To encrypt connections from other SQL clients

1. Append `encrypt=true` to your connection string. This string might be available as an option, or as a property on the connection page in GUI tools.

> **Note**
>
> To enable SSL encryption for clients that connect using JDBC, you might need to add the Amazon RDS SQL certificate to the Java CA certificate (cacerts) store. You can do this by using the keytool utility.

2. Confirm that your connection is encrypted by running the following query. Verify that the query returns `true` for `encrypt_option`.

```
select ENCRYPT_OPTION from SYS.DM_EXEC_CONNECTIONS where SESSION_ID = @@SPID
```

Configuring SQL Server security protocols and ciphers

You can turn certain security protocols and ciphers on and off using DB parameters. The security parameters that you can configure (except for TLS version 1.2) are shown in the following table.

DB parameter	Allowed values (default in bold)	Description
rds.tls10	**default**, enabled, disabled	TLS 1.0.
rds.tls11	**default**, enabled, disabled	TLS 1.1.
rds.tls12	**default**	TLS 1.2. You can't modify this value.
rds.fips	**0**, 1	When you set the parameter to 1, RDS forces the use of modules that are compliant with the Federal Information Processing Standard (FIPS) 140-2 standard. For more information, see Use SQL Server 2016 in FIPS 140-2-compliant mode in the Microsoft documentation.
rds.rc4	**default**, enabled, disabled	RC4 stream cipher.
rds.diffie-hellman	**default**, enabled, disabled	Diffie-Hellman key-exchange encryption.
rds.diffie-hellman-min-key-bit-length	**default**, 1024, 2048, 3072, 4096	Minimum bit length for Diffie-Hellman keys.
rds.curve25519	**default**, enabled, disabled	Curve25519 elliptic-curve encryption cipher. This parameter isn't supported for all engine versions.

DB parameter	Allowed values (default in bold)	Description
rds.3des168	**default**, enabled, disabled	Triple Data Encryption Standard (DES) encryption cipher with a 168-bit key length.

Note

For minor engine versions after 16.00.4120.1, 15.00.4365.2, 14.00.3465.1, 13.00.6435.1, and 12.00.6449.1, the default setting for the DB parameters `rds.tls10`, `rds.tls11`, `rds.rc4`, `rds.curve25519`, and `rds.3des168` is *disabled*. Otherwise the default setting is *enabled*.

For minor engine versions after 16.00.4120.1, 15.00.4365.2, 14.00.3465.1, 13.00.6435.1, and 12.00.6449.1, the default setting for `rds.diffie-hellman-min-key-bit-lengthis` 3072. Otherwise the default setting is 2048.

Use the following process to configure the security protocols and ciphers:

1. Create a custom DB parameter group.
2. Modify the parameters in the parameter group.
3. Associate the DB parameter group with your DB instance.

For more information on DB parameter groups, see Parameter groups for Amazon RDS.

Creating the security-related parameter group

Create a parameter group for your security-related parameters that corresponds to the SQL Server edition and version of your DB instance.

Console

The following procedure creates a parameter group for SQL Server Standard Edition 2016.

To create the parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
4. In the **Create parameter group** pane, do the following:
   a. For **Parameter group family**, choose **sqlserver-se-13.0**.
   b. For **Group name**, enter an identifier for the parameter group, such as **`sqlserver-ciphers-se-13`**.
   c. For **Description**, enter **`Parameter group for security protocols and ciphers`**.
5. Choose **Create**.

CLI

The following procedure creates a parameter group for SQL Server Standard Edition 2016.

To create the parameter group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name sqlserver-ciphers-se-13 \
    --db-parameter-group-family "sqlserver-se-13.0" \
    --description "Parameter group for security protocols and ciphers"
```

  For Windows:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name sqlserver-ciphers-se-13 ^
    --db-parameter-group-family "sqlserver-se-13.0" ^
    --description "Parameter group for security protocols and ciphers"
```

Modifying security-related parameters

Modify the security-related parameters in the parameter group that corresponds to the SQL Server edition and version of your DB instance.

Console

The following procedure modifies the parameter group that you created for SQL Server Standard Edition 2016. This example turns off TLS version 1.0.

To modify the parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose the parameter group, such as **sqlserver-ciphers-se-13**.
4. Under **Parameters**, filter the parameter list for **rds**.
5. Choose **Edit parameters**.
6. Choose **rds.tls10**.
7. For **Values**, choose **disabled**.
8. Choose **Save changes**.

CLI

The following procedure modifies the parameter group that you created for SQL Server Standard Edition 2016. This example turns off TLS version 1.0.

To modify the parameter group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name sqlserver-ciphers-se-13 \
    --parameters
 "ParameterName='rds.tls10',ParameterValue='disabled',ApplyMethod=pending-reboot"
```

For Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name sqlserver-ciphers-se-13 ^
    --parameters
 "ParameterName='rds.tls10',ParameterValue='disabled',ApplyMethod=pending-reboot"
```

Associating the security-related parameter group with your DB instance

To associate the parameter group with your DB instance, use the AWS Management Console or the AWS CLI.

Console

You can associate the parameter group with a new or existing DB instance:

- For a new DB instance, associate it when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, associate it by modifying the instance. For more information, see Modifying an Amazon RDS DB instance.

CLI

You can associate the parameter group with a new or existing DB instance.

To create a DB instance with the parameter group

- Specify the same DB engine type and major version as you used when creating the parameter group.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.m5.2xlarge \
    --engine sqlserver-se \
    --engine-version 13.00.5426.0.v1 \
```

```
    --allocated-storage 100 \
    --master-user-password secret123 \
    --master-username admin \
    --storage-type gp2 \
    --license-model li \
    --db-parameter-group-name sqlserver-ciphers-se-13
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-se ^
    --engine-version 13.00.5426.0.v1 ^
    --allocated-storage 100 ^
    --master-user-password secret123 ^
    --master-username admin ^
    --storage-type gp2 ^
    --license-model li ^
    --db-parameter-group-name sqlserver-ciphers-se-13
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

To modify a DB instance and associate the parameter group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-parameter-group-name sqlserver-ciphers-se-13 \
    --apply-immediately
```

  For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-parameter-group-name sqlserver-ciphers-se-13 ^
    --apply-immediately
```

Updating applications to connect to Microsoft SQL Server DB instances using new SSL/TLS certificates

As of January 13, 2023, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your RDS DB instances using Secure Socket Layer or Transport Layer Security (SSL/TLS). Following, you can find information about updating your applications to use the new certificates.

This topic can help you to determine whether any client applications use SSL/TLS to connect to your DB instances. If they do, you can further check whether those applications require certificate verification to connect.

> **Note**
>
> Some applications are configured to connect to SQL Server DB instances only if they can successfully verify the certificate on the server.
> For such applications, you must update your client application trust stores to include the new CA certificates.

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB instances. We strongly recommend testing these procedures in a development or staging environment before implementing them in your production environments.

For more information about certificate rotation, see Rotating your SSL/TLS certificate. For more information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . For information about using SSL/TLS with Microsoft SQL Server DB instances, see Using SSL with a Microsoft SQL Server DB instance.

Topics

- Determining whether any applications are connecting to your Microsoft SQL Server DB instance using SSL
- Determining whether a client requires certificate verification in order to connect
- Updating your application trust store

Determining whether any applications are connecting to your Microsoft SQL Server DB instance using SSL

Check the DB instance configuration for the value of the `rds.force_ssl` parameter. By default, the `rds.force_ssl` parameter is set to 0 (off). If the `rds.force_ssl` parameter is set to 1 (on), clients are required to use SSL/TLS for connections. For more information about parameter groups, see Parameter groups for Amazon RDS.

Run the following query to get the current encryption option for all the open connections to a DB instance. The column `ENCRYPT_OPTION` returns `TRUE` if the connection is encrypted.

```
select SESSION_ID,
    ENCRYPT_OPTION,
    NET_TRANSPORT,
    AUTH_SCHEME
    from SYS.DM_EXEC_CONNECTIONS
```

This query shows only the current connections. It doesn't show whether applications that have connected and disconnected in the past have used SSL.

Determining whether a client requires certificate verification in order to connect

You can check whether different types of clients require certificate verification to connect.

> **Note**
>
> If you use connectors other than the ones listed, see the specific connector's documentation for information about how it enforces encrypted connections. For more information, see Connection modules for Microsoft SQL databases in the Microsoft SQL Server documentation.

SQL Server Management Studio

Check whether encryption is enforced for SQL Server Management Studio connections:

1. Launch SQL Server Management Studio.
2. For **Connect to server**, enter the server information, login user name, and password.

3. Choose **Options**.
4. Check if **Encrypt connection** is selected in the connect page.

For more information about SQL Server Management Studio, see Use SQL Server Management Studio.

Sqlcmd

The following example with the `sqlcmd` client shows how to check a script's SQL Server connection to determine whether successful connections require a valid certificate. For more information, see Connecting with sqlcmd in the Microsoft SQL Server documentation.

When using `sqlcmd`, an SSL connection requires verification against the server certificate if you use the `-N` command argument to encrypt connections, as in the following example.

```
$ sqlcmd -N -S dbinstance.rds.amazon.com -d ExampleDB
```

> **Note**
>
> If `sqlcmd` is invoked with the `-C` option, it trusts the server certificate, even if that doesn't match the client-side trust store.

ADO.NET

In the following example, the application connects using SSL, and the server certificate must be verified.

```
using SQLC = Microsoft.Data.SqlClient;

...

    static public void Main()
    {
        using (var connection = new SQLC.SqlConnection(
            "Server=tcp:dbinstance.rds.amazon.com;" +
            "Database=ExampleDB;User ID=LOGIN_NAME;" +
```

```
            "Password=YOUR_PASSWORD;" +
            "Encrypt=True;TrustServerCertificate=False;"
            ))
        {
            connection.Open();
            ...
        }
```

Java

In the following example, the application connects using SSL, and the server certificate must be verified.

```
String connectionUrl =
    "jdbc:sqlserver://dbinstance.rds.amazon.com;" +
    "databaseName=ExampleDB;integratedSecurity=true;" +
    "encrypt=true;trustServerCertificate=false";
```

To enable SSL encryption for clients that connect using JDBC, you might need to add the Amazon RDS certificate to the Java CA certificate store. For instructions, see Configuring the client for encryption in the Microsoft SQL Server documentation. You can also provide the trusted CA certificate file name directly by appending `trustStore=`*`path-to-certificate-trust-store-file`* to the connection string.

> **Note**
>
> If you use `TrustServerCertificate=true` (or its equivalent) in the connection string, the connection process skips the trust chain validation. In this case, the application connects even if the certificate can't be verified. Using `TrustServerCertificate=false` enforces certificate validation and is a best practice.

Updating your application trust store

You can update the trust store for applications that use Microsoft SQL Server. For instructions, see Encrypting specific connections. Also, see Configuring the client for encryption in the Microsoft SQL Server documentation.

If you are using an operating system other than Microsoft Windows, see the software distribution documentation for SSL/TLS implementation for information about adding a new root CA certificate. For example, OpenSSL and GnuTLS are popular options. Use the implementation method to add trust to the RDS root CA certificate. Microsoft provides instructions for configuring certificates on some systems.

For information about downloading the root certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

For sample scripts that import certificates, see Sample script for importing certificates into your trust store.

> **Note**
>
> When you update the trust store, you can retain older certificates in addition to adding the new certificates.

Compliance program support for Microsoft SQL Server DB instances

AWS Services in scope have been fully assessed by a third-party auditor and result in a certification, attestation of compliance, or Authority to Operate (ATO). For more information, see AWS services in scope by compliance program.

HIPAA support for Microsoft SQL Server DB instances

You can use Amazon RDS for Microsoft SQL Server databases to build HIPAA-compliant applications. You can store healthcare-related information, including protected health information (PHI), under a Business Associate Agreement (BAA) with AWS. For more information, see HIPAA compliance.

Amazon RDS for SQL Server supports HIPAA for the following versions and editions:

- SQL Server 2022 Enterprise, Standard, and Web Editions
- SQL Server 2019 Enterprise, Standard, and Web Editions
- SQL Server 2017 Enterprise, Standard, and Web Editions

- SQL Server 2016 Enterprise, Standard, and Web Editions

To enable HIPAA support on your DB instance, set up the following three components.

Component	Details
Auditing	To set up auditing, set the parameter `rds.sqlserver_audit` to the value `fedramp_hipaa`. If your DB instance is not already using a custom DB parameter group, you must create a custom parameter group and attach it to your DB instance before you can modify the `rds.sqlserver_audit` parameter. For more information, see Parameter groups for Amazon RDS.
Transport encryption	To set up transport encryption, force all connections to your DB instance to use Secure Sockets Layer (SSL). For more information, see Forcing connections to your DB instance to use SSL.
Encryption at rest	To set up encryption at rest, you have two options: 1. If you're running SQL Server 2016–2022 Enterprise Edition or 2022 Standard Edition, you can use Transparent Data Encryption (TDE) to achieve encryption at rest. For more information, see Support for Transparent Data Encryption in SQL Server. 2. You can set up encryption at rest by using AWS Key Management Service (AWS KMS) encryption keys. For more information, see Encrypting Amazon RDS resources.

Microsoft SQL Server versions on Amazon RDS

You can specify any currently supported Microsoft SQL Server version when creating a new DB instance. You can specify the Microsoft SQL Server major version (such as Microsoft SQL Server 14.00), and any supported minor version for the specified major version. If no version is specified, Amazon RDS defaults to a supported version, typically the most recent version. If a major version is

specified but a minor version is not, Amazon RDS defaults to a recent release of the major version you have specified.

The following table shows the supported SQL Server versions for all editions and all AWS Regions, except where noted.

> **Note**
>
> You can also use the describe-db-engine-versions AWS CLI command to see a list of supported versions, as well as defaults for newly created DB instances. You can view the major versions of your SQL Server databases by running the describe-db-major-engine-versions AWS CLI command or by using the DescribeDBMajorEngineVersions RDS API operation.

Major version	Minor version	RDS API `EngineVersion` and CLI `engine-version`
SQL Server 2022	16.00.4236.2 (CU23)	`16.00.4236.2.v1`
	16.00.4230.2 (CU22 GDR)	`16.00.4230.2.v1`
	16.00.4225.2 (CU22)	`16.00.4225.2.v1`
	16.00.4215.2 (CU21)	`16.00.4215.2.v1`
	16.00.4210.1 (CU20 GDR)	`16.00.4210.1.v1`
	16.00.4205.1 (CU20)	`16.00.4205.1.v1`
	16.00.4195.2 (CU19)	`16.00.4195.2.v1`
	16.00.4185.3 (CU18)	`16.00.4185.3.v1`
	16.00.4175.1 (CU17)	`16.00.4175.1.v1`
	16.00.4165.4 (CU16)	`16.00.4165.4.v1`
	16.00.4150.1 (CU15)	`16.00.4150.1.v1`
	16.00.4140.3 (CU14 GDR)	`16.00.4140.3.v1`

Major version	Minor version	RDS API `EngineVersion` and CLI `engine-version`
	16.00.4135.4 (CU14)	`16.00.4135.4.v1`
	16.00.4131.2 (CU13)	`16.00.4131.2.v1`
	16.00.4125.3 (CU13)	`16.00.4125.3.v1`
	16.00.4120.1 (CU12 GDR)	`16.00.4120.1.v1`
	16.00.4115.5 (CU12)	`16.00.4115.5.v1`
	16.00.4105.2 (CU11)	`16.00.4105.2.v1`
	16.00.4095.4 (CU10)	`16.00.4095.4.v1`
	16.00.4085.2 (CU9)	`16.00.4085.2.v1`

Major version	Minor version	RDS API `EngineVersion` and CLI `engine-version`
SQL Server 2019	15.00.4455.2 (CU32 GDR)	`15.00.4455.2.v1`
	15.00.4445.1 (CU32 GDR)	`15.00.4445.1.v1`
	15.00.4440.1 (CU32 GDR)	`15.00.4440.1.v1`
	15.00.4435.7 (CU32)	`15.00.4435.7.v1`
	15.00.4430.1 (CU32)	`15.00.4430.1.v1`
	15.00.4420.2 (CU31)	`15.00.4420.2.v1`
	15.00.4415.2 (CU30)	`15.00.4415.2.v1`
	15.00.4410.1 (CU29 GDR)	`15.00.4410.1.v1`
	15.00.4395.2 (CU28)	`15.00.4395.2.v1`
	15.00.4390.2 (CU28)	`15.00.4390.2.v1`
	15.00.4385.2 (CU28)	`15.00.4385.2.v1`
	15.00.4382.1 (CU27)	`15.00.4382.1.v1`
	15.00.4375.4 (CU27)	`15.00.4375.4.v1`
	15.00.4365.2 (CU26)	`15.00.4365.2.v1`
	15.00.4355.3 (CU25)	`15.00.4355.3.v1`
	15.00.4345.5 (CU24)	`15.00.4345.5.v1`
	15.00.4335.1 (CU23)	`15.00.4335.1.v1`
	15.00.4322.2 (CU22)	`15.00.4322.2.v1`
	15.00.4316.3 (CU21)	`15.00.4316.3.v1`
	15.00.4312.2 (CU20)	`15.00.4312.2.v1`

Major version	Minor version	RDS API `EngineVersion` and CLI `engine-version`
	15.00.4236.7 (CU16)	`15.00.4236.7.v1`
	15.00.4198.2 (CU15)	`15.00.4198.2.v1`
	15.00.4153.1 (CU12)	`15.00.4153.1.v1`
	15.00.4073.23 (CU8)	`15.00.4073.23.v1`
	15.00.4043.16 (CU5)	`15.00.4043.16.v1`

Major version	Minor version	RDS API EngineVersion and CLI engine-version
SQL Server 2017	14.00.3515.1 (CU31 GDR)	14.00.3515.1.v1
	14.00.3505.1 (CU31 GDR)	14.00.3505.1.v1
	14.00.3500.1.(CU31 GDR)	14.00.3500.1.v1
	14.00.3495.9 (CU31 GDR)	14.00.3495.9.v1
	14.00.3485.1 (CU31 GDR)	14.00.3485.1.v1
	14.00.3480.1 (CU31)	14.00.3480.1.v1
	14.00.3475.1 (CU31)	14.00.3475.1.v1
	14.00.3471.2 (CU31)	14.00.3471.2.v1
	14.00.3465.1 (CU31)	14.00.3465.1.v1
	14.00.3460.9 (CU31)	14.00.3460.9.v1
	14.00.3451.2 (CU30)	14.00.3451.2.v1
	14.00.3421.10 (CU27)	14.00.3421.10.v1
	14.00.3401.7 (CU25)	14.00.3401.7.v1
	14.00.3381.3 (CU23)	14.00.3381.3.v1
	14.00.3356.20 (CU22)	14.00.3356.20.v1
	14.00.3294.2 (CU20)	14.00.3294.2.v1
	14.00.3281.6 (CU19)	14.00.3281.6.v1

Major version	Minor version	RDS API `EngineVersion` and CLI `engine-version`
SQL Server 2016	13.00.6475.1 (GDR)	`14.00.6475.1.v1`
	13.00.6470.1 (GDR)	`14.00.6470.1.v1`
	13.00.6465.1 (GDR)	`13.00.6465.1.v1`
	13.00.6460.7 (GDR)	`13.00.6460.7.v1`
	13.00.6455.2 (GDR)	`13.00.6455.2.v1`
	13.00.6450.1 (GDR)	`13.00.6450.1.v1`
	13.00.6445.1 (GDR)	`13.00.6445.1.v1`
	13.00.6441.1 (GDR)	`13.00.6441.1.v1`
	13.00.6435.1 (GDR)	`13.00.6435.1.v1`
	13.00.6430.49 (GDR)	`13.00.6430.49.v1`
	13.00.6419.1 (SP3 + Hotfix)	`13.00.6419.1.v1`
	13.00.6300.2 (SP3)	`13.00.6300.2.v1`

Version management in Amazon RDS

Amazon RDS includes flexible version management that enables you to control when and how your DB instance is patched or upgraded. This enables you to do the following for your DB engine:

- Maintain compatibility with database engine patch versions.
- Test new patch versions to verify that they work with your application before you deploy them in production.
- Plan and perform version upgrades to meet your service level agreements and timing requirements.

Microsoft SQL Server engine patching in Amazon RDS

Amazon RDS periodically aggregates official Microsoft SQL Server database patches into a DB instance engine version that's specific to Amazon RDS. For more information about the Microsoft SQL Server patches in each engine version, see Version and feature support on Amazon RDS.

Currently, you manually perform all engine upgrades on your DB instance. For more information, see Upgrades of the Microsoft SQL Server DB engine.

Deprecation schedule for major engine versions of Microsoft SQL Server on Amazon RDS

The following table displays the planned schedule of deprecations for major engine versions of Microsoft SQL Server.

Date	Information
July 14, 2026	Microsoft will stop critical patch updates for SQL Server 2016. For more informatio Server 2016 in the Microsoft documentation.
July 14, 2026	Amazon RDS plans to end support of Microsoft SQL Server 2016 on RDS for SQL Se remaining instances will be scheduled to migrate to SQL Server 2017 (latest minor more information, see Announcement: Amazon RDS for SQL Server ending suppor Server 2016. To avoid an automatic upgrade from Microsoft SQL Server 2016, you can upgrade a nient to you. For more information, see Upgrading a DB instance engine version.
January 15, 2026	Amazon RDS is starting to disable the creation of new RDS for SQL Server DB instar SQL Server 2016. For more information, see Announcement: Amazon RDS for SQL for Microsoft SQL Server 2016.
July 9, 2024	Microsoft will stop critical patch updates for SQL Server 2014. For more informatio Server 2014 in the Microsoft documentation.
June 1, 2024	Amazon RDS plans to end support of Microsoft SQL Server 2014 on RDS for SQL Se remaining instances will be scheduled to migrate to SQL Server 2016 (latest minor

Date	Information
	more information, see Announcement: Amazon RDS for SQL Server ending suppor major versions. To avoid an automatic upgrade from Microsoft SQL Server 2014, you can upgrade a nient to you. For more information, see Upgrading a DB instance engine version.
July 12, 2022	Microsoft will stop critical patch updates for SQL Server 2012. For more informatio Server 2012 in the Microsoft documentation.
June 1, 2022	Amazon RDS plans to end support of Microsoft SQL Server 2012 on RDS for SQL Se remaining instances will be scheduled to migrate to SQL Server 2014 (latest minor more information, see Announcement: Amazon RDS for SQL Server ending suppor major versions. To avoid an automatic upgrade from Microsoft SQL Server 2012, you can upgrade a nient to you. For more information, see Upgrading a DB instance engine version.
September 1, 2021	Amazon RDS is starting to disable the creation of new RDS for SQL Server DB instar SQL Server 2012. For more information, see Announcement: Amazon RDS for SQL for SQL Server 2012 major versions.
July 12, 2019	The Amazon RDS team deprecated support for Microsoft SQL Server 2008 R2 in Ju instances of Microsoft SQL Server 2008 R2 are migrating to SQL Server 2012 (lates available). To avoid an automatic upgrade from Microsoft SQL Server 2008 R2, you can upgrad convenient to you. For more information, see Upgrading a DB instance engine vers
April 25, 2019	Before the end of April 2019, you will no longer be able to create new Amazon RDS database instances using Microsoft SQL Server 2008R2.

Microsoft SQL Server features on Amazon RDS

The supported SQL Server versions on Amazon RDS include the following features. In general, a version also includes features from the previous versions, unless otherwise noted in the Microsoft documentation.

Topics

- Microsoft SQL Server 2022 features
- Microsoft SQL Server 2019 features
- Microsoft SQL Server 2017 features
- Microsoft SQL Server 2016 features
- Microsoft SQL Server 2014 end of support on Amazon RDS
- Microsoft SQL Server 2012 end of support on Amazon RDS
- Microsoft SQL Server 2008 R2 end of support on Amazon RDS
- Change data capture support for Microsoft SQL Server DB instances
- Features not supported and features with limited support

Microsoft SQL Server 2022 features

SQL Server 2022 includes many new features, such as the following:

- Parameter Sensitive Plan Optimization – allows multiple cached plans for a single parameterized statement, potentially reducing issues with parameter sniffing.
- SQL Server Ledger – provides the ability to cryptographically prove that your data hasn't been altered without authorization.
- Instant file initialization for transaction log file growth events – results in faster execution of log growth events up to 64MB, including for databases with TDE enabled.
- System page latch concurrency enhancements – reduces page latch contention while allocating and deallocating data pages and extents, providing significant performance enhancements to `tempdb` heavy workloads.

For the full list of SQL Server 2022 features, see What's new in SQL Server 2022 (16.x) in the Microsoft documentation.

For a list of unsupported features, see Features not supported and features with limited support.

Microsoft SQL Server 2019 features

SQL Server 2019 includes many new features, such as the following:

- Accelerated database recovery (ADR) – Reduces crash recovery time after a restart or a long-running transaction rollback.
- Intelligent Query Processing (IQP):
  - Row mode memory grant feedback – Corrects excessive grants automatically, that would otherwise result in wasted memory and reduced concurrency.
  - Batch mode on rowstore – Enables batch mode execution for analytic workloads without requiring columnstore indexes.
  - Table variable deferred compilation – Improves plan quality and overall performance for queries that reference table variables.
- Intelligent performance:
  - `OPTIMIZE_FOR_SEQUENTIAL_KEY` index option – Improves throughput for high-concurrency inserts into indexes.
  - Improved indirect checkpoint scalability – Helps databases with heavy DML workloads.
  - Concurrent Page Free Space (PFS) updates – Enables handling as a shared latch rather than an exclusive latch.
- Monitoring improvements:
  - `WAIT_ON_SYNC_STATISTICS_REFRESH` wait type – Shows accumulated instance-level time spent on synchronous statistics refresh operations.
  - Database-scoped configurations – Include `LIGHTWEIGHT_QUERY_PROFILING` and `LAST_QUERY_PLAN_STATS`.
  - Dynamic management functions (DMFs) – Include `sys.dm_exec_query_plan_stats` and `sys.dm_db_page_info`.
- Verbose truncation warnings – The data truncation error message defaults to include table and column names and the truncated value.
- Resumable online index creation – In SQL Server 2017, only resumable online index rebuild is supported.

For the full list of SQL Server 2019 features, see What's new in SQL Server 2019 (15.x) in the Microsoft documentation.

For a list of unsupported features, see Features not supported and features with limited support.

Microsoft SQL Server 2017 features

SQL Server 2017 includes many new features, such as the following:

- Adaptive query processing
- Automatic plan correction (an automatic tuning feature)
- GraphDB
- Resumable index rebuilds

For the full list of SQL Server 2017 features, see What's new in SQL Server 2017 in the Microsoft documentation.

For a list of unsupported features, see Features not supported and features with limited support.

Microsoft SQL Server 2016 features

Amazon RDS supports the following features of SQL Server 2016:

- Always Encrypted
- JSON Support
- Operational Analytics
- Query Store
- Temporal Tables

For the full list of SQL Server 2016 features, see What's new in SQL Server 2016 in the Microsoft documentation.

Microsoft SQL Server 2014 end of support on Amazon RDS

SQL Server 2014 has reached its end of support on Amazon RDS.

RDS is upgrading all existing DB instances that are still using SQL Server 2014 to the latest minor version of SQL Server 2016. For more information, see Version management in Amazon RDS.

Microsoft SQL Server 2012 end of support on Amazon RDS

SQL Server 2012 has reached its end of support on Amazon RDS.

RDS is upgrading all existing DB instances that are still using SQL Server 2012 to the latest minor version of SQL Server 2016. For more information, see Version management in Amazon RDS.

Microsoft SQL Server 2008 R2 end of support on Amazon RDS

SQL Server 2008 R2 has reached its end of support on Amazon RDS.

RDS is upgrading all existing DB instances that are still using SQL Server 2008 R2 to the latest minor version of SQL Server 2012. For more information, see Version management in Amazon RDS.

Change data capture support for Microsoft SQL Server DB instances

Amazon RDS supports change data capture (CDC) for your DB instances running Microsoft SQL Server. CDC captures changes that are made to the data in your tables, and stores metadata about each change that you can access later. For more information, see Change data capture in the Microsoft documentation.

Amazon RDS supports CDC for the following SQL Server editions and versions:

- Microsoft SQL Server Enterprise Edition (All versions)
- Microsoft SQL Server Standard Edition:
  - 2022
  - 2019
  - 2017
  - 2016 version 13.00.4422.0 SP1 CU2 and later

To use CDC with your Amazon RDS DB instances, first enable or disable CDC at the database level by using RDS-provided stored procedures. After that, any user that has the `db_owner` role for that database can use the native Microsoft stored procedures to control CDC on that database. For more information, see Using change data capture for Amazon RDS for SQL Server.

You can use CDC and AWS Database Migration Service to enable ongoing replication from SQL Server DB instances.

Features not supported and features with limited support

The following Microsoft SQL Server features aren't supported on Amazon RDS:

- Backing up to Microsoft Azure Blob Storage
- Buffer pool extension
- Custom password policies
- Data Quality Services
- Database Log Shipping
- Database snapshots (Amazon RDS supports only DB instance snapshots)
- Extended stored procedures, including xp_cmdshell
- FILESTREAM support
- File tables
- Machine Learning and R Services (requires OS access to install it)
- Maintenance plans
- Performance Data Collector
- Policy-Based Management
- PolyBase
- Replication
- Server-level triggers
- Service Broker endpoints
- Stretch database
- TRUSTWORTHY database property (requires sysadmin role)
- T-SQL endpoints (all operations using CREATE ENDPOINT are unavailable)
- WCF Data Services

The following Microsoft SQL Server features have limited support on Amazon RDS:

- Distributed queries/linked servers. For more information, see Implement linked servers with Amazon RDS for Microsoft SQL Server.
- Common Runtime Language (CLR). On RDS for SQL Server 2016 and lower versions, CLR is supported in SAFE mode and using assembly bits only. CLR isn't supported on RDS for SQL

Server 2017 and higher versions. For more information, see Common Runtime Language Integration in the Microsoft documentation.

- Link servers with Oracle OLEDB in Amazon RDS for SQL Server. For more information, see Support for Linked Servers with Oracle OLEDB in Amazon RDS for SQL Server.

The following features aren't supported on Amazon RDS with SQL Server 2022:

- Suspend database for snapshot
- External Data Source
- Backup and restore to S3 compatible object storage
- Object store integration
- TLS 1.3 and MS-TDS 8.0
- Backup compression offloading with QAT
- SQL Server Analysis Services (SSAS)
- Database mirroring with Multi-AZ deployments. SQL Server Always On is the only supported method with Multi-AZ deployments.

Multi-AZ deployments using Microsoft SQL Server Database Mirroring or Always On availability groups

Amazon RDS supports Multi-AZ deployments for DB instances running Microsoft SQL Server by using SQL Server Database Mirroring (DBM) or Always On Availability Groups (AGs). Multi-AZ deployments provide increased availability, data durability, and fault tolerance for DB instances. In the event of planned database maintenance or unplanned service disruption, Amazon RDS automatically fails over to the up-to-date secondary replica so database operations can resume quickly without manual intervention. The primary and secondary instances use the same endpoint, whose physical network address transitions to the passive secondary replica as part of the failover process. You don't have to reconfigure your application when a failover occurs.

Amazon RDS manages failover by actively monitoring your Multi-AZ deployment and initiating a failover when a problem with your primary occurs. Failover doesn't occur unless the standby and primary are fully in sync. Amazon RDS actively maintains your Multi-AZ deployment by automatically repairing unhealthy DB instances and re-establishing synchronous replication. You don't have to manage anything. Amazon RDS handles the primary, the witness, and the

standby instance for you. When you set up SQL Server Multi-AZ, RDS configures passive secondary instances for all of the databases on the instance.

For more information, see Multi-AZ deployments for Amazon RDS for Microsoft SQL Server.

Using Transparent Data Encryption to encrypt data at rest

Amazon RDS supports Microsoft SQL Server Transparent Data Encryption (TDE), which transparently encrypts stored data. Amazon RDS uses option groups to enable and configure these features. For more information about the TDE option, see Support for Transparent Data Encryption in SQL Server.

Functions and stored procedures for Amazon RDS for Microsoft SQL Server

Following, you can find a list of the Amazon RDS functions and stored procedures that help automate SQL Server tasks.

Task type	Procedure or function	Where it's used
Administrative tasks	`rds_drop_database`	Dropping a database in an Amazon RDS for Microsoft SQL Server DB instance
	`rds_failover_time`	Determining the last failover time for Amazon RDS for SQL Server
	`rds_manage_view_db_permission`	Deny or allow viewing database names for Amazon RDS for SQL Server
	`rds_modify_db_name`	Renaming a Amazon RDS for Microsoft SQL Server database in a Multi-AZ deployment

Task type	Procedure or function	Where it's used
	`rds_read_error_log`	Viewing error and agent logs
	`rds_set_configuration`	This operation is used to set various DB instance configurations: • Change data capture for Multi-AZ instances • Setting the retention period for trace and dump files • Compressing backup files
	`rds_set_database_online`	Transitioning a Amazon RDS for SQL Server database from OFFLINE to ONLINE

	`rds_set_system_database_sync_objects`
	
	`rds_fn_get_system_database_sync_objects`
	
	`rds_fn_server_object_last_sync_time`
		Turning on SQL Server Agent job replication

Task type	Procedure or function	Where it's used
	`rds_show_configuration`	To see the values that are set using `rds_set_configuration`, see these topics: • Change data capture for Multi-AZ instances • Setting the retention period for trace and dump files
	`rds_shrink_tempdbfile`	Shrinking the tempdb database
Change data capture (CDC)	`rds_cdc_disable_db`	Disabling CDC
	`rds_cdc_enable_db`	Enabling CDC
Database Mail	`rds_fn_sysmail_allitems`	Viewing messages, logs, and attachments
	`rds_fn_sysmail_event_log`	Viewing messages, logs, and attachments
	`rds_fn_sysmail_mailattachments`	Viewing messages, logs, and attachments
	`rds_sysmail_control`	This operation is used in starting and stopping the mail queue: • Starting the mail queue • Stopping the mail queue

Task type	Procedure or function	Where it's used
	`rds_sysmail_delete_mailitems_sp`	Deleting messages
Native backup and restore	`rds_backup_database`	Backing up a database
	`rds_cancel_task`	Canceling a task
	`rds_finish_restore`	Finishing a database restore
	`rds_restore_database`	Restoring a database
	`rds_restore_log`	Restoring a log
Amazon S3 file transfer	`rds_delete_from_filesystem`	Deleting files on the RDS DB instance
	`rds_download_from_s3`	Downloading files from an Amazon S3 bucket to a SQL Server DB instance
	`rds_gather_file_details`	Listing files on the RDS DB instance

Task type	Procedure or function	Where it's used
	`rds_upload_to_s3`	Uploading files from a SQL Server DB instance to an Amazon S3 bucket
Microsoft Distributed Transaction Coordinator (MSDTC)	`rds_msdtc_transaction_tracing`	Using transaction tracing
SQL Server Audit	`rds_fn_get_audit_file`	Viewing audit logs

Transparent Data Encryption	`rds_backup_tde_certificate`
	`rds_drop_tde_certificate`
	`rds_restore_tde_certificate`
	`rds_fn_list_user_tde_certificates`
		Support for Transparent Data Encryption in SQL Server

Task type	Procedure or function	Where it's used
Microsoft Business Intelligence (MSBI)	`rds_msbi_task`	This operation is used with SQL Server Analysis Services (SSAS): • Deploying SSAS projects on Amazon RDS • Adding a domain user as a database administrator • Backing up an SSAS database • Restoring an SSAS database This operation is also used with SQL Server Integration Services (SSIS): • Administrative permissions on SSISDB • Deploying an SSIS project This operation is also used with SQL Server Reporting Services (SSRS): • Granting access to domain users • Revoking system-level permissions
	`rds_fn_task_status`	This operation shows the status of MSBI tasks: • SSAS: Monitoring the status of a deployment task • SSIS: Monitoring the status of a deployment task • SSRS: Monitoring the status of a task
SSIS	`rds_drop_ssis_database`	Dropping the SSISDB database
	`rds_sqlagent_proxy`	Creating an SSIS proxy

Task type	Procedure or function	Where it's used
SSRS	`rds_drop_ssrs_databases`	Deleting the SSRS databases
Resource Governor	`rds_create_resource_pool`	Create resource Pool
	`rds_alter_resource_pool`	Alter resource pool
	`rds_drop_resource_pool`	Drop resource pool
	`rds_create_workload_group`	Create workload group
	`rds_alter_workload_group`	Alter workload group
	`rds_drop_workload_group`	Drop workload group
	`rds_create_classifier_function`	Create and register classifier function

Task type	Procedure or function	Where it's used
	`rds_drop_classifier_function`	Drop classifier function
	`rds_alter_resource_governor_configuration`	Resource governor configuration changes
	`rds_bind_tempdb_metadata_to_resource_pool`	Bind TempDB to a resource pool
	`rds_unbind_tempdb_metadata_from_resource_pool`	Unbind TempDB from a resource pool
	`rds_cleanup_resource_governor`	Cleanup resource governor

Local time zone for Microsoft SQL Server DB instances

The time zone of an Amazon RDS DB instance running Microsoft SQL Server is set by default. The current default is Coordinated Universal Time (UTC). You can set the time zone of your DB instance to a local time zone instead, to match the time zone of your applications.

You set the time zone when you first create your DB instance. You can create your DB instance by using the AWS Management Console, the Amazon RDS API CreateDBInstance action, or the AWS CLI create-db-instance command.

If your DB instance is part of a Multi-AZ deployment (using SQL Server DBM or AGs), then when you fail over, your time zone remains the local time zone that you set. For more information, see Multi-AZ deployments using Microsoft SQL Server Database Mirroring or Always On availability groups.

When you request a point-in-time restore, you specify the time to restore to. The time is shown in your local time zone. For more information, see Restoring a DB instance to a specified time for Amazon RDS.

The following are limitations to setting the local time zone on your DB instance:

- You can't modify the time zone of an existing SQL Server DB instance.
- You can't restore a snapshot from a DB instance in one time zone to a DB instance in a different time zone.
- We strongly recommend that you don't restore a backup file from one time zone to a different time zone. If you restore a backup file from one time zone to a different time zone, you must audit your queries and applications for the effects of the time zone change. For more information, see Importing and exporting SQL Server databases using native backup and restore.

Supported time zones

You can set your local time zone to one of the values listed in the following table.

Time zone	Standard time offset	Description	Notes
Afghanistan Standard Time	(UTC+04:30)	Kabul	This time zone doesn't observe daylight saving time.
Alaskan Standard Time	(UTC–09:00)	Alaska	
Aleutian Standard Time	(UTC–10:00)	Aleutian Islands	

Time zone	Standard time offset	Description	Notes
Altai Standard Time	(UTC+07:00)	Barnaul, Gorno-Altaysk	
Arab Standard Time	(UTC+03:00)	Kuwait, Riyadh	This time zone doesn't observe daylight saving time.
Arabian Standard Time	(UTC+04:00)	Abu Dhabi, Muscat	
Arabic Standard Time	(UTC+03:00)	Baghdad	This time zone doesn't observe daylight saving time.
Argentina Standard Time	(UTC–03:00)	City of Buenos Aires	This time zone doesn't observe daylight saving time.
Astrakhan Standard Time	(UTC+04:00)	Astrakhan, Ulyanovsk	
Atlantic Standard Time	(UTC–04:00)	Atlantic Time (Canada)	
AUS Central Standard Time	(UTC+09:30)	Darwin	This time zone doesn't observe daylight saving time.
Aus Central W. Standard Time	(UTC+08:45)	Eucla	
AUS Eastern Standard Time	(UTC+10:00)	Canberra, Melbourne, Sydney	

Time zone	Standard time offset	Description	Notes
Azerbaijan Standard Time	(UTC+04:00)	Baku	
Azores Standard Time	(UTC–01:00)	Azores	
Bahia Standard Time	(UTC–03:00)	Salvador	
Bangladesh Standard Time	(UTC+06:00)	Dhaka	This time zone doesn't observe daylight saving time.
Belarus Standard Time	(UTC+03:00)	Minsk	This time zone doesn't observe daylight saving time.
Bougainville Standard Time	(UTC+11:00)	Bougainville Island	
Canada Central Standard Time	(UTC–06:00)	Saskatchewan	This time zone doesn't observe daylight saving time.
Cape Verde Standard Time	(UTC–01:00)	Cabo Verde Is.	This time zone doesn't observe daylight saving time.
Caucasus Standard Time	(UTC+04:00)	Yerevan	
Cen. Australia Standard Time	(UTC+09:30)	Adelaide	
Central America Standard Time	(UTC–06:00)	Central America	This time zone doesn't observe daylight saving time.

Time zone	Standard time offset	Description	Notes
Central Asia Standard Time	(UTC+06:00)	Astana	This time zone doesn't observe daylight saving time.
Central Brazilian Standard Time	(UTC–04:00)	Cuiaba	
Central Europe Standard Time	(UTC+01:00)	Belgrade, Bratislava, Budapest, Ljubljana, Prague	
Central European Standard Time	(UTC+01:00)	Sarajevo, Skopje, Warsaw, Zagreb	
Central Pacific Standard Time	(UTC+11:00)	Solomon Islands, New Caledonia	This time zone doesn't observe daylight saving time.
Central Standard Time	(UTC–06:00)	Central Time (US and Canada)	
Central Standard Time (Mexico)	(UTC–06:00)	Guadalajara, Mexico City, Monterrey	
Chatham Islands Standard Time	(UTC+12:45)	Chatham Islands	
China Standard Time	(UTC+08:00)	Beijing, Chongqing, Hong Kong, Urumqi	This time zone doesn't observe daylight saving time.
Cuba Standard Time	(UTC–05:00)	Havana	

Time zone	Standard time offset	Description	Notes
Dateline Standard Time	(UTC–12:00)	International Date Line West	This time zone doesn't observe daylight saving time.
E. Africa Standard Time	(UTC+03:00)	Nairobi	This time zone doesn't observe daylight saving time.
E. Australia Standard Time	(UTC+10:00)	Brisbane	This time zone doesn't observe daylight saving time.
E. Europe Standard Time	(UTC+02:00)	Chisinau	
E. South America Standard Time	(UTC–03:00)	Brasilia	
Easter Island Standard Time	(UTC–06:00)	Easter Island	
Eastern Standard Time	(UTC–05:00)	Eastern Time (US and Canada)	
Eastern Standard Time (Mexico)	(UTC–05:00)	Chetumal	
Egypt Standard Time	(UTC+02:00)	Cairo	
Ekaterinburg Standard Time	(UTC+05:00)	Ekaterinburg	
Fiji Standard Time	(UTC+12:00)	Fiji	

Time zone	Standard time offset	Description	Notes
FLE Standard Time	(UTC+02:00)	Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius	
Georgian Standard Time	(UTC+04:00)	Tbilisi	This time zone doesn't observe daylight saving time.
GMT Standard Time	(UTC)	Dublin, Edinburgh, Lisbon, London	This time zone isn't the same as Greenwich Mean Time. This time zone does observe daylight saving time.
Greenland Standard Time	(UTC–03:00)	Greenland	
Greenwich Standard Time	(UTC)	Monrovia, Reykjavik	This time zone doesn't observe daylight saving time.
GTB Standard Time	(UTC+02:00)	Athens, Bucharest	
Haiti Standard Time	(UTC–05:00)	Haiti	
Hawaiian Standard Time	(UTC–10:00)	Hawaii	
India Standard Time	(UTC+05:30)	Chennai, Kolkata, Mumbai, New Delhi	This time zone doesn't observe daylight saving time.
Iran Standard Time	(UTC+03:30)	Tehran	

Time zone	Standard time offset	Description	Notes
Israel Standard Time	(UTC+02:00)	Jerusalem	
Jordan Standard Time	(UTC+02:00)	Amman	
Kaliningrad Standard Time	(UTC+02:00)	Kaliningrad	
Kamchatka Standard Time	(UTC+12:00)	Petropavlovsk-Kamchatsky – Old	
Korea Standard Time	(UTC+09:00)	Seoul	This time zone doesn't observe daylight saving time.
Libya Standard Time	(UTC+02:00)	Tripoli	
Line Islands Standard Time	(UTC+14:00)	Kiritimati Island	
Lord Howe Standard Time	(UTC+10:30)	Lord Howe Island	
Magadan Standard Time	(UTC+11:00)	Magadan	This time zone doesn't observe daylight saving time.
Magallanes Standard Time	(UTC–03:00)	Punta Arenas	
Marquesas Standard Time	(UTC–09:30)	Marquesas Islands	
Mauritius Standard Time	(UTC+04:00)	Port Louis	This time zone doesn't observe daylight saving time.
Middle East Standard Time	(UTC+02:00)	Beirut	
Montevideo Standard Time	(UTC–03:00)	Montevideo	

Time zone	Standard time offset	Description	Notes
Morocco Standard Time	(UTC+01:00)	Casablanca	
Mountain Standard Time	(UTC–07:00)	Mountain Time (US and Canada)	
Mountain Standard Time (Mexico)	(UTC–07:00)	Chihuahua, La Paz, Mazatlan	
Myanmar Standard Time	(UTC+06:30)	Yangon (Rangoon)	This time zone doesn't observe daylight saving time.
N. Central Asia Standard Time	(UTC+07:00)	Novosibirsk	
Namibia Standard Time	(UTC+02:00)	Windhoek	
Nepal Standard Time	(UTC+05:45)	Kathmandu	This time zone doesn't observe daylight saving time.
New Zealand Standard Time	(UTC+12:00)	Auckland, Wellington	
Newfoundland Standard Time	(UTC–03:30)	Newfoundland	
Norfolk Standard Time	(UTC+11:00)	Norfolk Island	
North Asia East Standard Time	(UTC+08:00)	Irkutsk	
North Asia Standard Time	(UTC+07:00)	Krasnoyarsk	
North Korea Standard Time	(UTC+09:00)	Pyongyang	

Time zone	Standard time offset	Description	Notes
Omsk Standard Time	(UTC+06:00)	Omsk	
Pacific SA Standard Time	(UTC–03:00)	Santiago	
Pacific Standard Time	(UTC–08:00)	Pacific Time (US and Canada)	
Pacific Standard Time (Mexico)	(UTC–08:00)	Baja California	
Pakistan Standard Time	(UTC+05:00)	Islamabad, Karachi	This time zone doesn't observe daylight saving time.
Paraguay Standard Time	(UTC–04:00)	Asuncion	
Romance Standard Time	(UTC+01:00)	Brussels, Copenhagen, Madrid, Paris	
Russia Time Zone 10	(UTC+11:00)	Chokurdakh	
Russia Time Zone 11	(UTC+12:00)	Anadyr, Petropavlovsk-Kamchatsky	
Russia Time Zone 3	(UTC+04:00)	Izhevsk, Samara	
Russian Standard Time	(UTC+03:00)	Moscow, St. Petersburg, Volgograd	This time zone doesn't observe daylight saving time.

Time zone	Standard time offset	Description	Notes
SA Eastern Standard Time	(UTC–03:00)	Cayenne, Fortaleza	This time zone doesn't observe daylight saving time.
SA Pacific Standard Time	(UTC–05:00)	Bogota, Lima, Quito, Rio Branco	This time zone doesn't observe daylight saving time.
SA Western Standard Time	(UTC–04:00)	Georgetown, La Paz, Manaus, San Juan	This time zone doesn't observe daylight saving time.
Saint Pierre Standard Time	(UTC–03:00)	Saint Pierre and Miquelon	
Sakhalin Standard Time	(UTC+11:00)	Sakhalin	
Samoa Standard Time	(UTC+13:00)	Samoa	
Sao Tome Standard Time	(UTC+01:00)	Sao Tome	
Saratov Standard Time	(UTC+04:00)	Saratov	
SE Asia Standard Time	(UTC+07:00)	Bangkok, Hanoi, Jakarta	This time zone doesn't observe daylight saving time.
Singapore Standard Time	(UTC+08:00)	Kuala Lumpur, Singapore	This time zone doesn't observe daylight saving time.

Time zone	Standard time offset	Description	Notes
South Africa Standard Time	(UTC+02:00)	Harare, Pretoria	This time zone doesn't observe daylight saving time.
Sri Lanka Standard Time	(UTC+05:30)	Sri Jayawarde nepura	This time zone doesn't observe daylight saving time.
Sudan Standard Time	(UTC+02:00)	Khartoum	
Syria Standard Time	(UTC+02:00)	Damascus	
Taipei Standard Time	(UTC+08:00)	Taipei	This time zone doesn't observe daylight saving time.
Tasmania Standard Time	(UTC+10:00)	Hobart	
Tocantins Standard Time	(UTC–03:00)	Araguaina	
Tokyo Standard Time	(UTC+09:00)	Osaka, Sapporo, Tokyo	This time zone doesn't observe daylight saving time.
Tomsk Standard Time	(UTC+07:00)	Tomsk	
Tonga Standard Time	(UTC+13:00)	Nuku'alofa	This time zone doesn't observe daylight saving time.
Transbaikal Standard Time	(UTC+09:00)	Chita	

Time zone	Standard time offset	Description	Notes
Turkey Standard Time	(UTC+03:00)	Istanbul	
Turks And Caicos Standard Time	(UTC–05:00)	Turks and Caicos	
Ulaanbaatar Standard Time	(UTC+08:00)	Ulaanbaatar	This time zone doesn't observe daylight saving time.
US Eastern Standard Time	(UTC–05:00)	Indiana (East)	
US Mountain Standard Time	(UTC–07:00)	Arizona	This time zone doesn't observe daylight saving time.
UTC	UTC	Coordinated Universal Time	This time zone doesn't observe daylight saving time.
UTC–02	(UTC–02:00)	Coordinated Universal Time–02	This time zone doesn't observe daylight saving time.
UTC–08	(UTC–08:00)	Coordinated Universal Time–08	
UTC–09	(UTC–09:00)	Coordinated Universal Time–09	

Time zone	Standard time offset	Description	Notes
UTC–11	(UTC–11:00)	Coordinated Universal Time–11	This time zone doesn't observe daylight saving time.
UTC+12	(UTC+12:00)	Coordinated Universal Time+12	This time zone doesn't observe daylight saving time.
UTC+13	(UTC+13:00)	Coordinated Universal Time+13	
Venezuela Standard Time	(UTC–04:00)	Caracas	This time zone doesn't observe daylight saving time.
Vladivostok Standard Time	(UTC+10:00)	Vladivostok	
Volgograd Standard Time	(UTC+04:00)	Volgograd	
W. Australia Standard Time	(UTC+08:00)	Perth	This time zone doesn't observe daylight saving time.
W. Central Africa Standard Time	(UTC+01:00)	West Central Africa	This time zone doesn't observe daylight saving time.
W. Europe Standard Time	(UTC+01:00)	Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna	

Time zone	Standard time offset	Description	Notes
W. Mongolia Standard Time	(UTC+07:00)	Hovd	
West Asia Standard Time	(UTC+05:00)	Ashgabat, Tashkent	This time zone doesn't observe daylight saving time.
West Bank Standard Time	(UTC+02:00)	Gaza, Hebron	
West Pacific Standard Time	(UTC+10:00)	Guam, Port Moresby	This time zone doesn't observe daylight saving time.
Yakutsk Standard Time	(UTC+09:00)	Yakutsk	

Licensing Microsoft SQL Server on Amazon RDS

When you set up an Amazon RDS DB instance for Microsoft SQL Server, the software license is included.

This means that you don't need to purchase SQL Server licenses separately. AWS holds the license for the SQL Server database software. Amazon RDS pricing includes the software license, underlying hardware resources, and Amazon RDS management capabilities.

Amazon RDS supports the following Microsoft SQL Server editions:

- Enterprise
- Standard
- Web
- Express

> **Note**
>
> SQL Server Web Edition is designed for Web hosters and Web VAPs to host public and internet-accessible web pages, websites, web applications, and web services. This level of support is required for compliance with Microsoft's usage rights. For more information, see AWS service terms.

Amazon RDS supports Multi-AZ deployments for DB instances running Microsoft SQL Server by using SQL Server Database Mirroring (DBM), Always On Availability Groups (AGs), and block level replication for SQL Server Web Edition. There are no additional licensing requirements for Multi-AZ deployments. For more information, see Multi-AZ deployments for Amazon RDS for Microsoft SQL Server.

Restoring license-terminated DB instances

Amazon RDS takes snapshots of license-terminated DB instances. If your instance is terminated for licensing issues, you can restore it from the snapshot to a new DB instance. New DB instances have a license included.

For more information, see Restoring license-terminated DB instances for Amazon RDS for SQL Server.

Development and test

For development and testing scenarios, you can use Express Edition for many development, testing, and other non-production needs. You can also use Developer Edition, which includes all SQL Server Enterprise Edition features but is licensed only for non-production use. You can download and install SQL Server Developer Edition on RDS for SQL Server. For more information, see Working with SQL Server Developer Edition on RDS for SQL Server using a Custom Engine Version (CEV) with Bring Your Own Media (BYOM).You can also download and install SQL Server Developer Edition on RDS Custom for SQL Server using the same approach. For more information, see Preparing a CEV using Bring Your Own Media (BYOM). For more information on the difference between SQL Server editions, see Editions and supported features of SQL Server 2019 in the Microsoft documentation.

Connecting to your Microsoft SQL Server DB instance

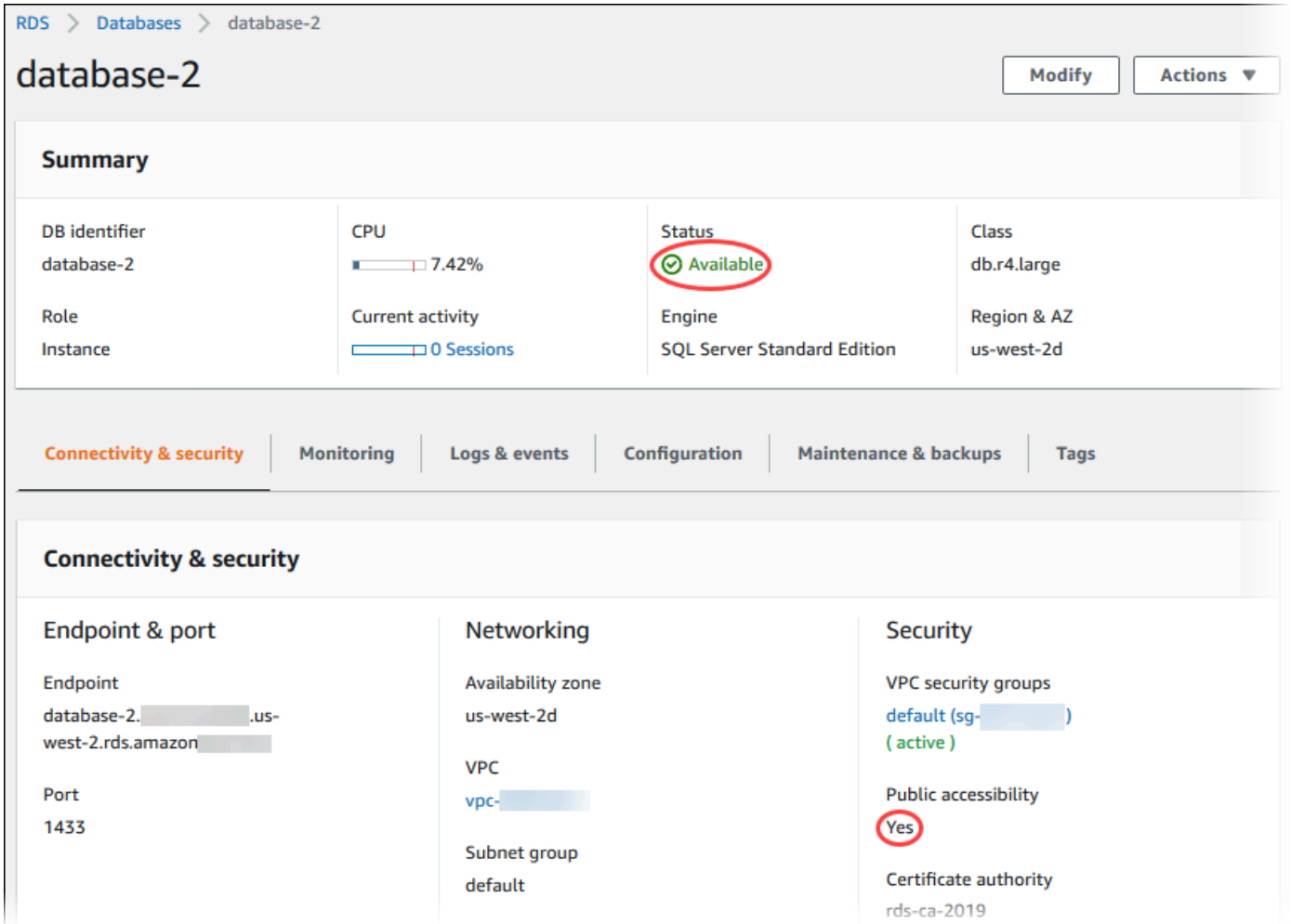
After Amazon RDS provisions your DB instance, you can use any standard SQL client application to connect to the DB instance. In this topic, you connect to your DB instance by using either Microsoft SQL Server Management Studio (SSMS) or SQL Workbench/J.

For an example that walks you through the process of creating and connecting to a sample DB instance, see Creating and connecting to a Microsoft SQL Server DB instance.

Before you connect

Before you can connect to your DB instance, it has to be available and accessible.

1. Make sure that its status is `available`. You can check this on the details page for your instance in the AWS Management Console or by using the describe-db-instances AWS CLI command.



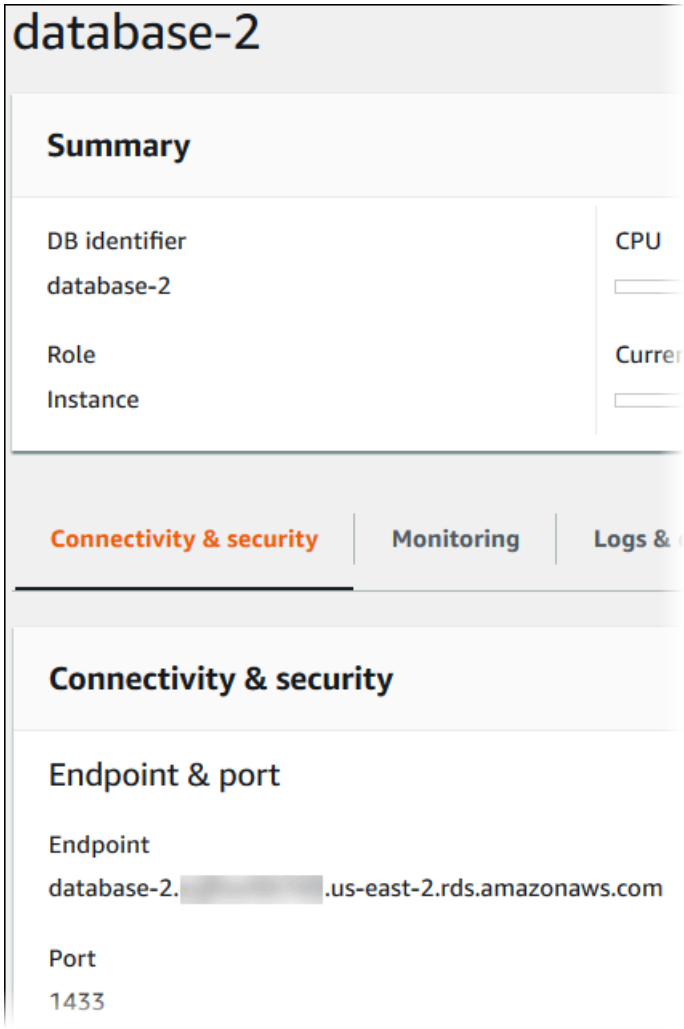
2. Make sure that it is accessible to your source. Depending on your scenario, it may not need to be publicly accessible. For more information, see Amazon VPC and Amazon RDS.
3. Make sure that the inbound rules of your VPC security group allow access to your DB instance. For more information, see Can't connect to Amazon RDS DB instance.

Finding the DB instance endpoint and port number

You need both the endpoint and the port number to connect to the DB instance.

To find the endpoint and port

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region of your DB instance.
3. Find the Domain Name System (DNS) name (endpoint) and port number for your DB instance:
    a. Open the RDS console and choose **Databases** to display a list of your DB instances.
    b. Choose the SQL Server DB instance name to display its details.
    c. On the **Connectivity & security** tab, copy the endpoint.



d. Note the port number.

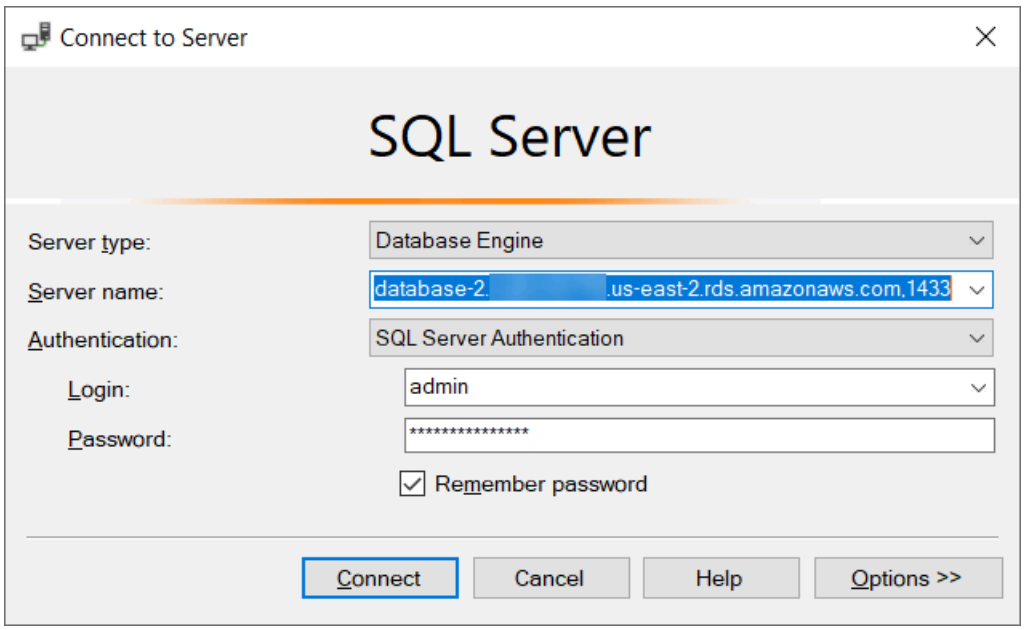
Connecting to your DB instance with Microsoft SQL Server Management Studio

In this procedure, you connect to your sample DB instance by using Microsoft SQL Server Management Studio (SSMS). To download a standalone version of this utility, see Download SQL Server Management Studio (SSMS) in the Microsoft documentation.

To connect to a DB instance using SSMS

1. Start SQL Server Management Studio.

   The **Connect to Server** dialog box appears.



2. Provide the information for your DB instance:

   a. For **Server type**, choose **Database Engine**.

   b. For **Server name**, enter the DNS name (endpoint) and port number of your DB instance, separated by a comma.

   > **Important**
   >
   > Change the colon between the endpoint and port number to a comma.

   Your server name should look like the following example.

   ```
   database-2.cg034itsfake.us-east-1.rds.amazonaws.com,1433
   ```

   c. For **Authentication**, choose **SQL Server Authentication**.

   d. For **Login**, enter the master user name for your DB instance.

   e. For **Password**, enter the password for your DB instance.

3. Choose **Connect**.

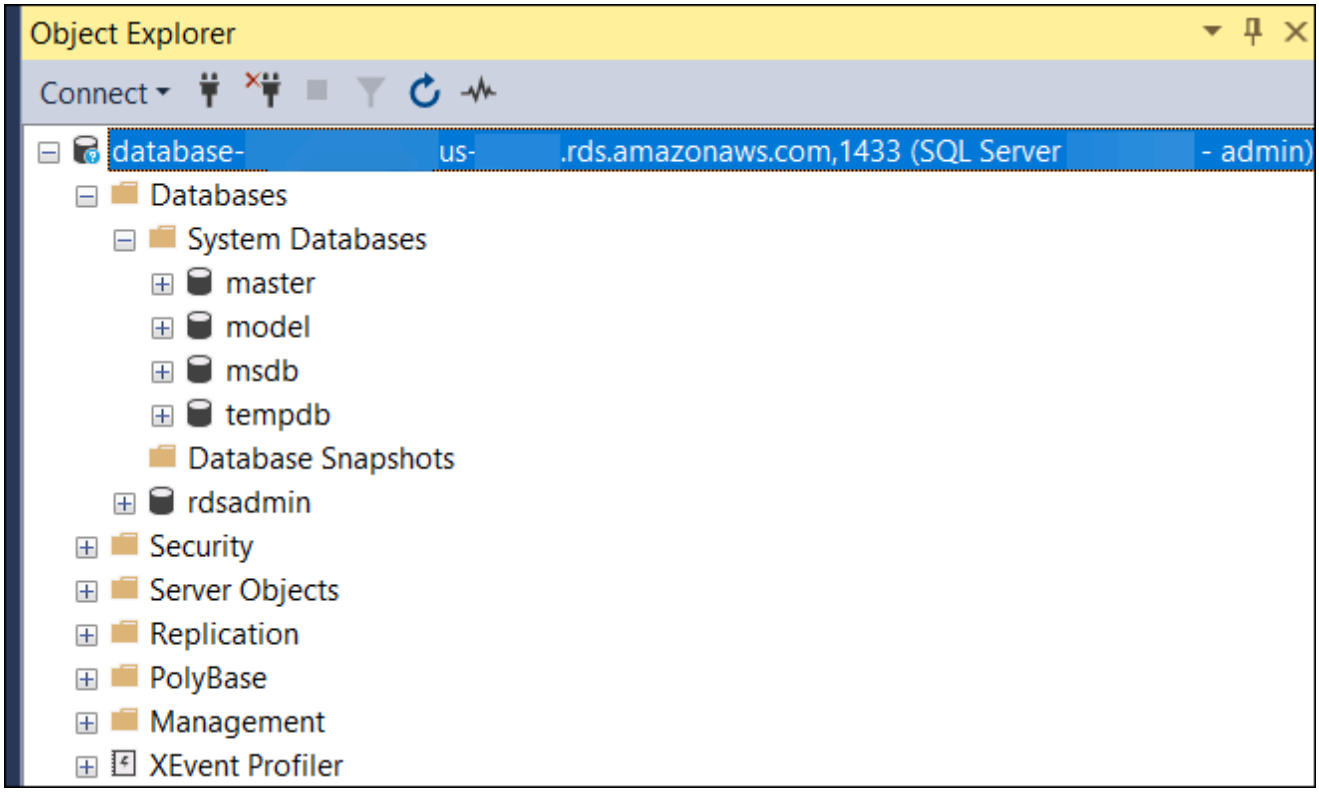
   After a few moments, SSMS connects to your DB instance.

   If you can't connect to your DB instance, see Security group considerations and Troubleshooting connections to your SQL Server DB instance.

4. Your SQL Server DB instance comes with SQL Server's standard built-in system databases (`master`, `model`, `msdb`, and `tempdb`). To explore the system databases, do the following:

   a. In SSMS, on the **View** menu, choose **Object Explorer**.

   b. Expand your DB instance, expand **Databases**, and then expand **System Databases**.



5. Your SQL Server DB instance also comes with a database named `rdsadmin`. Amazon RDS uses this database to store the objects that it uses to manage your database. The `rdsadmin` database also includes stored procedures that you can run to perform advanced tasks. For more information, see Common DBA tasks for Amazon RDS for Microsoft SQL Server.

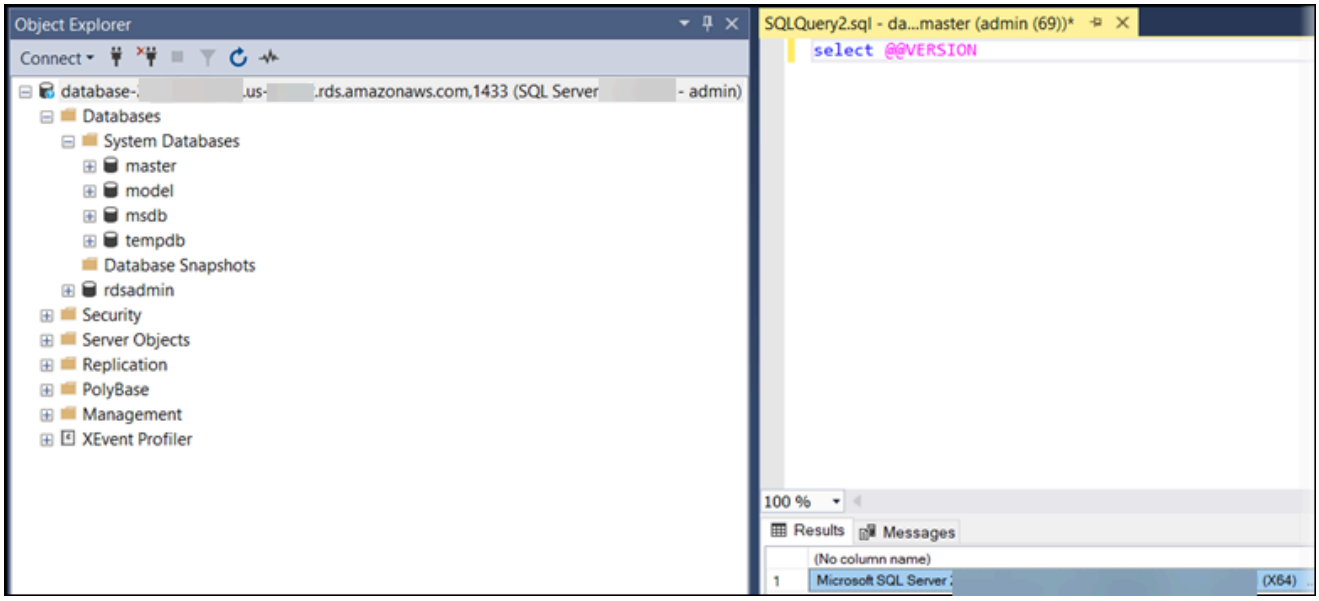
6. You can now start creating your own databases and running queries against your DB instance and databases as usual. To run a test query against your DB instance, do the following:

   a. In SSMS, on the **File** menu point to **New** and then choose **Query with Current Connection**.

   b. Enter the following SQL query.

```
select @@VERSION
```

   c. Run the query. SSMS returns the SQL Server version of your Amazon RDS DB instance.



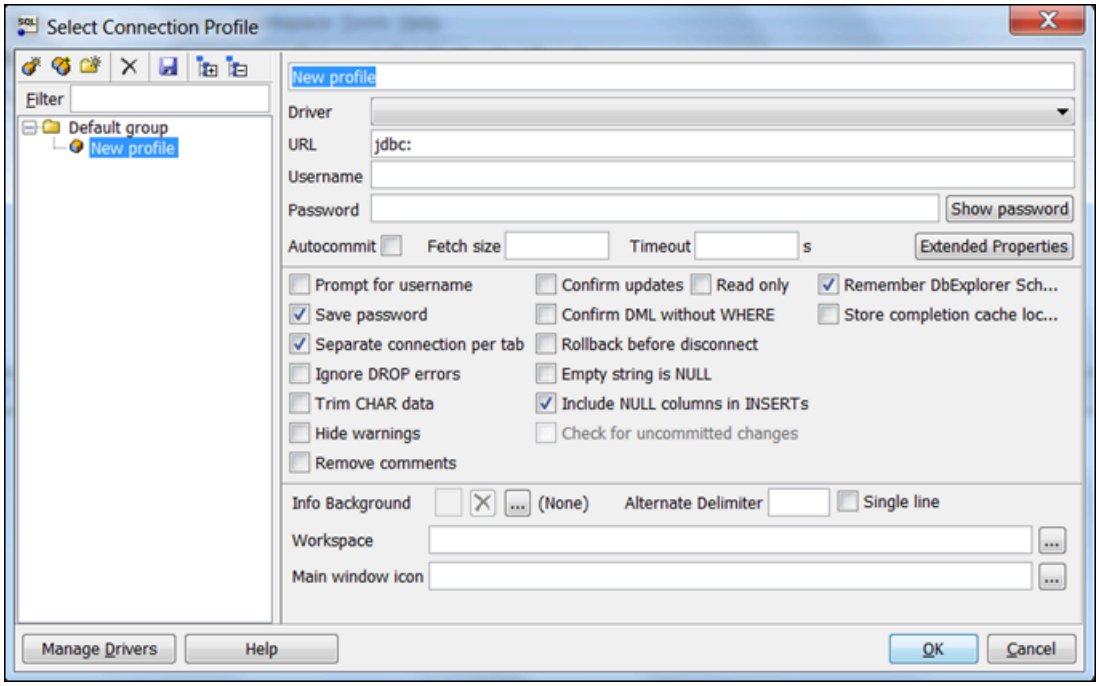
Connecting to your DB instance with SQL Workbench/J

This example shows how to connect to a DB instance running the Microsoft SQL Server database engine by using the SQL Workbench/J database tool. To download SQL Workbench/J, see SQL Workbench/J.

SQL Workbench/J uses JDBC to connect to your DB instance. You also need the JDBC driver for SQL Server. To download this driver, see Download Microsoft JDBC Driver for SQL Server.

To connect to a DB instance using SQL Workbench/J

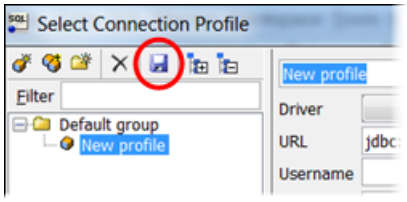
1. Open SQL Workbench/J. The **Select Connection Profile** dialog box appears, as shown following.



2. In the first box at the top of the dialog box, enter a name for the profile.
3. For **Driver**, choose **`SQL JDBC 4.0`**.
4. For **URL**, enter **`jdbc:sqlserver://`**, then enter the endpoint of your DB instance. For example, the URL value might be the following.

```
jdbc:sqlserver://sqlsvr-pdz.abcd12340.us-west-2.rds.amazonaws.com:1433
```

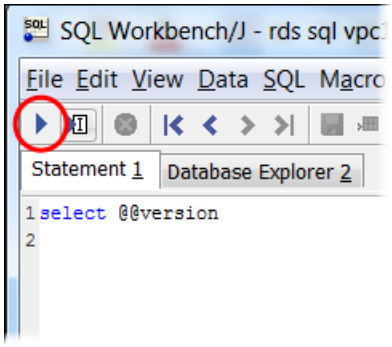
5. For **Username**, enter the master user name for the DB instance.
6. For **Password**, enter the password for the master user.
7. Choose the save icon in the dialog toolbar, as shown following.



8. Choose **OK**. After a few moments, SQL Workbench/J connects to your DB instance. If you can't connect to your DB instance, see Security group considerations and Troubleshooting connections to your SQL Server DB instance.
9. In the query pane, enter the following SQL query.

```
select @@VERSION
```

10. Choose the `Execute` icon in the toolbar, as shown following.



The query returns the version information for your DB instance, similar to the following.

```
Microsoft SQL Server 2017 (RTM-CU22) (KB4577467) - 14.0.3356.20 (X64)
```

Security group considerations

To connect to your DB instance, your DB instance must be associated with a security group. This security group contains the IP addresses and network configuration that you use to access the DB instance. You might have associated your DB instance with an appropriate security group when you created your DB instance. If you assigned a default, no-configured security group when you created your DB instance, your DB instance firewall prevents connections.

In some cases, you might need to create a new security group to make access possible. For instructions on creating a new security group, see Controlling access with security groups. For a topic that walks you through the process of setting up rules for your VPC security group, see Tutorial: Create a VPC for use with a DB instance (IPv4 only).

After you have created the new security group, modify your DB instance to associate it with the security group. For more information, see Modifying an Amazon RDS DB instance.

You can enhance security by using SSL to encrypt connections to your DB instance. For more information, see Using SSL with a Microsoft SQL Server DB instance.

Troubleshooting connections to your SQL Server DB instance

The following table shows error messages that you might encounter when you attempt to connect to your SQL Server DB instance.

Issue	Troubleshooting suggestions
Could not open a connection to SQL Server – Microsoft SQL Server, Error: 53	Make sure that you specified the server name correctly. For **Server name**, enter the DNS name and port number of your sample DB instance, separated by a comma. **Important** If you have a colon between the DNS name and port number, change the colon to a comma. Your server name should look like the following example. `sample-instance.cg034itsfake.us-east-1.rds.amazonaws.com,1433`
No connection could be made because the target machine actively refused it – Microsoft SQL Server, Error: 10061	You were able to reach the DB instance but the connection was refused. This issue is usually caused by specifying the user name or password incorrectly. Verify the user name and password, then retry.
A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible... The wait operation timed out – Microsoft SQL Server, Error: 258	The access rules enforced by your local firewall and the IP addresses authorized to access your DB instance might not match. The problem is most likely the inbound rules in your security group. For more information, see Security in Amazon RDS . Your database instance must be publicly accessible. To connect to it from outside of the VPC, the instance must have a public IP address assigned.

Note

For more information on connection issues, see Can't connect to Amazon RDS DB instance.

Working with SQL Server Developer Edition on RDS for SQL Server

RDS for SQL Server supports SQL Server Developer Edition. Developer Edition includes all SQL Server Enterprise Edition features but is licensed only for non-production use. You can create RDS for SQL Server Developer Edition instances using your own installation media through the custom engine version (CEV) feature.

Benefits

You can use RDS for SQL Server Developer Edition to:

- Lower costs in development and test environments while maintaining feature parity with production databases.
- Access Enterprise Edition capabilities in non-production environments without Enterprise licensing fees.
- Use Amazon RDS-automated management features, including backups, patching, and monitoring.

> **Note**
>
> SQL Server Developer Edition is licensed for development and testing purposes only and cannot be used in production environments.

Region availability

RDS for SQL Server Developer Edition is available in the following AWS Regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- US West (N. California)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)

- Asia Pacific (Singapore)
- Asia Pacific (Osaka)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Europe (Ireland)
- Europe (Frankfurt)
- Europe (London)
- Europe (Stockholm)
- Europe (Paris)
- Canada (Central)
- South America (São Paulo)
- Africa (Cape Town)

Licensing and usage

SQL Server Developer Edition is licensed by Microsoft for development and test environments only. You cannot use Developer Edition as a production server. When you use SQL Server Developer Edition on Amazon RDS, you are responsible for complying with Microsoft's SQL Server Developer Edition licensing terms. You pay only for the AWS infrastructure costs - there is no additional SQL Server licensing fee. For pricing details, see RDS for SQL Server pricing.

Prerequisites

Before using SQL Server Developer Edition on RDS for SQL Server, ensure you have the following requirements:

- You must obtain the installation binaries directly from Microsoft and ensure compliance with Microsoft's licensing terms.
- You must have access to use the following resources to create a Developer Edition DB instance:
  - AWS account with `AmazonRDSFullAccess` and `s3:GetObject` permissions.
- An Amazon S3 bucket is required for storing installation media. You will need an ISO and cumulative update file to upload to the Amazon S3 bucket as part of CEV creation. For more information, see Uploading installation media to an Amazon S3 bucket.

- All installation media files must reside within the same Amazon S3 bucket and the same folder path within that Amazon S3 bucket in the same Region where the custom engine version is created.

Supported versions

Developer Edition on RDS for SQL Server supports the following versions:

- SQL Server 2022 CU 21 (16.00.4215.2)
- SQL Server 2019 CU 32 GDR (15.00.4455.2)

To list all supported engine versions for Developer Edition CEV creation, use the following AWS CLI command:

```
aws rds describe-db-engine-versions --engine sqlserver-dev-ee --output json --
query "{DBEngineVersions: DBEngineVersions[?Status=='requires-custom-engine-
version'].{Engine: Engine, EngineVersion: EngineVersion, Status: Status,
 DBEngineVersionDescription: DBEngineVersionDescription}}"
```

The command returns output similar to the following example:

```
{
    "DBEngineVersions": [
        {
            "Engine": "sqlserver-dev-ee",
            "EngineVersion": "16.00.4215.2.v1",
            "Status": "requires-custom-engine-version",
            "DBEngineDescription": "Microsoft SQL Server Enterprise Developer Edition",
            "DBEngineVersionDescription": "SQL Server 2022 16.00.4215.2.v1"
        }
    ]
}
```

The engine version status as `requires_custom_engine_version` identifies template engine versions that are supported. These templates show which SQL Server versions you can import.

Limitations

The following limitations apply to SQL Server Developer Edition on Amazon RDS:

- Currently only supported on M6i and R6i instance classes.
- Multi-AZ deployments and read replicas are not supported.
- You must provide and manage your own SQL Server installation media.
- Custom engine versions for SQL Server Developer Edition (sqlserver-dev-ee) cannot be shared cross-Region or cross-account.

Preparing a CEV for RDS for SQL Server

Prerequisites

Before creating a custom engine version, make sure you have completed the following prerequisites:

Prepare SQL Server Developer Edition installation media

You must obtain the SQL Server Developer Edition installation media from Microsoft and prepare it for upload to S3.

To download installation media from Microsoft

1. **Option A:** Use your Visual Studio subscription to download the Developer Edition ISO. Only the English version is supported.
2. **Option B: Using SQL Server Installer**
    a. Download the SQL Server Developer Edition installer.
    b. Run the installer and choose **Download Media** to download the full ISO.
    c. Choose **English** as the preferred language.
    d. Choose **ISO** as the media type.
    e. Choose **Download**.

To download cumulative updates

1. Visit the Microsoft Catalog Update page.
2. Find a SQL Server Developer Edition supported by RDS for SQL Server, for example "SQL Server 2022 Cumulative Update".
3. Download the latest supported CU executable file and save it to your machine.

4. Example files: `SQLServer2022-KB5065865-x64.exe` (CU21 for SQL Server 2022)

> **Important**
>
> RDS for SQL Server only supports specific Cumulative Update (CU) versions. You must use the exact version listed in the table below. Do not use newer CU versions even if available from Microsoft, as they may not be compatible with RDS.

Alternatively, you can also download the required Cumulative Update (CU) file directly from the following:

The following table lists the supported SQL Server Developer Edition version and its corresponding Cumulative Update for use with RDS:

SQL Server Version	Supported CU	KB Article	Download File Name
SQL Server 2022	CU21	KB5065865	SQLServer2022-KB5065865-x64.exe
SQL Server 2019	CU32 GDR	KB5068404	SQLServer2019-KB5068404-x64.exe

Creating a custom engine version for RDS for SQL Server

A custom engine version (CEV) for RDS for SQL Server consists of your SQL Server Developer Edition installation media imported into Amazon RDS. It is necessary to upload the base ISO installer and cumulative update files (.exe) to your Amazon S3 bucket. Once uploaded, you should provide the Amazon S3 location to RDS for it to download, validate, and subsequently create your CEV.

Naming limitations

When creating a CEV, you must follow specific naming conventions:

- CEV name must follow the pattern `major-version.minor-version.customized-string`.

- `customized-string` can contain 1-50 alphanumeric characters, underscores, dashes, and periods. For example: `16.00.4215.2.my-dev-cev` for SQL Server 2022.

To list all supported engine versions, use the following command:

```
aws rds describe-db-engine-versions --engine sqlserver-dev-ee --output json --
query "{DBEngineVersions: DBEngineVersions[?Status=='requires-custom-engine-
version'].{Engine: Engine, EngineVersion: EngineVersion, Status: Status,
 DBEngineVersionDescription: DBEngineVersionDescription}}"

{
    "DBEngineVersions": [
        {
            "Engine": "sqlserver-dev-ee",
            "EngineVersion": "16.00.4215.2.v1",
            "Status": "requires-custom-engine-version",
            "DBEngineDescription": "Microsoft SQL Server Enterprise Developer Edition",
            "DBEngineVersionDescription": "SQL Server 2022 16.00.4215.2.v1"
        }
    ]
}
```

AWS CLI

To create the custom engine version

- Use the create-custom-db-engine-version command.

  The following options are required:

  - `--engine`
  - `--engine-version`
  - `--database-installation-files-s3-bucket-name`
  - `--database-installation-files`
  - `--region`

  You can also specify the following options:

  - `--database-installation-files-s3-prefix`

- `--description`
- `--tags`

```
aws rds create-custom-db-engine-version \
--engine sqlserver-dev-ee \
--engine-version 16.00.4215.2.cev-dev-ss2022-cu21 \
--region us-west-2 \
--database-installation-files-s3-bucket-name my-s3-installation-media-bucket \
--database-installation-files-s3-prefix sqlserver-dev-media \
--database-installation-files "SQLServer2022-x64-ENU-Dev.iso" "SQLServer2022-
KB5065865-x64.exe"
```

CEV creation typically takes 15-30 minutes. To monitor the CEV creation progress, use the following command:

```
# Check CEV status
aws rds describe-db-engine-versions \
--engine sqlserver-dev-ee \
--engine-version 16.00.4215.2.my-dev-cev \
--region us-west-2
```

Lifecycle of an RDS for SQL Server CEV

When working with SQL Server Developer Edition on RDS for SQL Server, your custom engine versions transition through various lifecycle states.

Lifecycle State	Description	When It Occurs	Available Actions
pending-validation	Initial state when CEV is created	This is the initial state after creation with the `create-custom-db-engine-version` command.	Monitor status via `describe-db-engine-version` .

Lifecycle State	Description	When It Occurs	Available Actions
validating	CEV Validation state	Amazon RDS is validating your custom engine version (CEV). This asynchronous process may take some time to complete.	Monitor the status until validation is complete.
available	The custom engine version (CEV) validation completed successfully.	Custom Engine Version (CEV) is now available. Amazon RDS successfully validated your SQL Server ISO and cumulative update files. You can now create DB instances using this CEV.	Create DB instances using this CEV
failed	RDS for SQL Server can't create the custom engine version (CEV) because the validation check failed.	ISO and cumulativ e media validation failed.	ISO validation failed. Check the failure reason in `describe-db-engine-version`, fix any file issues such as hash mismatches or corrupted content, and then recreate your custom engine version (CEV).

Lifecycle State	Description	When It Occurs	Available Actions
deleting	The custom engine version (CEV) is deleting	After customer calls `delete-custom-db-engine-version` until deletion workflow completes.	Monitor status via `describe-db-engine-version` .
incompatible-installation-media	Amazon RDS was unable to validate the installation media provided for the custom engine version (CEV)	The custom engine version (CEV) validation has failed. This is a terminal state.	See failureReason via `describe-db-engine-versions` for info on why validation failed; delete CEV.

Describe CEV Status

You can see the state of your CEVs using the AWS CLI:

```
aws rds describe-db-engine-versions \
--engine sqlserver-dev-ee \
--engine-version 16.00.4215.2.my-dev-cev \
--region us-west-2 \
--query 'DBEngineVersions[0].{Version:EngineVersion,Status:Status}'
```

Sample output

```
| DescribeDBEngineVersions                    |
+------------+--------------------------------+
| Status | Version                            |
+------------+--------------------------------+
| available | 16.00.4215.2.cev-dev-ss2022-cu21    |
+------------+--------------------------------+
```

When a CEV shows `failed` status, you can determine the reason using the following command:

```
aws rds describe-db-engine-versions \
--engine sqlserver-dev-ee \
--engine-version 16.00.4215.2.my-dev-cev \
```

```
--region us-west-2 \
--query 'DBEngineVersions[0].
{Version:EngineVersion,Status:Status,FailureReason:FailureReason}'
```

Creating an RDS for SQL Server Developer Edition DB instance

Launching Developer Edition instance on RDS for SQL Server follows a two-step process: first create a CEV with `create-custom-db-engine-version`, Once your custom engine version is in the available state, you can create Amazon RDS database instances using the CEV.

Key differences for Developer Edition instance creation

Parameter	Developer Edition
`--engine`	sqlserver-dev-ee
`--engine-version`	Custom engine version (e.g., 16.00.4215.2.cev-dev-ss2022-cu21)
`--license-model`	bring-your-own-license

AWS CLI

To create a SQL Server Developer Edition DB instance, use the create-db-instance command with the following parameters:

The following options are required:

- `--db-instance-identifier`
- `--db-instance-class`
- `--engine` – `sqlserver-dev-ee`
- `--region`

Examples:

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-instance-identifier my-dev-sqlserver \
```

```
--db-instance-class db.m6i.xlarge \
--engine sqlserver-dev-ee \
--engine-version 16.00.4215.2.my-dev-cev \
--allocated-storage 200 \
--master-username admin \
--master-user-password changeThisPassword \
--license-model bring-your-own-license \
--no-multi-az \
--vpc-security-group-ids sg-xxxxxxxxx \
--db-subnet-group-name my-db-subnet-group \
--backup-retention-period 7 \
--region us-west-2
```

For Windows:

```
aws rds create-db-instance ^
--db-instance-identifier my-dev-sqlserver ^
--db-instance-class db.m6i.xlarge ^
--engine sqlserver-dev-ee ^
--engine-version 16.00.4215.2.cev-dev-ss2022-cu21 ^
--allocated-storage 200 ^
--master-username admin ^
--master-user-password master_user_password ^
--license-model bring-your-own-license ^
--no-multi-az ^
--vpc-security-group-ids sg-xxxxxxxxx ^
--db-subnet-group-name my-db-subnet-group ^
--backup-retention-period 7 ^
--region us-west-2
```

Refer to Creating a DB instance to create using the AWS console.

Applying database minor version upgrades

RDS for SQL Server Developer Edition requires creating a new custom engine version (CEV) with latest cumulative update to apply database minor version upgrade. Database minor version upgrades for SQL Server Developer Edition involve the following steps:

1. Before upgrading, verify current engine version on the instance, identify the target database engine version from Amazon RDS supported versions. For information about what SQL Server versions are available on Amazon RDS, see Working with SQL Server Developer Edition on RDS for SQL Server.

2. Obtain and upload installation media (ISO and CU), then create a new custom engine version.
3. Apply database minor version upgrade by using Amazon RDS `modify-db-instance` with the new CEV.

```
aws rds modify-db-instance \
--db-instance-identifier <instance-id> \
--engine-version <new-cev-version> \
--no-apply-immediately ## use --apply-immediately for immediate update
```

> **Note**
>
> `--no-apply-immediately` (the default) to apply the changes during the next maintenance window.

Viewing and managing custom engine versions

To view all your RDS for SQL Server Developer Edition CEVs, use the `describe-db-engine-versions` command with the `--engine` input as `sqlserver-dev-ee`.

```
aws rds describe-db-engine-versions \
--engine sqlserver-dev-ee \
--include-all \
--region us-west-2
```

To view the details of a specific CEV, use the following command:

```
aws rds describe-db-engine-versions \
--engine sqlserver-dev-ee \
--engine-version 16.00.4215.2.cev-dev-ss2022-cu21 \
--region us-west-2
```

> **Note**
>
> This command only returns CEVs with an `available` status. To view CEVs in validating or other states, include the `--include-all` flag.

Deleting custom engine versions

Before deleting a CEV, make sure it isn't being used by any of the following:

- An Amazon RDS DB instance
- A snapshot of an Amazon RDS DB instance
- An automated backup of an Amazon RDS DB instance

> **Note**
>
> You can't delete a CEV if there are any resources associated with it.

To delete a custom engine version, use the delete-custom-db-engine-version command.

- `--engine`: Specify `sqlserver-dev-ee` for Developer Edition
- `--engine-version`: The exact CEV version identifier to delete
- `--region`: AWS Region where the CEV exists

```
aws rds delete-custom-db-engine-version \
--engine sqlserver-dev-ee \
--engine-version 16.00.4215.2.my-dev-cev \
--region us-west-2
```

To monitor the CEV deletion process, use the `describe-db-engine-versions` command and specify your RDS for SQL Server CEV engine version

```
aws rds describe-db-engine-versions \
--engine sqlserver-dev-ee \
--engine-version 16.00.4215.2.my-dev-cev \
--region us-west-2
```

Status Values:

- `deleting`: CEV deletion in progress
- No results returned: CEV successfully deleted

Troubleshooting SQL Server Developer Edition for RDS for SQL Server

The following table lists some common errors and corresponding solutions when working with SQL Server Developer Edition for RDS for SQL Server

Common Errors and Solutions

Error Code	Description	Solution
InvalidParameterValue	Invalid CEV parameters or file references	Verify file names, paths, and parameter syntax
DBSubnetGroupNotFound	Subnet group doesn't exist	Create subnet group or verify name
InvalidVPCNetworkState	VPC configuration issue	Check VPC, subnets, and routing tables
InvalidEngineVersion	CEV not available or invalid	Verify CEV status and name
InvalidDBInstanceClass	Instance class not supported	Choose supported instance class
CustomDBEngineVersionQuotaExceededFault	You have reached the maximum number of custom engine versions	If required increase service quota, alternatively Delete unused CEVs if required
CreateCustomDBEngineVersionFault	Amazon RDS was unable to access the specified installer file in the Amazon S3 bucket.	Amazon RDS cannot access the SQL Server installation files in the specified Amazon S3 location. Grant the Amazon RDS service principal (rds.amazonaws.com) s3:GetObject permission in your Amazon S3 bucket policy. Validate that the Amazon S3 bucket Region is the same as the Region where you are creating the CEV.

Working with Active Directory with RDS for SQL Server

You can join an RDS for SQL Server DB instance to a Microsoft Active Directory (AD) domain. Your AD domain can be hosted on AWS Managed AD within AWS, or on a self-managed AD in a location of your choice, including your corporate data centers, on AWS EC2, or with other cloud providers.

You can authenticate domain users using NTLM authentication and Kerberos authentication with self-managed Active Directory and AWS Managed Microsoft AD.

In the following sections, you can find information about working with self-managed Active Directory and AWS Managed Active Directory for Microsoft SQL Server on Amazon RDS.

Topics

- Working with self-managed Active Directory with an Amazon RDS for SQL Server DB instance
- Working with AWS Managed Active Directory with RDS for SQL Server

Working with self-managed Active Directory with an Amazon RDS for SQL Server DB instance

Amazon RDS for SQL Server seamlessly integrates with your self-managed Active Directory (AD) domain, regardless of where your AD is hosted - whether in your data center, on Amazon EC2, or with other cloud providers. This integration enables direct user authentication through NTLM or Kerberos protocols, eliminating the need for complex intermediary domains or forest trusts. When you connect to your RDS SQL Server DB instance, authentication requests are securely forwarded to your designated AD domain, maintaining your existing identity management structure while leveraging Amazon RDS's managed database capabilities.

Topics

- Region and version availability
- Requirements
- Considerations
- Setting up self-managed Active Directory
- Joining your DB instance to self-managed Active Directory
- Managing a DB instance in a self-managed Active Directory Domain
- Understanding self-managed Active Directory Domain membership
- Troubleshooting self-managed Active Directory
- Restoring a SQL Server DB instance and then adding it to a self-managed Active Directory domain

Region and version availability

Amazon RDS supports self-managed AD for SQL Server using NTLM and Kerberos in all commercial AWS Regions and AWS GovCloud (US) Regions.

Requirements

Make sure you've met the following requirements before joining an RDS for SQL Server DB instance to your self-managed AD domain.

Topics

- Configure your on-premises AD

- Configure your network connectivity
- Configure your AD domain service account
- Configuring secure communication over LDAPS

Configure your on-premises AD

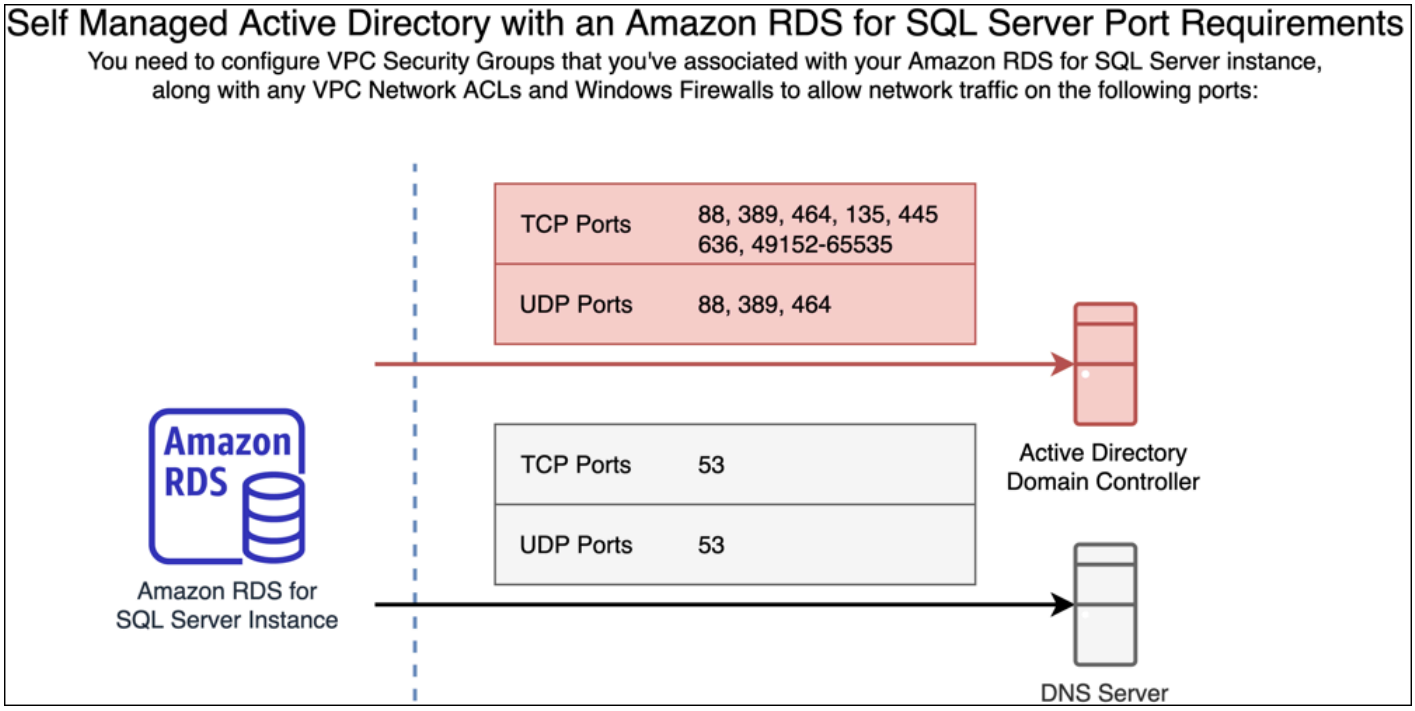
Make sure that you have an on-premises or other self-managed Microsoft AD that you can join the Amazon RDS for SQL Server instance to. Your on-premises AD should have the following configuration:

- If you have AD sites defined, make sure the subnets in the VPC associated with your RDS for SQL Server DB instance are defined in your AD site. Confirm there aren't any conflicts between the subnets in your VPC and the subnets in your other AD sites.
- Your AD domain controller has a domain functional level of Windows Server 2008 R2 or higher.
- Your AD domain name can't be in Single Label Domain (SLD) format. RDS for SQL Server does not support SLD domains.
- The fully qualified domain name (FQDN) for your AD can't exceed 47 characters.

Configure your network connectivity

Make sure that you have met the following network configurations:

- Configure connectivity between the Amazon VPC where you want to create the RDS for SQL Server DB instance and your self-managed AD. You can set up connectivity using AWS Direct Connect, AWS VPN, VPC peering, or AWS Transit Gateway.
- For VPC security groups, the default security group for your default Amazon VPC is already added to your RDS for SQL Server DB instance in the console. Ensure that the security group and the VPC network ACLs for the subnet(s) where you're creating your RDS for SQL Server DB instance allow traffic on the ports and in the directions shown in the following diagram.



The following table identifies the role of each port.

Protocol	Ports	Role
TCP/UDP	53	Domain Name System (DNS)
TCP/UDP	88	Kerberos authentication
TCP/UDP	464	Change/Set password
TCP/UDP	389	Lightweight Directory Access Protocol (LDAP)
TCP	135	Distributed Computing Environment / End Point Mapper (DCE / EPMAP)
TCP	445	Directory Services SMB file sharing

Protocol	Ports	Role
TCP	636	Lightweight Directory Access Protocol over TLS/SSL (LDAPS)
TCP	49152 - 65535	Ephemeral ports for RPC

- Generally, the domain DNS servers are located in the AD domain controllers. You do not need to configure the VPC DHCP option set to use this feature. For more information, see DHCP option sets in the *Amazon VPC User Guide*.

> **Important**
>
> If you're using VPC network ACLs, you must also allow outbound traffic on dynamic ports (49152-65535) from your RDS for SQL Server DB instance. Ensure that these traffic rules are also mirrored on the firewalls that apply to each of the AD domain controllers, DNS servers, and RDS for SQL Server DB instances.
> While VPC security groups require ports to be opened only in the direction that network traffic is initiated, most Windows firewalls and VPC network ACLs require ports to be open in both directions.

Configure your AD domain service account

Make sure that you have met the following requirements for an AD domain service account:

- Make sure that you have a domain service account in your self-managed AD domain with delegated permissions to join computers to the domain. A domain service account is a user account in your self-managed AD that has been delegated permission to perform certain tasks.
- The domain service account needs to be delegated the following permissions in the Organizational Unit (OU) that you're joining your RDS for SQL Server DB instance to:
  - Validated ability to write to the DNS host name
  - Validated ability to write to the service principal name
  - Create and delete computer objects

These represent the minimum set of permissions that are required to join computer objects to your self-managed AD. For more information, see Errors when attempting to join computers to a domain in the Microsoft Windows Server documentation.

- To use Kerberos authentication, you need to provide Service Principal Names (SPNs) and DNS permissions to your AD domain service account:
  - **Write SPN**: Delegate the **Write SPN** permission to the AD domain service account in the OU where you need to join the RDS for SQL Server DB instance. This permissions is different from the validated write SPN.
  - **DNS permissions**: Provide the following permissions to the AD domain service account in the DNS manager at the server level for your domain controller:
    - List contents
    - Read all properties
    - Read permissions

> **Important**
>
> Do not move computer objects that RDS for SQL Server creates in the Organizational Unit after your DB instance is created. Moving the associated objects will cause your RDS for SQL Server DB instance to become misconfigured. If you need to move the computer objects created by Amazon RDS, use the ModifyDBInstance RDS API operation to modify the domain parameters with the desired location of the computer objects.

Configuring secure communication over LDAPS

Communication over LDAPS is recommended for RDS to query and access computer objects as well as SPNs in the domain controller. To use secure LDAP, use a valid SSL certificate on your domain controller that meets the requirements for secure LDAPS. If a valid SSL certificate does not exist on the domain controller, the RDS for SQL Server DB instance defaults to using LDAP. For more information on certificate validity, see Requirements for an LDAPS certificate.

Considerations

When adding an RDS for SQL Server DB instance to a self-managed AD, keep the consider the following:

- Your DB instances sync with AWS's NTP service and not the AD domain's time server. For database connections between linked SQL Server instances within your AD domain, you can only SQL authentication and not Windows authentication.
- Group Policy Object settings from your self-managed AD domain are not propagated to your RDS for SQL Server instances.

Setting up self-managed Active Directory

To set up a self-managed AD, take the following steps.

Topics

- Step 1: Create an Organizational Unit in your AD
- Step 2: Create an AD domain service account in your AD
- Step 3: Delegate control to the AD domain service account
- Step 4: Create an AWS KMS key
- Step 5: Create an AWS secret

Step 1: Create an Organizational Unit in your AD

> **Important**
>
> We recommend creating a dedicated OU and service credential scoped to that OU for any AWS account that owns an RDS for SQL Server DB instance joined your self-managed AD domain. By dedicating an OU and service credential, you can avoid conflicting permissions and follow the principal of least privilege.

To create an OU in your AD

1. Connect to your AD domain as a domain administrator.
2. Open **Active Directory Users and Computers** and select the domain where you want to create your OU.
3. Right-click the domain and choose **New**, then **Organizational Unit**.
4. Enter a name for the OU.
5. Keep the box selected for **Protect container from accidental deletion**.

6. Click **OK**. Your new OU will appear under your domain.

Step 2: Create an AD domain service account in your AD

The domain service account credentials will be used for the secret in AWS Secrets Manager.

To create an AD domain service account in your AD

1. Open **Active Directory Users and Computers** and select the domain and OU where you want to create your user.
2. Right-click the **Users** object and choose **New**, then **User**.
3. Enter a first name, last name, and logon name for the user. Click **Next**.
4. Enter a password for the user. Don't select **"User must change password at next login"**. Don't select **"Account is disabled"**. Click **Next**.
5. Click **OK**. Your new user will appear under your domain.

Step 3: Delegate control to the AD domain service account

To delegate control to the AD domain service account in your domain

1. Open **Active Directory Users and Computers** MMC snap-in and select the domain where you want to create your user.
2. Right-click the OU that you created earlier and choose **Delegate Control**.
3. On the **Delegation of Control Wizard**, click **Next**.
4. On the **Users or Groups** section, click **Add**.
5. On the **Select Users, Computers, or Groups** section, enter the AD domain service account you created and click **Check Names**. If your AD domain service account check is successful, click **OK**.
6. On the **Users or Groups** section, confirm your AD domain service account was added and click **Next**.
7. On the **Tasks to Delegate** section, choose **Create a custom task to delegate** and click **Next**.
8. On the **Active Directory Object Type** section:
    a. Choose **Only the following objects in the folder**.
    b. Select **Computer Objects**.

c. Select **Create selected objects in this folder**.

d. Select **Delete selected objects in this folder** and click **Next**.

9. On the **Permissions** section:

   a. Keep **General** selected.

   b. Select **Validated write to DNS host name**.

   c. Select **Validated write to service principal name** and click **Next**.

   d. To enable Kerberos authentication, keep **Property-specific** selected and select **Write servicePrincipalName** from the list.

10. For **Completing the Delegation of Control Wizard**, review and confirm your settings and click **Finish**.

11. For Kerberos authentication, open the DNS Manager and open **Server** properties.

    a. In the Windows dialog box, type `dnsmgmt.msc`.

    b. Add the AD domain service account under the **Security** tab.

    c. Select the **Read** permission and apply your changes.

Step 4: Create an AWS KMS key

The KMS key is used to encrypt your AWS secret.

To create an AWS KMS key

> **Note**
>
> For **Encryption Key**, don't use the AWS default KMS key. Be sure to create the AWS KMS key in the same AWS account that contains the RDS for SQL Server DB instance that you want to join to your self-managed AD.

1. In the AWS KMS console, choose **Create key**.
2. For **Key Type**, choose **Symmetric**.
3. For **Key Usage**, choose **Encrypt and decrypt**.
4. For **Advanced options**:

   a. For **Key material origin**, choose **KMS**.

b. For **Regionality**, choose **Single-Region key** and click **Next**.

5. For **Alias**, provide a name for the KMS key.
6. (Optional) For **Description**, provide a description of the KMS key.
7. (Optional) For **Tags**, provide a tag the KMS key and click **Next**.
8. For **Key administrators**, provide the name of an IAM user and select it.
9. For **Key deletion**, keep the box selected for **Allow key administrators to delete this key** and click **Next**.
10. For **Key users**, provide the same IAM user from the previous step and select it. Click **Next**.
11. Review the configuration.
12. For **Key policy**, include the following to the policy **Statement**:

```
{
    "Sid": "Allow use of the KMS key on behalf of RDS",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "rds.amazonaws.com"
        ]
    },
    "Action": "kms:Decrypt",
    "Resource": "*"
}
```

13. Click **Finish**.

Step 5: Create an AWS secret

To create a secret

> **Note**
>
> Be sure to create the secret in the same AWS account that contains the RDS for SQL Server DB instance that you want to join to your self-managed AD.

1. In AWS Secrets Manager, choose **Store a new secret**.
2. For **Secret type**, choose **Other type of secret**.

3. For **Key/value pairs**, add your two keys:

   a. For the first key, enter `SELF_MANAGED_ACTIVE_DIRECTORY_USERNAME`.

   b. For the value of the first key, enter only the username (without the domain prefix) of the AD user. Do not include the domain name as this causes instance creation to fail.

   c. For the second key, enter `SELF_MANAGED_ACTIVE_DIRECTORY_PASSWORD`.

   d. For the value of the second key, enter the password that you created for the AD user on your domain.

4. For **Encryption key**, enter the KMS key that you created in a previous step and click **Next**.
5. For **Secret name**, enter a descriptive name that helps you find your secret later.
6. (Optional) For **Description**, enter a description for the secret name.
7. For **Resource permission**, click **Edit**.
8. Add the following policy to the permission policy:

> **Note**
>
> We recommend that you use the `aws:sourceAccount` and `aws:sourceArn` conditions in the policy to avoid the *confused deputy* problem. Use your AWS account for `aws:sourceAccount` and the RDS for SQL Server DB instance ARN for `aws:sourceArn`. For more information, see Preventing cross-service confused deputy problems.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":
    [
        {
            "Effect": "Allow",
            "Principal":
            {
                "Service": "rds.amazonaws.com"
            },
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "*",
```

```
            "Condition":
            {
                "StringEquals":
                {
                    "aws:sourceAccount": "123456789012"
                },
                "ArnLike":
                {
                    "aws:sourceArn": "arn:aws:rds:us-
west-2:123456789012:db:*"
                }
            }
        }
    ]
}
```

9. Click **Save** then click **Next**.
10. For **Configure rotation settings**, keep the default values and choose **Next**.
11. Review the settings for the secret and click **Store**.
12. Choose the secret you created and copy the value for the **Secret ARN**. This will be used in the next step to set up self-managed Active Directory.

Joining your DB instance to self-managed Active Directory

To join your RDS for SQL Server DB instance to your self-managed AD, follow these steps:

Step 1: Create or modify a SQL Server DB instance

You can use the console, CLI, or RDS API to associate an RDS for SQL Server DB instance with a self-managed AD domain. You can do this in one of the following ways:

- Create a new SQL Server DB instance using the console, the create-db-instance CLI command, or the CreateDBInstance RDS API operation.

  For instructions, see Creating an Amazon RDS DB instance.
- Modify an existing SQL Server DB instance using the console, the modify-db-instance CLI command, or the ModifyDBInstance RDS API operation.

  For instructions, see Modifying an Amazon RDS DB instance.

- Restore a SQL Server DB instance from a DB snapshot using the console, the restore-db-instance-from-db-snapshot CLI command, or the RestoreDBInstanceFromDBSnapshot RDS API operation.

  For instructions, see Restoring to a DB instance.

- Restore a SQL Server DB instance to a point-in-time using the console, the restore-db-instance-to-point-in-time CLI command, or the RestoreDBInstanceToPointInTime RDS API operation.

  For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

When you use the AWS CLI, the following parameters are required for the DB instance to be able to use the self-managed AD domain that you created:

- For the `--domain-fqdn` parameter, use the fully qualified domain name (FQDN) of your self-managed AD.
- For the `--domain-ou` parameter, use the OU that you created in your self-managed AD.
- For the `--domain-auth-secret-arn` parameter, use the value of the **Secret ARN** that you created in a previous step.
- For the `--domain-dns-ips` parameter, use the primary and secondary IPv4 addresses of the DNS servers for your self-managed AD. If you don't have a secondary DNS server IP address, enter the primary IP address twice.

The following example CLI commands show how to create, modify, and remove an RDS for SQL Server DB instance with a self-managed AD domain.

> **Important**
>
> If you modify a DB instance to join it to or remove it from a self-managed AD domain, a reboot of the DB instance is required for the modification to take effect. You can choose to apply the changes immediately or wait until the next maintenance window. Choosing the **Apply Immediately** option will cause downtime for a single-AZ DB instance. A multi-AZ DB instance will perform a failover before completing a reboot. For more information, see Using the schedule modifications setting.

The following CLI command creates a new RDS for SQL Server DB instance and joins it to a self-managed AD domain.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier my-DB-instance \
    --db-instance-class db.m5.xlarge \
    --allocated-storage 50 \
    --engine sqlserver-se \
    --engine-version 15.00.4043.16.v1 \
    --license-model license-included \
    --master-username my-master-username \
    --master-user-password my-master-password \
    --domain-fqdn my_AD_domain.my_AD.my_domain \
    --domain-ou OU=my-AD-test-OU,DC=my-AD-test,DC=my-AD,DC=my-domain \
    --domain-auth-secret-arn "arn:aws:secretsmanager:region:account-number:secret:my-
AD-test-secret-123456" \
    --domain-dns-ips "10.11.12.13" "10.11.12.14"
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier my-DB-instance ^
    --db-instance-class db.m5.xlarge ^
    --allocated-storage 50 ^
    --engine sqlserver-se ^
    --engine-version 15.00.4043.16.v1 ^
    --license-model license-included ^
    --master-username my-master-username ^
    --master-user-password my-master-password ^
    --domain-fqdn my-AD-test.my-AD.mydomain ^
    --domain-ou OU=my-AD-test-OU,DC=my-AD-test,DC=my-AD,DC=my-domain ^
    --domain-auth-secret-arn "arn:aws:secretsmanager:region:account-number:secret:my-
AD-test-secret-123456" \ ^
    --domain-dns-ips "10.11.12.13" "10.11.12.14"
```

The following CLI command modifies an existing RDS for SQL Server DB instance to use a self-managed AD domain.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-DB-instance \
    --domain-fqdn my_AD_domain.my_AD.my_domain \
```

```
    --domain-ou OU=my-AD-test-OU,DC=my-AD-test,DC=my-AD,DC=my-domain \
    --domain-auth-secret-arn "arn:aws:secretsmanager:region:account-number:secret:my-
AD-test-secret-123456" \
    --domain-dns-ips "10.11.12.13" "10.11.12.14"
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-DBinstance ^
    --domain-fqdn my_AD_domain.my_AD.my_domain ^
    --domain-ou OU=my-AD-test-OU,DC=my-AD-test,DC=my-AD,DC=my-domain ^
    --domain-auth-secret-arn "arn:aws:secretsmanager:region:account-number:secret:my-
AD-test-secret-123456" ^
    --domain-dns-ips "10.11.12.13" "10.11.12.14"
```

The following CLI command removes an RDS for SQL Server DB instance from a self-managed AD domain.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-DB-instance \
    --disable-domain
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-DB-instance ^
    --disable-domain
```

Step 2: Using Kerberos or NTLM Authentication

NTLM authentication

Each Amazon RDS DB instance has an endpoint and each endpoint has a DNS name and port number for the DB instance. To connect to your DB instance using a SQL client application, you need the DNS name and port number for your DB instance. To authenticate using NTLM authentication, you must connect to the RDS endpoint or the listener endpoint if you are using a Multi-AZ deployment.

During planned database maintenance or unplanned service disruption, Amazon RDS automatically fails over to the up-to-date secondary database so operations can resume quickly without manual intervention. The primary and secondary instances use the same endpoint, whose physical network address transitions to the secondary as part of the failover process. You don't have to reconfigure your application when a failover occurs.

Kerberos authentication

Kerberos-based authentication for RDS for SQL Server requires connections be made to a specific Service Principal Name (SPN). However, after a failover event, the application might not be aware of the new SPN. To address this, RDS for SQL Server offers a Kerberos-based endpoint.

The Kerberos-based endpoint follows a specific format. If your RDS endpoint is *`rds-instance-name`*`.`*`account-region-hash`*`.`*`aws-region`*`.rds.amazonaws.com`, the corresponding Kerberos-based endpoint would be *`rds-instance-name`*`.`*`account-region-hash`*`.`*`aws-region`*`.awsrds.`*`fully qualified domain name (FQDN)`*.

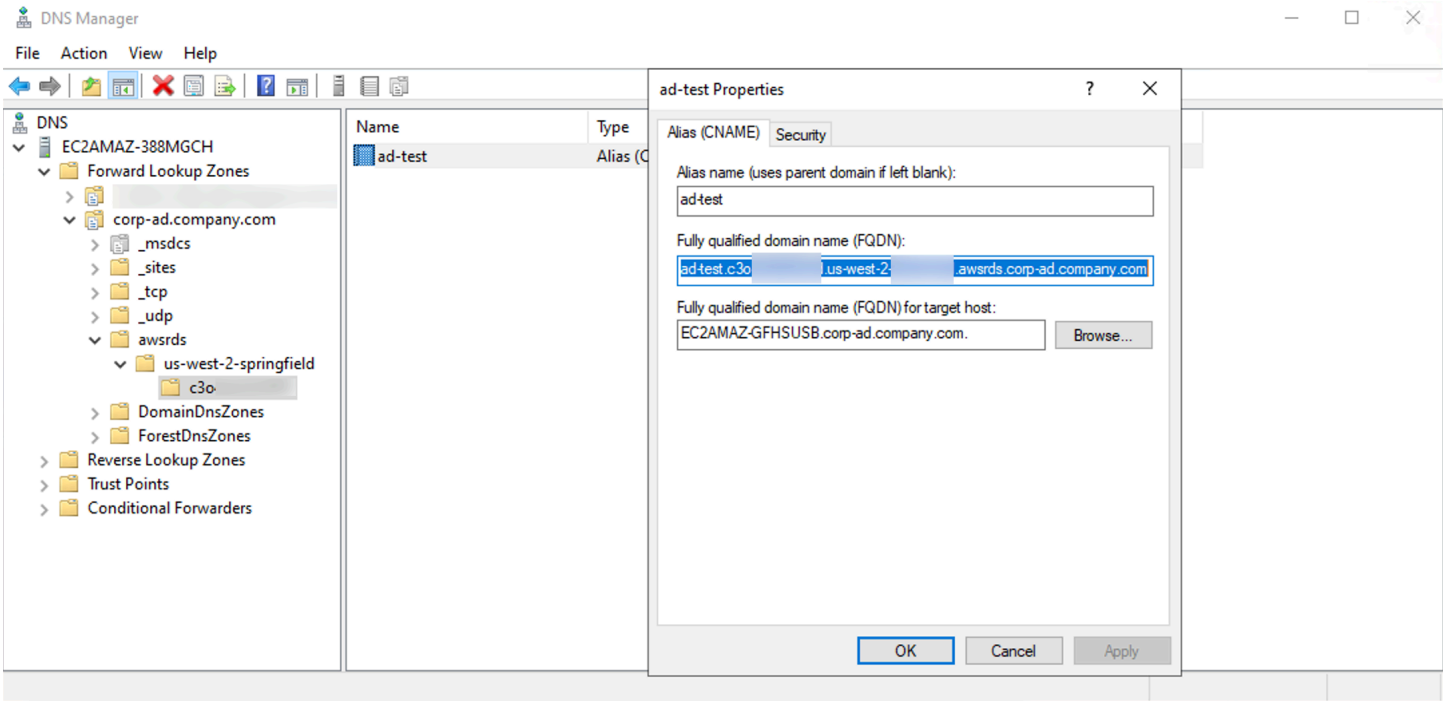
For example, if the RDS endpoint is `ad-test.cocv6zwtircu.us-east-1.rds.amazonaws.com` and the domain name is `corp-ad.company.com`, the Kerberos-based endpoint would be `ad-test.cocv6zwtircu.us-east-1.awsrds.corp-ad.company.com`.

This Kerberos-based endpoint can be used to authenticate with the SQL Server instance using Kerberos, even after a failover event, as the endpoint is automatically updated to point to the new SPN of the primary SQL Server instance.

Finding your CNAME

To find your CNAME, connect to your domain controller and open **DNS Manager**. Navigate to **Forward Lookup Zones** and your FQDN.

Navigate through **awsrds**, **aws-region**, and **account and region specific hash**.



If after connecting CNAME from remote client, an NTLM connection is returned, check if required ports are allowlisted.

To check if your connection is using Kerberos, run the following query:

```
SELECT net_transport, auth_scheme
    FROM sys.dm_exec_connections
    WHERE session_id = @@SSPID;
```

If your instance returns an NTLM connection when you connect to a Kerberos endpoint, verify your network configuration and user configurations. See Configure your network connectivity.

Step 3: Create Windows Authentication SQL Server logins

Use the Amazon RDS master user credentials to connect to the SQL Server DB instance as you do for any other DB instance. Because the DB instance is joined to the self-managed AD domain, you can provision SQL Server logins and users. You do this from the AD users and groups utility in your self-managed AD domain. Database permissions are managed through standard SQL Server permissions granted and revoked to these Windows logins.

In order for a self-managed AD domain service account to authenticate with SQL Server, a SQL Server Windows login must exist for the self-managed AD domain service account or a self-

managed AD group that the user is a member of. Fine-grained access control is handled through granting and revoking permissions on these SQL Server logins. A self-managed AD domain service account that doesn't have a SQL Server login or belong to a self-managed AD group with such a login can't access the SQL Server DB instance.

The ALTER ANY LOGIN permission is required to create a self-managed AD SQL Server login. If you haven't created any logins with this permission, connect as the DB instance's master user using SQL Server Authentication and create your self-managed AD SQL Server logins under the context of the master user.

You can run a data definition language (DDL) command such as the following to create a SQL Server login for an self-managed AD domain service account or group.

> **Note**
>
> Specify users and groups using the pre-Windows 2000 login name in the format *my_AD_domain**my_AD_domain_user*. You can't use a user principal name (UPN) in the format *my_AD_domain_user*@*my_AD_domain*.

```
USE [master]
GO
CREATE LOGIN [my_AD_domain\my_AD_domain_user] FROM WINDOWS WITH DEFAULT_DATABASE =
 [master], DEFAULT_LANGUAGE = [us_english];
GO
```

For more information, see CREATE LOGIN (Transact-SQL) in the Microsoft Developer Network documentation.

Users (both humans and applications) from your domain can now connect to the RDS for SQL Server instance from a self-managed AD domain-joined client machine using Windows authentication.

Managing a DB instance in a self-managed Active Directory Domain

You can use the console, AWS CLI, or the Amazon RDS API to manage your DB instance and its relationship with your self-managed AD domain. For example, you can move the DB instance into, out of, or between domains.

For example, using the Amazon RDS API, you can do the following:

- To reattempt a self-managed domain join for a failed membership, use the ModifyDBInstance API operation and specify the same set of parameters:
  - `--domain-fqdn`
  - `--domain-dns-ips`
  - `--domain-ou`
  - `--domain-auth-secret-arn`
- To remove a DB instance from a self-managed domain, use the `ModifyDBInstance` API operation and specify `--disable-domain` for the domain parameter.
- To move a DB instance from one self-managed domain to another, use the `ModifyDBInstance` API operation and specify the domain parameters for the new domain:
  - `--domain-fqdn`
  - `--domain-dns-ips`
  - `--domain-ou`
  - `--domain-auth-secret-arn`
- To list self-managed AD domain membership for each DB instance, use the DescribeDBInstances API operation.

Understanding self-managed Active Directory Domain membership

After you create or modify your DB instance while specifying AD details, the instance becomes a member of the self-managed AD domain. The AWS console indicates the status of the self-managed Active Directory domain membership for the DB instance. The status of the DB instance can be one of the following:

- **joined** – The instance is a member of the AD domain.
- **joining** – The instance is in the process of becoming a member of the AD domain.
- **pending-join** – The instance membership is pending.
- **pending-maintenance-join** – AWS will attempt to make the instance a member of the AD domain during the next scheduled maintenance window.
- **pending-removal** – The removal of the instance from the AD domain is pending.
- **pending-maintenance-removal** – AWS will attempt to remove the instance from the AD domain during the next scheduled maintenance window.

- **failed** – A configuration problem has prevented the instance from joining the AD domain. Check and fix your configuration before reissuing the instance modify command.
- **removing** – The instance is being removed from the self-managed AD domain.

> **Important**
>
> A request to become a member of a self-managed AD domain can fail because of a network connectivity issue. For example, you might create a DB instance or modify an existing instance and have the attempt fail for the DB instance to become a member of a self-managed AD domain. In this case, either reissue the command to create or modify the DB instance or modify the newly created instance to join the self-managed AD domain.

Troubleshooting self-managed Active Directory

The following are issues you might encounter when you set up or modify self-managed AD.

Error Code	Description	Common causes	Troubleshooting suggestions
Error 2 / 0x2	The system cannot find the file specified.	The format or location for the Organizational Unit (OU) specified with the `–domain-ou` parameter is invalid. The domain service account specified via AWS Secrets Manager lack the permissions required to join the OU.	Review the `–domain-ou` parameter. Ensure the domain service account has the correct permissions to the OU. For more information, see Configure your AD domain service account.
Error 5 / 0x5	Access is denied.	Misconfigured permissions for the domain service account, or the computer account already exists in the domain.	Review the domain service account permissions in the domain, and verify that the RDS computer account is not duplicated in the domain. You can verify the name

Error Code	Description	Common causes	Troubleshooting suggestions
			of the RDS computer account by running `SELECT @@SERVERNAME` on your RDS for SQL Server DB instance. If you are using Multi-AZ, try rebooting with failover and then verify that the RDS computer account again. For more information, see Rebooting a DB instance.
Error 87 / 0x57	The parameter is incorrect.	The domain service account specified via AWS Secrets Manager doesn't have the correct permissions. The user profile may also be corrupted.	Review the requirements for the domain service account. For more information, see Configure your AD domain service account.
Error 234 / 0xEA	Specified Organizational Unit (OU) does not exist.	The OU specified with the `–domain-ou` parameter doesn't exist in your self-managed AD.	Review the `–domain-ou` parameter and ensure the specified OU exists in your self-managed AD.

Error Code	Description	Common causes	Troubleshooting suggestions
Error 1326 / 0x52E	The user name or password is incorrect.	The domain service account credentials provided in AWS Secrets Manager contains an unknown username or bad password. The domain account may also be disabled in your self-managed AD.	Ensure the credentials provided in AWS Secrets Manager are correct and the domain account is enabled in your self-managed AD.
Error 1355 / 0x54B	The specified domain either does not exist or could not be contacted.	The domain is down, the specified set of DNS IPs are unreachable, or the specified FQDN is unreachable.	Review the `–domain-dns-ips` and `–domain-fqdn` parameters to ensure they're correct. Review the networking configuration of your RDS for SQL Server DB instance and ensure your self-managed AD is reachable. For more information, see Configure your network connectivity.
Error 1722 / 0x6BA	The RPC server is unavailable.	There was an issue reaching the RPC service of your AD domain. This might be a service or network issue.	Validate that the RPC service is running on your domain controllers and that the TCP ports 135 and `49152-65535` are reachable on your domain from your RDS for SQL Server DB instance.

Error Code	Description	Common causes	Troubleshooting suggestions
Error 1727 / 0x6BF	The remote procedure call failed and did not execute.	Network connectivity issue or firewall restriction blocking RPC communication to the domain controller.	If using Cross VPC domain join, validate Cross VPC communication is setup correctly with either VPC peering or Transit Gateway. Ensure TCP high ports 49152-65535 are reachable on your domain from your RDS for SQL Server DB instance, including any possible firewall restrictions.
Error 2224 / 0x8B0	The user account already exists.	The computer account that's attempting to be added to your self-managed AD already exists.	Identify the computer account by running `SELECT @@SERVERNAME` on your RDS for SQL Server DB instance and then carefully remove it from your self-managed AD.
Error 2242 / 0x8c2	The password of this user has expired.	The password for the domain service account specified via AWS Secrets Manager has expired.	Update the password for the domain service account used to join your RDS for SQL Server DB instance to your self-managed AD.

After joining your DB instance to a self-managed Active Directory domain, you might receive RDS events related to your domain health.

```
Unhealthy domain state detected while attempt to verify or
```

```
configure your Kerberos endpoint in your domain on
node node_n. message
```

For Multi-AZ instances, you might notice the error reporting for both node1 and node2, which indicates your instance's Kerberos configuration is not ready for failover. In the event of a failover, you might experience authentication difficulties using Kerberos. Resolve the configuration issues to ensure Kerberos setup is valid and up to date. For Multi-AZ instances, no actions are required to use Kerberos authentication on the new primary host given all network and permission configurations are in place.

For Single-AZ instances, node1 is the primary node. If your Kerberos authentication is not working as expected, check the instance events and resolve the configuration issues to ensure Kerberos setup is valid and up to date.

Restoring a SQL Server DB instance and then adding it to a self-managed Active Directory domain

You can restore a DB snapshot or do point-in-time recovery (PITR) for a SQL Server DB instance and then add it to a self-managed Active Directory domain. Once the DB instance is restored, modify the instance using the process explained in Step 1: Create or modify a SQL Server DB instance to add the DB instance to a self-managed AD domain.

Working with AWS Managed Active Directory with RDS for SQL Server

You can use AWS Managed Microsoft AD to authenticate users with Windows Authentication when they connect to your RDS for SQL Server DB instance. The DB instance works with AWS Directory Service for Microsoft Active Directory, also called AWS Managed Microsoft AD, to enable Windows Authentication. When users authenticate with a SQL Server DB instance joined to the trusting domain, authentication requests are forwarded to the domain directory that you create with Directory Service.

Region and version availability

Amazon RDS supports using only AWS Managed Microsoft AD for Windows Authentication. RDS doesn't support using AD Connector. For more information, see the following:

- Application compatibility policy for AWS Managed Microsoft AD
- Application compatibility policy for AD Connector

For information on version and Region availability, see Kerberos authentication with RDS for SQL Server.

Overview of setting up Windows authentication

Amazon RDS uses mixed mode for Windows Authentication. This approach means that the *master user* (the name and password used to create your SQL Server DB instance) uses SQL Authentication. Because the master user account is a privileged credential, you should restrict access to this account.

To get Windows Authentication using an on-premises or self-hosted Microsoft Active Directory, create a forest trust. The trust can be one-way or two-way. For more information on setting up forest trusts using Directory Service, see When to create a trust relationship in the *AWS Directory Service Administration Guide*.

To set up Windows authentication for a SQL Server DB instance, do the following steps, explained in greater detail in Setting up Windows Authentication for SQL Server DB instances:

1. Use AWS Managed Microsoft AD, either from the AWS Management Console or Directory Service API, to create an AWS Managed Microsoft AD directory.
2. If you use the AWS CLI or Amazon RDS API to create your SQL Server DB instance, create an AWS Identity and Access Management (IAM) role. This role uses the managed IAM policy

`AmazonRDSDirectoryServiceAccess` and allows Amazon RDS to make calls to your directory. If you use the console to create your SQL Server DB instance, AWS creates the IAM role for you.

For the role to allow access, the AWS Security Token Service (AWS STS) endpoint must be activated in the AWS Region for your AWS account. AWS STS endpoints are active by default in all AWS Regions, and you can use them without any further actions. For more information, see Managing AWS STS in an AWS Region in the *IAM User Guide*.

3. Create and configure users and groups in the AWS Managed Microsoft AD directory using the Microsoft Active Directory tools. For more information about creating users and groups in your Active Directory, see Manage users and groups in AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.
4. If you plan to locate the directory and the DB instance in different VPCs, enable cross-VPC traffic.
5. Use Amazon RDS to create a new SQL Server DB instance either from the console, AWS CLI, or Amazon RDS API. In the create request, you provide the domain identifier ("`d-*`" identifier) that was generated when you created your directory and the name of the role you created. You can also modify an existing SQL Server DB instance to use Windows Authentication by setting the domain and IAM role parameters for the DB instance.
6. Use the Amazon RDS master user credentials to connect to the SQL Server DB instance as you do any other DB instance. Because the DB instance is joined to the AWS Managed Microsoft AD domain, you can provision SQL Server logins and users from the Active Directory users and groups in their domain. (These are known as SQL Server "Windows" logins.) Database permissions are managed through standard SQL Server permissions granted and revoked to these Windows logins.

When you create a domain-connected RDS for SQL Server DB instance using the Amazon RDS console, AWS automatically creates the `rds-directoryservice-access-role` IAM role. This role is essential for managing domain-connected instances and is required for the following operations:

- Making configuration changes to domain-connected SQL Server instances
- Managing Active Directory integration settings
- Performing maintenance operations on domain-joined instances

Important

If you delete the `rds-directoryservice-access-role` IAM role, you can't make changes to your domain-connected SQL Server instance through the Amazon RDS console or API. Attempting to modify the instance results in an error message stating: You don't have permission to iam:CreateRole. To request access, copy the following text and send it to your AWS administrator.
This error occurs because Amazon RDS needs to recreate the role to manage the domain connection, but lacks the necessary permissions. Additionally, this error is not logged in CloudTrail, which can make troubleshooting more difficult.

If you accidentally delete the `rds-directoryservice-access-role`, you must have `iam:CreateRole` permissions to recreate it before you can make any changes to your domain-connected SQL Server instance. To recreate the role manually, ensure it has the `AmazonRDSDirectoryServiceAccess` managed policy attached and the appropriate trust relationship that allows the RDS service to assume the role.

Creating the endpoint for Kerberos authentication

Kerberos-based authentication requires that the endpoint be the customer-specified host name, a period, and then the fully qualified domain name (FQDN). For example, the following is an example of an endpoint you might use with Kerberos-based authentication. In this example, the SQL Server DB instance host name is `ad-test` and the domain name is `corp-ad.company.com`.

```
ad-test.corp-ad.company.com
```

If you want to make sure your connection is using Kerberos, run the following query:

```
SELECT net_transport, auth_scheme
  FROM sys.dm_exec_connections
 WHERE session_id = @@SPID;
```

Setting up Windows Authentication for SQL Server DB instances

You use AWS Directory Service for Microsoft Active Directory, also called AWS Managed Microsoft AD, to set up Windows Authentication for a SQL Server DB instance. To set up Windows Authentication, take the following steps.

Step 1: Create a directory using the AWS Directory Service for Microsoft Active Directory

Directory Service creates a fully managed, Microsoft Active Directory in the AWS Cloud. When you create an AWS Managed Microsoft AD directory, Directory Service creates two domain controllers and Domain Name Service (DNS) servers on your behalf. The directory servers are created in two subnets in two different Availability Zones within a VPC. This redundancy helps ensure that your directory remains accessible even if a failure occurs.

When you create an AWS Managed Microsoft AD directory, Directory Service performs the following tasks on your behalf:

- Sets up a Microsoft Active Directory within the VPC.
- Creates a directory administrator account with the user name Admin and the specified password. You use this account to manage your directory.
- Creates a security group for the directory controllers.

When you launch an AWS Directory Service for Microsoft Active Directory, AWS creates an Organizational Unit (OU) that contains all your directory's objects. This OU, which has the NetBIOS name that you typed when you created your directory, is located in the domain root. The domain root is owned and managed by AWS.

The *admin* account that was created with your AWS Managed Microsoft AD directory has permissions for the most common administrative activities for your OU:

- Create, update, or delete users, groups, and computers.
- Add resources to your domain such as file or print servers, and then assign permissions for those resources to users and groups in your OU.
- Create additional OUs and containers.
- Delegate authority.
- Create and link group policies.
- Restore deleted objects from the Active Directory Recycle Bin.
- Run AD and DNS Windows PowerShell modules on the Active Directory Web Service.

The admin account also has rights to perform the following domain-wide activities:

- Manage DNS configurations (add, remove, or update records, zones, and forwarders).

- View DNS event logs.
- View security event logs.

To create a directory with AWS Managed Microsoft AD

1. In the Directory Service console navigation pane, choose **Directories** and choose **Set up directory**.
2. Choose **AWS Managed Microsoft AD**. This is the only option currently supported for use with Amazon RDS.
3. Choose **Next**.
4. On the **Enter directory information** page, provide the following information:

   Edition

   Choose the edition that meets your requirements.

   Directory DNS name

   The fully qualified name for the directory, such as `corp.example.com`. Names longer than 47 characters aren't supported by SQL Server.

   Directory NetBIOS name

   An optional short name for the directory, such as `CORP`.

   Directory description

   An optional description for the directory.

   Admin password

   The password for the directory administrator. The directory creation process creates an administrator account with the user name Admin and this password.

   The directory administrator password can't include the word `admin`. The password is case-sensitive and must be 8–64 characters in length. It must also contain at least one character from three of the following four categories:

   - Lowercase letters (a-z)
   - Uppercase letters (A-Z)
   - Numbers (0-9)

- Non-alphanumeric characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Confirm password

Retype the administrator password.

5. Choose **Next**.
6. On the **Choose VPC and subnets** page, provide the following information:

VPC

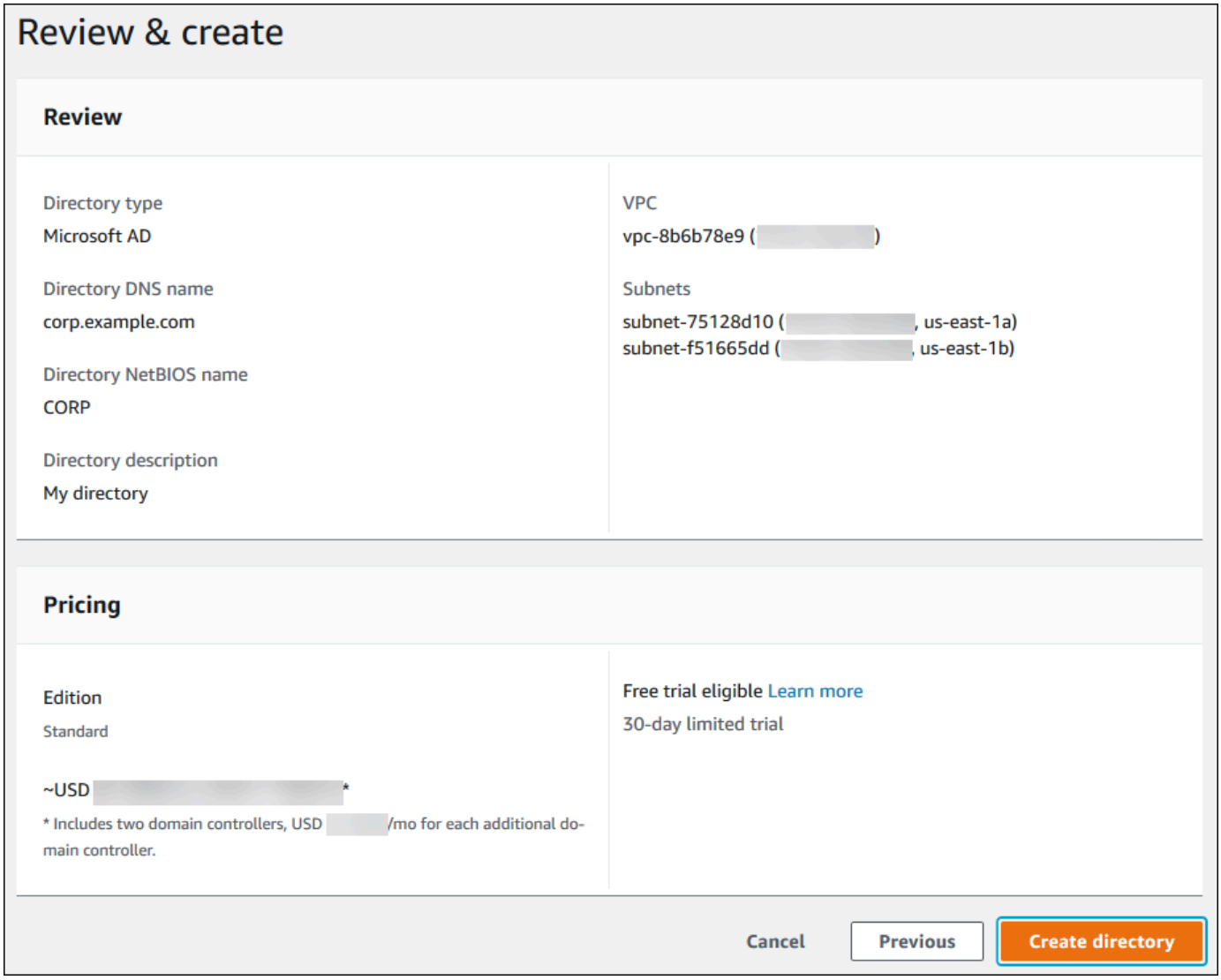
Choose the VPC for the directory.

> **Note**
>
> You can locate the directory and the DB instance in different VPCs, but if you do so, make sure to enable cross-VPC traffic. For more information, see Step 4: Enable cross-VPC traffic between the directory and the DB instance.

Subnets

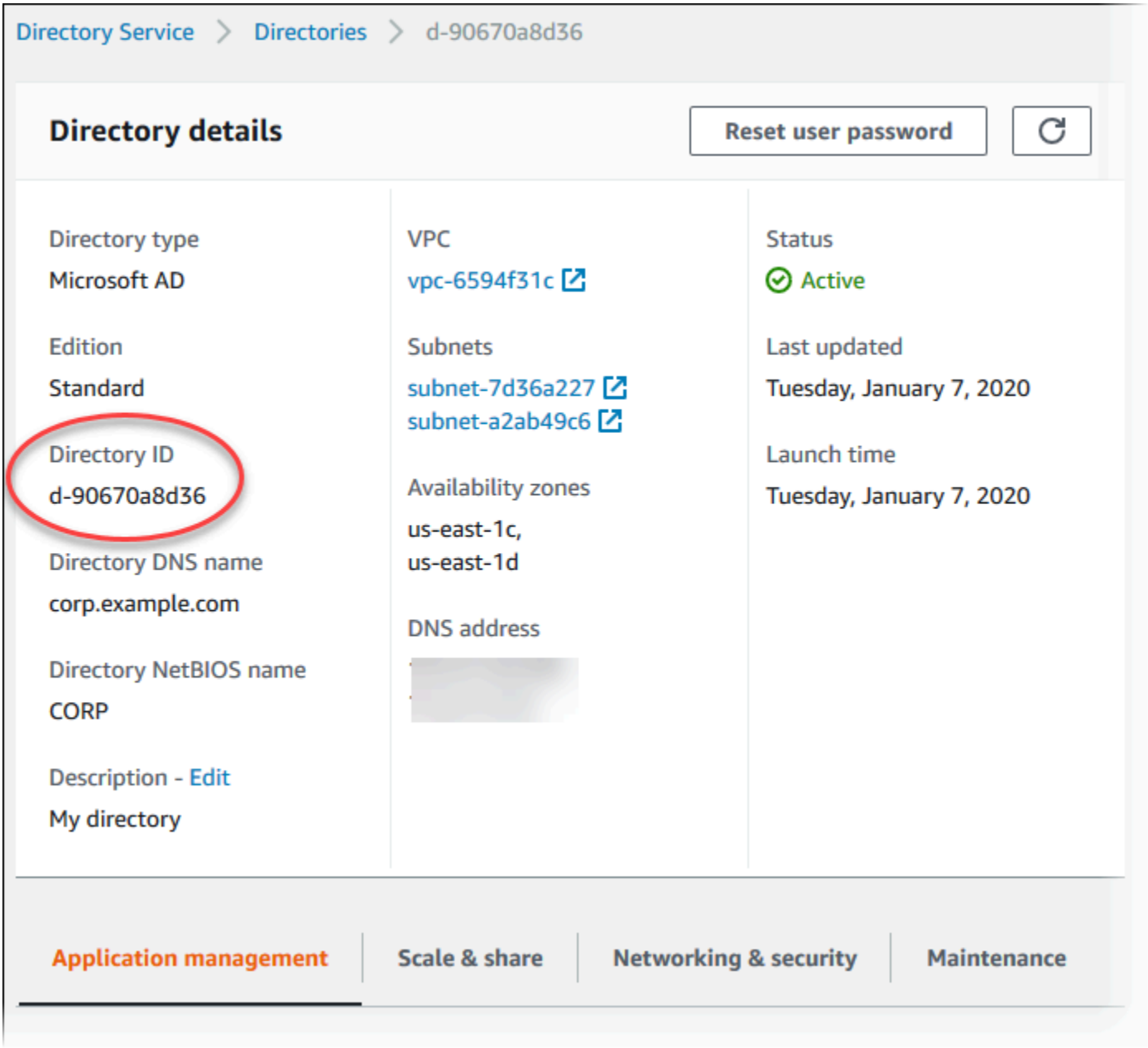
Choose the subnets for the directory servers. The two subnets must be in different Availability Zones.

7. Choose **Next**.
8. Review the directory information. If changes are needed, choose **Previous**. When the information is correct, choose **Create directory**.



It takes several minutes for the directory to be created. When it has been successfully created, the **Status** value changes to **Active**.

To see information about your directory, choose the directory ID in the directory listing. Make a note of the **Directory ID**. You need this value when you create or modify your SQL Server DB instance.



Step 2: Create the IAM role for use by Amazon RDS

If you use the console to create your SQL Server DB instance, you can skip this step. If you use the CLI or RDS API to create your SQL Server DB instance, you must create an IAM role that uses the `AmazonRDSDirectoryServiceAccess` managed IAM policy. This role allows Amazon RDS to make calls to the Directory Service for you.

If you are using a custom policy for joining a domain, rather than using the AWS-managed `AmazonRDSDirectoryServiceAccess` policy, make sure that you allow the

`ds:GetAuthorizedApplicationDetails` action. This requirement is effective starting July 2019, due to a change in the Directory Service API.

The following IAM policy, `AmazonRDSDirectoryServiceAccess`, provides access to Directory Service.

Example IAM policy for providing access to Directory Service

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
            "ds:DescribeDirectories",
            "ds:AuthorizeApplication",
            "ds:UnauthorizeApplication",
            "ds:GetAuthorizedApplicationDetails"
        ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based trust relationships to limit the service's permissions to a specific resource. This is the most effective way to protect against the confused deputy problem.

You might use both global condition context keys and have the `aws:SourceArn` value contain the account ID. In this case, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same statement.

- Use `aws:SourceArn` if you want cross-service access for a single resource.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

In the trust relationship, make sure to use the `aws:SourceArn` global condition context key with the full Amazon Resource Name (ARN) of the resources accessing the role. For Windows Authentication, make sure to include the DB instances, as shown in the following example.

Example trust relationship with global condition context key for Windows Authentication

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": [

 "arn:aws:rds:Region:my_account_ID:db:db_instance_identifier"
                    ]
                }
            }
        }
    ]
}
```

Create an IAM role using this IAM policy and trust relationship. For more information about creating IAM roles, see Creating customer managed policies in the *IAM User Guide*.

Step 3: Create and configure users and groups

You can create users and groups with the Active Directory Users and Computers tool. This tool is one of the Active Directory Domain Services and Active Directory Lightweight Directory Services tools. Users represent individual people or entities that have access to your directory. Groups are very useful for giving or denying privileges to groups of users, rather than having to apply those privileges to each individual user.

To create users and groups in an Directory Service directory, you must be connected to a Windows EC2 instance that is a member of the Directory Service directory. You must also be logged in as a user that has privileges to create users and groups. For more information, see Add users and groups (Simple AD and AWS Managed Microsoft AD) in the *AWS Directory Service Administration Guide*.

Step 4: Enable cross-VPC traffic between the directory and the DB instance

If you plan to locate the directory and the DB instance in the same VPC, skip this step and move on to Step 5: Create or modify a SQL Server DB instance.

If you plan to locate the directory and the DB instance in different VPCs, configure cross-VPC traffic using VPC peering or AWS Transit Gateway.

The following procedure enables traffic between VPCs using VPC peering. Follow the instructions in What is VPC peering? in the *Amazon Virtual Private Cloud Peering Guide*.

To enable cross-VPC traffic using VPC peering

1. Set up appropriate VPC routing rules to ensure that network traffic can flow both ways.
2. Ensure that the DB instance's security group can receive inbound traffic from the directory's security group.
3. Ensure that there is no network access control list (ACL) rule to block traffic.

If a different AWS account owns the directory, you must share the directory.

To share the directory between AWS accounts

1. Start sharing the directory with the AWS account that the DB instance will be created in by following the instructions in Tutorial: Sharing your AWS Managed Microsoft AD directory for seamless EC2 domain-join in the *Directory Service Administration Guide*.
2. Sign in to the Directory Service console using the account for the DB instance, and ensure that the domain has the SHARED status before proceeding.
3. While signed into the Directory Service console using the account for the DB instance, note the **Directory ID** value. You use this directory ID to join the DB instance to the domain.

Step 5: Create or modify a SQL Server DB instance

Create or modify a SQL Server DB instance for use with your directory. You can use the console, CLI, or RDS API to associate a DB instance with a directory. You can do this in one of the following ways:

- Create a new SQL Server DB instance using the console, the create-db-instance CLI command, or the CreateDBInstance RDS API operation.

  For instructions, see Creating an Amazon RDS DB instance.
- Modify an existing SQL Server DB instance using the console, the modify-db-instance CLI command, or the ModifyDBInstance RDS API operation.

  For instructions, see Modifying an Amazon RDS DB instance.
- Restore a SQL Server DB instance from a DB snapshot using the console, the restore-db-instance-from-db-snapshot CLI command, or the RestoreDBInstanceFromDBSnapshot RDS API operation.

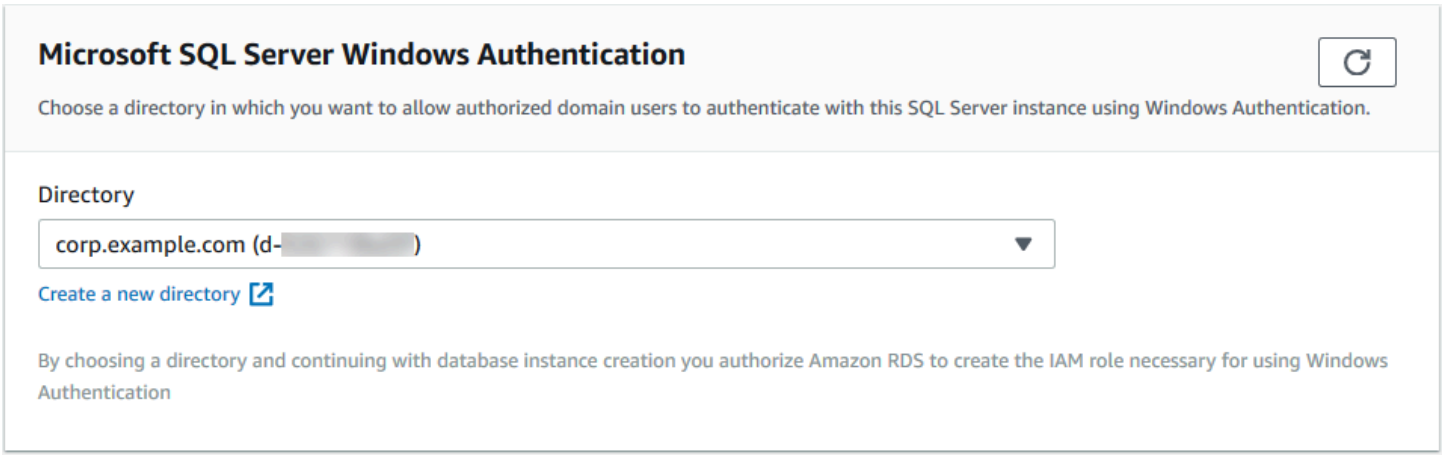
  For instructions, see Restoring to a DB instance.
- Restore a SQL Server DB instance to a point-in-time using the console, the restore-db-instance-to-point-in-time CLI command, or the RestoreDBInstanceToPointInTime RDS API operation.

  For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

Windows Authentication is only supported for SQL Server DB instances in a VPC.

For the DB instance to be able to use the domain directory that you created, the following is required:

- For **Directory**, you must choose the domain identifier (d-*ID*) generated when you created the directory.
- Make sure that the VPC security group has an outbound rule that lets the DB instance communicate with the directory.



When you use the AWS CLI, the following parameters are required for the DB instance to be able to use the directory that you created:

- For the `--domain` parameter, use the domain identifier (`d-`*`ID`*) generated when you created the directory.
- For the `--domain-iam-role-name` parameter, use the role that you created that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

For example, the following CLI command modifies a DB instance to use a directory.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --domain d-ID \
    --domain-iam-role-name role-name
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

Important

If you modify a DB instance to enable Kerberos authentication, reboot the DB instance after making the change.

Step 6: Create Windows Authentication SQL Server logins

Use the Amazon RDS master user credentials to connect to the SQL Server DB instance as you do any other DB instance. Because the DB instance is joined to the AWS Managed Microsoft AD domain, you can provision SQL Server logins and users. You do this from the Active Directory users and groups in your domain. Database permissions are managed through standard SQL Server permissions granted and revoked to these Windows logins.

For an Active Directory user to authenticate with SQL Server, a SQL Server Windows login must exist for the user or a group that the user is a member of. Fine-grained access control is handled through granting and revoking permissions on these SQL Server logins. A user that doesn't have a SQL Server login or belong to a group with such a login can't access the SQL Server DB instance.

The ALTER ANY LOGIN permission is required to create an Active Directory SQL Server login. If you haven't created any logins with this permission, connect as the DB instance's master user using SQL Server Authentication.

Run a data definition language (DDL) command such as the following example to create a SQL Server login for an Active Directory user or group.

Note

Specify users and groups using the pre-Windows 2000 login name in the format *domainName**login_name*. You can't use a user principal name (UPN) in the format *login_name*@*DomainName*.
You can only create a Windows Authentication login on an RDS for SQL Server instance by using T-SQL statements. You can't use the SQL Server Management studio to create a Windows Authentication login.

```
USE [master]
GO
```

```
CREATE LOGIN [mydomain\myuser] FROM WINDOWS WITH DEFAULT_DATABASE = [master],
 DEFAULT_LANGUAGE = [us_english];
GO
```

For more information, see CREATE LOGIN (Transact-SQL) in the Microsoft Developer Network documentation.

Users (both humans and applications) from your domain can now connect to the RDS for SQL Server instance from a domain-joined client machine using Windows authentication.

Managing a DB instance in a Domain

You can use the console, AWS CLI, or the Amazon RDS API to manage your DB instance and its relationship with your domain. For example, you can move the DB instance into, out of, or between domains.

For example, using the Amazon RDS API, you can do the following:

- To reattempt a domain join for a failed membership, use the ModifyDBInstance API operation and specify the current membership's directory ID.
- To update the IAM role name for membership, use the `ModifyDBInstance` API operation and specify the current membership's directory ID and the new IAM role.
- To remove a DB instance from a domain, use the `ModifyDBInstance` API operation and specify none as the domain parameter.
- To move a DB instance from one domain to another, use the `ModifyDBInstance` API operation and specify the domain identifier of the new domain as the domain parameter.
- To list membership for each DB instance, use the DescribeDBInstances API operation.

Understanding Domain membership

After you create or modify your DB instance, the instance becomes a member of the domain. The AWS console indicates the status of the domain membership for the DB instance. The status of the DB instance can be one of the following:

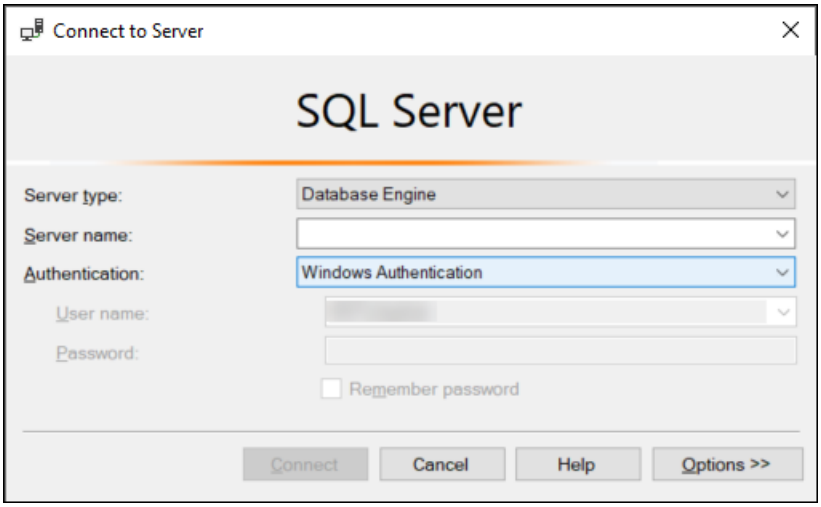
- **joined** – The instance is a member of the domain.
- **joining** – The instance is in the process of becoming a member of the domain.
- **pending-join** – The instance membership is pending.

- **pending-maintenance-join** – AWS will attempt to make the instance a member of the domain during the next scheduled maintenance window.
- **pending-removal** – The removal of the instance from the domain is pending.
- **pending-maintenance-removal** – AWS will attempt to remove the instance from the domain during the next scheduled maintenance window.
- **failed** – A configuration problem has prevented the instance from joining the domain. Check and fix your configuration before reissuing the instance modify command.
- **removing** – The instance is being removed from the domain.

A request to become a member of a domain can fail because of a network connectivity issue or an incorrect IAM role. For example, you might create a DB instance or modify an existing instance and have the attempt fail for the DB instance to become a member of a domain. In this case, either reissue the command to create or modify the DB instance or modify the newly created instance to join the domain.

Connecting to SQL Server with Windows authentication

To connect to SQL Server with Windows Authentication, you must be logged into a domain-joined computer as a domain user. After launching SQL Server Management Studio, choose **Windows Authentication** as the authentication type, as shown following.



Restoring a SQL Server DB instance and then adding it to a domain

You can restore a DB snapshot or do point-in-time recovery (PITR) for a SQL Server DB instance and then add it to a domain. Once the DB instance is restored, modify the instance using the process explained in Step 5: Create or modify a SQL Server DB instance to add the DB instance to a domain.

Upgrades of the Microsoft SQL Server DB engine

When Amazon RDS supports a new version of a database engine, you can upgrade your DB instances to the new version. There are two kinds of upgrades for SQL Server DB instances: major version upgrades and minor version upgrades.

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, you must *manually* perform major version upgrades of your DB instances. You can initiate a major version upgrade by modifying your DB instance. However, before you perform a major version upgrade, we recommend that you test the upgrade by following the steps described in Testing an RDS for SQL Server upgrade.

Minor version upgrades contain only changes that are backward-compatible with existing applications. You can upgrade the minor version for your DB instance in two ways:

- *Manually* – Modify your DB instance to initiate the upgrade
- *Automatically* – Enable automatic minor version upgrades for your DB instance

When you enable automatic minor version upgrades, RDS for SQL Server automatically upgrades your database instance during scheduled maintenance windows when critical security updates are available in a newer minor version.

For minor engine versions after `16.00.4120.1`, `15.00.4365.2`, `14.00.3465.1`, `13.00.6435.1`, the following security protocols are disabled by default:

- `rds.tls10` (TLS 1.0 protocol)
- `rds.tls11` (TLS 1.1 protocol)
- `rds.rc4` (RC4 cipher)
- `rds.curve25519` (Curve25519 encryption)
- `rds.3des168` (Triple DES encryption)

For earlier engine versions, Amazon RDS enables these security protocols by default.

```
...

"ValidUpgradeTarget": [
    {
```

```
        "Engine": "sqlserver-se",
        "EngineVersion": "14.00.3281.6.v1",
        "Description": "SQL Server 2017 14.00.3281.6.v1",
        "AutoUpgrade": false,
        "IsMajorVersionUpgrade": false
    }
...
```

For more information about performing upgrades, see Upgrading a SQL Server DB instance. For information about what SQL Server versions are available on Amazon RDS, see Amazon RDS for Microsoft SQL Server.

Amazon RDS also supports upgrade rollout policy to manage automatic minor version upgrades across multiple database resources and AWS accounts. For more information, see Using AWS Organizations upgrade rollout policy for automatic minor version upgrades.

Topics

- Major version upgrades for RDS for SQL Server
- Considerations for SQL Server upgrades
- Testing an RDS for SQL Server upgrade
- Upgrading a SQL Server DB instance
- Upgrading deprecated DB instances before support ends

Major version upgrades for RDS for SQL Server

Amazon RDS currently supports the following major version upgrades to a Microsoft SQL Server DB instance.

You can upgrade your existing DB instance to SQL Server 2017 or 2019 from any version except SQL Server 2008. To upgrade from SQL Server 2008, first upgrade to one of the other versions.

Current version	Supported upgrade versions
SQL Server 2019	SQL Server 2022
SQL Server 2017	SQL Server 2022 SQL Server 2019

Current version	Supported upgrade versions
SQL Server 2016	SQL Server 2022
	SQL Server 2019
	SQL Server 2017

You can use an AWS CLI query, such as the following example, to find the available upgrades for a particular database engine version.

Example

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
    --engine sqlserver-se \
    --engine-version 14.00.3281.6.v1 \
    --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" \
    --output table
```

For Windows:

```
aws rds describe-db-engine-versions ^
    --engine sqlserver-se ^
    --engine-version 14.00.3281.6.v1 ^
    --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
    --output table
```

The output shows that you can upgrade version 14.00.3281.6 to the latest available SQL Server 2017 or 2019 versions.

```
-------------------------
|DescribeDBEngineVersions|
+------------------------+
|      EngineVersion     |
+------------------------+
|  14.00.3294.2.v1       |
|  14.00.3356.20.v1      |
|  14.00.3381.3.v1       |
```

```
|  14.00.3401.7.v1       |
|  14.00.3421.10.v1      |
|  14.00.3451.2.v1       |
|  15.00.4043.16.v1      |
|  15.00.4073.23.v1      |
|  15.00.4153.1.v1       |
|  15.00.4198.2.v1       |
|  15.00.4236.7.v1       |
+------------------------+
```

Database compatibility level

You can use Microsoft SQL Server database compatibility levels to adjust some database behaviors to mimic previous versions of SQL Server. For more information, see Compatibility level in the Microsoft documentation. When you upgrade your DB instance, all existing databases remain at their original compatibility level.

You can change the compatibility level of a database by using the ALTER DATABASE command. For example, to change a database named `customeracct` to be compatible with SQL Server 2016, issue the following command:

```
ALTER DATABASE customeracct SET COMPATIBILITY_LEVEL = 130
```

Considerations for SQL Server upgrades

Amazon RDS takes two DB snapshots during the upgrade process. The first DB snapshot is of the DB instance before any upgrade changes have been made. The second DB snapshot is taken after the upgrade finishes.

> **Note**
>
> Amazon RDS only takes DB snapshots if you have set the backup retention period for your DB instance to a number greater than 0. To change your backup retention period, see Modifying an Amazon RDS DB instance.

After an upgrade is completed, you can't revert to the previous version of the database engine. If you want to return to the previous version, restore from the DB snapshot that was taken before the upgrade to create a new DB instance.

During a minor or major version upgrade of SQL Server, the **Free Storage Space** and **Disk Queue Depth** metrics will display -1. After the upgrade is completed, both metrics will return to normal.

Before you upgrade your SQL Server instance, review the following information.

Topics

- Best practices before initiating an upgrade
- Multi-AZ considerations
- Read replica considerations
- Option group considerations
- Parameter group considerations

Best practices before initiating an upgrade

Before starting the upgrade process, implement the following preparatory stpes to allow optimal upgrade performance and minimize potential issues:

Timing and workload management

- Schedule upgrades during low transaction volume periods.
- Minimize write operations during the upgrade window.

  This allows Amazon RDS to complete upgrades faster by reducing the number of transaction log backup files that RDS needs to restore during secondary-to-primary pairing.

Transaction management

- Identify and monitor long-running transactions.
- Ensure all critical transactions are commited before starting the upgrade.
- Prevent long-running transactions during the upgrade window.

Log file optimization

Review and optimize transaction log files:

- Shrink oversized log files.
- Reduce high log consumption patterns.
- Manage Virtual Log Files (VLFs).

- Maintain adequate free space for normal operations.

Multi-AZ considerations

Amazon RDS supports Multi-AZ deployments for DB instances running Microsoft SQL Server by using SQL Server Database Mirroring (DBM) or Always On Availability Groups (AGs). For more information, see Multi-AZ deployments for Amazon RDS for Microsoft SQL Server.

In a Multi-AZ deployment (Mirroring/AlwaysOn), when an upgrade is requested, RDS follows a rolling upgrade strategy for the primary and secondary instances. Rolling upgrades ensure at least one instance is available for transactions while the secondary instance is upgraded. The outage is expected to only last the duration of a failover.

During the upgrade, RDS removes the secondary instance from the Multi-AZ configuration, performs an upgrade of the secondary instance, and restores any transaction log backups from the primary taken during the time it was disconnected. After all the log backups are restored, RDS joins the upgraded secondary to the primary. When all the databases are in a synchronized state, RDS performs a failover to the upgraded secondary instance. Once the failover is completed, RDS proceeds with upgrading the old primary instance, restores any transaction log backups, and pairs it with the new primary.

To minimize this failover duration, we recommend using AlwaysOn AGs availability group listener endpoint when using client libraries that support the `MultiSubnetFailover` connection option in the connection string. When using the availability group listener endpoint, failover times are typically less than 10 seconds, however, this duration does not include any additional crash recovery time.

Read replica considerations

During a database version upgrade, Amazon RDS upgrades all of your read replicas along with the primary DB instance. Amazon RDS does not support database version upgrades on the read replicas separately. For more information on read replicas, see Working with read replicas for Microsoft SQL Server in Amazon RDS.

When you perform a database version upgrade of the primary DB instance, all its read-replicas are also automatically upgraded. Amazon RDS will upgrade all of the read replicas simultaneously before upgrading the primary DB instance. Read replicas may not be available until the database version upgrade on the primary DB instance is complete.

Option group considerations

If your DB instance uses a custom DB option group, in some cases Amazon RDS can't automatically assign your DB instance a new option group. For example, when you upgrade to a new major version, you must specify a new option group. We recommend that you create a new option group, and add the same options to it as your existing custom option group.

For more information, see Creating an option group or Copying an option group.

Parameter group considerations

If your DB instance uses a custom DB parameter group:

- Amazon RDS automatically reboots the DB instance after an upgrade.
- In some cases, RDS can't automatically assign a new parameter group to your DB instance.

  For example, when you upgrade to a new major version, you must specify a new parameter group. We recommend that you create a new parameter group, and configure the parameters as in your existing custom parameter group.

For more information, see Creating a DB parameter group in Amazon RDS or Copying a DB parameter group in Amazon RDS.

Testing an RDS for SQL Server upgrade

Before you perform a major version upgrade on your DB instance, you should thoroughly test your database, and all applications that access the database, for compatibility with the new version. We recommend that you use the following procedure.

To test a major version upgrade

1. Review Upgrade SQL Server in the Microsoft documentation for the new version of the database engine to see if there are compatibility issues that might affect your database or applications.
2. If your DB instance uses a custom option group, create a new option group compatible with the new version you are upgrading to. For more information, see Option group considerations.
3. If your DB instance uses a custom parameter group, create a new parameter group compatible with the new version you are upgrading to. For more information, see Parameter group considerations.

4. Create a DB snapshot of the DB instance to be upgraded. For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.
5. Restore the DB snapshot to create a new test DB instance. For more information, see Restoring to a DB instance.
6. Modify this new test DB instance to upgrade it to the new version, by using one of the following methods:
   - Console
   - AWS CLI
   - RDS API
7. Evaluate the storage used by the upgraded instance to determine if the upgrade requires additional storage.
8. Run as many of your quality assurance tests against the upgraded DB instance as needed to ensure that your database and application work correctly with the new version. Implement any new tests needed to evaluate the impact of any compatibility issues you identified in step 1. Test all stored procedures and functions. Direct test versions of your applications to the upgraded DB instance.
9. If all tests pass, then perform the upgrade on your production DB instance. We recommend that you do not allow write operations to the DB instance until you confirm that everything is working correctly.

Upgrading a SQL Server DB instance

For information about manually or automatically upgrading a SQL Server DB instance, see the following:

- Upgrading a DB instance engine version
- Best practices for upgrading SQL Server 2008 R2 to SQL Server 2016 on Amazon RDS for SQL Server

> **Important**
>
> If you have any snapshots that are encrypted using AWS KMS, we recommend that you initiate an upgrade before support ends.

Upgrading deprecated DB instances before support ends

After a major version is deprecated, you can't install it on new DB instances. RDS will try to automatically upgrade all existing DB instances.

If you need to restore a deprecated DB instance, you can do point-in-time recovery (PITR) or restore a snapshot. Doing this gives you temporary access a DB instance that uses the version that is being deprecated. However, after a major version is fully deprecated, these DB instances will also be automatically upgraded to a supported version.

Working with storage in RDS for SQL Server

With RDS for SQL Server, you can attach up to three additional volumes to your RDS for SQL Server instance, each mapped to a unique Windows drive letter. This allows you to distribute database files across multiple volumes beyond the default `D:` drive. When you add a storage volume, you get enhanced flexibility for database file management and storage optimization.

Benefits include:

- **Flexible file distribution** – Distribute database data files and log files across multiple volumes for improved I/O performance.
- **Storage optimization** – Use different storage types and configurations for different workload requirements.
- **Scalability** – Add storage capacity without modifying existing volumes.

On this page

- Considerations for using additional storage volumes with RDS for SQL Server
- Add, remove, or modify storage volumes with RDS for SQL Server
- Restore operations for additional storage volumes with RDS for SQL Server
- Use cases for additional storage volumes with RDS for SQL Server

Considerations for using additional storage volumes with RDS for SQL Server

Take note of the following features and limitations when using additional storage volumes with RDS for SQL Server:

- You can only add storage volumes on SQL Server Standard Edition (SE), Enterprise Edition (EE), and Developer Edition (DEV-EE).
- You can add up to 3 additional storage volumes per instance.
- Volume names are automatically mapped to Windows drive letters as follows:
  - `rdsdbdata2` – `H:` drive
  - `rdsdbdata3` – `I:` drive
  - `rdsdbdata4` – `J:` drive

- TempDB files continue to use the `T:` drive when using NVMe instance storage. SQL Server Audit files and Microsoft Business Intelligence (MSBI) files remain on the `D:` drive.
- You can only add General Purpose SSD (gp3) and Provisioned IOPS SSD (io2) storage types.
- The minimum storage size of additional storage volumes is the same as the limit set for the default `D:` drive. The maximum storage size for your DB instance is 256 TiB total across all volumes.
- Adding storage volumes to instances with read replicas or to read replica instances isn't supported.
- Adding storage volumes to instances enabled for cross-region automated backup isn't supported.
- Configuring additional storage volumes for storage autoscaling isn't supported.
- Moving files between volumes after creation isn't supported.
- You can't delete the `D:` volume, but you can delete other storage volumes as long as they're empty.
- Modifying the size of existing volumes during snapshot restore or point-in-time recovery (PITR) isn't supported. However, you can add new storage volumes during restore operations.

Add, remove, or modify storage volumes with RDS for SQL Server

You can add, modify, and remove additional storage volumes using the AWS CLI or AWS Management Console. All operations use the `modify-db-instance` API operation with the `additional-storage-volumes` parameter.

> **Important**
>
> Adding or removing additional storage volumes creates a backup pending action and a point-in-time restore blackout window. This window closes when the backup workflow completes.

Topics

- Adding storage volumes
- Scaling additional storage volumes
- Removing additional storage volumes

Adding storage volumes

You can add up to three storage volumes beyond the default `D:` drive. To add a new storage volume to your RDS for SQL Server instance, use the `modify-db-instance` command with the `additional-storage-volumes` parameter.

The following example adds a new 4,000 GiB General Purpose SSD (gp3) volume named `rdsdbdata4`.

```
aws rds modify-db-instance \
  --db-instance-identifier my-sql-server-instance \
  --region us-east-1 \
  --additional-storage-volumes
 '[{"VolumeName":"rdsdbdata4","StorageType":"gp3","AllocatedStorage":4000}]' \
  --apply-immediately
```

Scaling additional storage volumes

You can modify any storage setting for your additional volumes except for storage size. The following example modifies the IOPS setting for the `rdsdbdata2` volume.

```
aws rds modify-db-instance \
  --db-instance-identifier my-sql-server-instance \
  --region us-east-1 \
  --additional-storage-volumes '[{"VolumeName":"rdsdbdata2","IOPS":4000}]' \
  --apply-immediately
```

Removing additional storage volumes

You can't delete the `D:` volume, but you can delete other storage volumes when they're empty.

> **Warning**
>
> Before you remove an additional storage volume, make sure that no database files are stored on the volume.

The following example removes the `rdsdbdata4` volume.

```
aws rds modify-db-instance \
  --db-instance-identifier my-sql-server-instance \
```

```
  --region us-east-1 \
  --additional-storage-volumes '[{"VolumeName":"rdsdbdata4","SetForDelete":true}]' \
  --apply-immediately
```

Restore operations for additional storage volumes with RDS for SQL Server

When you restore your database, you can add storage volumes. You can also modify the storage settings of existing volumes.

Topics

- Snapshot restore
- Point-in-time recovery
- Native database restore

Snapshot restore

When restoring from a snapshot, you can add new additional storage volumes or modify the IOPS, throughput, and storage type settings of existing volumes.

The following example restores a DB instance from a snapshot and modifies the IOPS setting for the `rdsdbdata2` volume:

```
aws rds restore-db-instance-from-db-snapshot \
  --db-instance-identifier my-restored-instance \
  --db-snapshot-identifier my-snapshot \
  --region us-east-1 \
  --additional-storage-volumes '[{"VolumeName":"rdsdbdata2","IOPS":5000}]'
```

Point-in-time recovery

During point-in-time recovery (PITR), you can add new additional storage volumes with custom configurations.

The following example performs PITR and adds a new 5,000 GiB General Purpose SSD (gp3) volume:

```
aws rds restore-db-instance-to-point-in-time \
  --source-db-instance-identifier my-source-instance \
```

```
  --target-db-instance my-pitr-instance \
  --use-latest-restorable-time \
  --region us-east-1 \
  --additional-storage-volumes
'[{"VolumeName":"rdsdbdata4","StorageType":"gp3","AllocatedStorage":5000,"IOPS":5000,"StorageT
```

Native database restore

You can use the `rds_restore_database` stored procedure to restore databases to specific additional storage volumes. Two new parameters support volume selection:

`data_file_volume`

Specifies the drive letter for database data files

`log_file_volume`

Specifies the drive letter for database log files

The following example restores a database with data files on the `H:` drive and log files on the `I:` drive:

```
EXEC msdb.dbo.rds_restore_database
    @restore_db_name='my_database',
    @s3_arn_to_restore_from='arn:aws:s3:::my-bucket/backup-file.bak',
    @data_file_volume='H:',
    @log_file_volume='I:';
```

If you don't specify volume parameters, or if you specify the `D:` drive for both parameters, the database files are restored to the default `D:` drive:

```
EXEC msdb.dbo.rds_restore_database
    @restore_db_name='my_database',
    @s3_arn_to_restore_from='arn:aws:s3:::my-bucket/backup-file.bak';
```

Use cases for additional storage volumes with RDS for SQL Server

Additional storage volumes support various database management scenarios. The following sections describe common use cases and implementation approaches.

Topics

- Creating databases on additional storage volumes
- Extending storage capacity
- Moving databases between volumes
- Archiving data to cost-effective storage

Creating databases on additional storage volumes

You can create new databases directly on additional storage volumes using standard SQL Server `CREATE DATABASE` statements.

The following example creates a database with data files on the `H:` drive and log files on the `I:` drive:

```
CREATE DATABASE MyDatabase
ON (
    NAME = 'MyDatabase_Data',
    FILENAME = 'H:\rdsdbdata\data\MyDatabase_Data.mdf',
    SIZE = 100MB,
    FILEGROWTH = 10MB
)
LOG ON (
    NAME = 'MyDatabase_Log',
    FILENAME = 'I:\rdsdbdata\data\MyDatabase_Log.ldf',
    SIZE = 10MB,
    FILEGROWTH = 10%
);
```

Extending storage capacity

When the default `D:` drive reaches its maximum capacity, you can add additional storage volumes, scale existing volumes, and create new data files or log files on the new volumes.

To extend storage capacity

1. Add a storage volume to your instance using the `modify-db-instance` command.
2. Add a new data file to the additional storage volume:

```
ALTER DATABASE MyDatabase
ADD FILE (
```

```
    NAME = 'MyDatabase_Data2',
    FILENAME = 'H:\rdsdbdata\data\MyDatabase_Data2.ndf',
    SIZE = 500MB,
    FILEGROWTH = 50MB
);
```

Moving databases between volumes

To move a database to a different volume, use the backup and restore approach with the `rds_backup_database` and `rds_restore_database` stored procedures. For more information, see Using native backup and restore.

To move a database to a different volume

1. Back up the database using `rds_backup_database`:

```
EXEC msdb.dbo.rds_backup_database
    @source_db_name='MyDatabase',
    @s3_arn_to_backup_to='arn:aws:s3:::my-bucket/database-backup.bak';
```

2. Restore the database to the target volume:

```
EXEC msdb.dbo.rds_restore_database
    @restore_db_name='MyDatabase_New',
    @s3_arn_to_restore_from='arn:aws:s3:::my-bucket/database-backup.bak',
    @data_file_volume='H:',
    @log_file_volume='I:';
```

3. Drop the database from your old drive to release the space. For more information, see Dropping a database in an Amazon RDS for Microsoft SQL Server DB instance.

Archiving data to cost-effective storage

For partitioned tables, you can archive older data to additional storage volumes with different performance characteristics.

To archive partitioned data

1. Add a storage volume with appropriate storage type and capacity.
2. Create a new filegroup on the additional storage volume:

```
ALTER DATABASE MyDatabase
ADD FILEGROUP ArchiveFileGroup;

ALTER DATABASE MyDatabase
ADD FILE (
    NAME = 'Archive_Data',
    FILENAME = 'H:\rdsdbdata\data\Archive_Data.ndf',
    SIZE = 1GB,
    FILEGROWTH = 100MB
) TO FILEGROUP ArchiveFileGroup;
```

3. Move partitions to the new filegroup using SQL Server partition management commands.

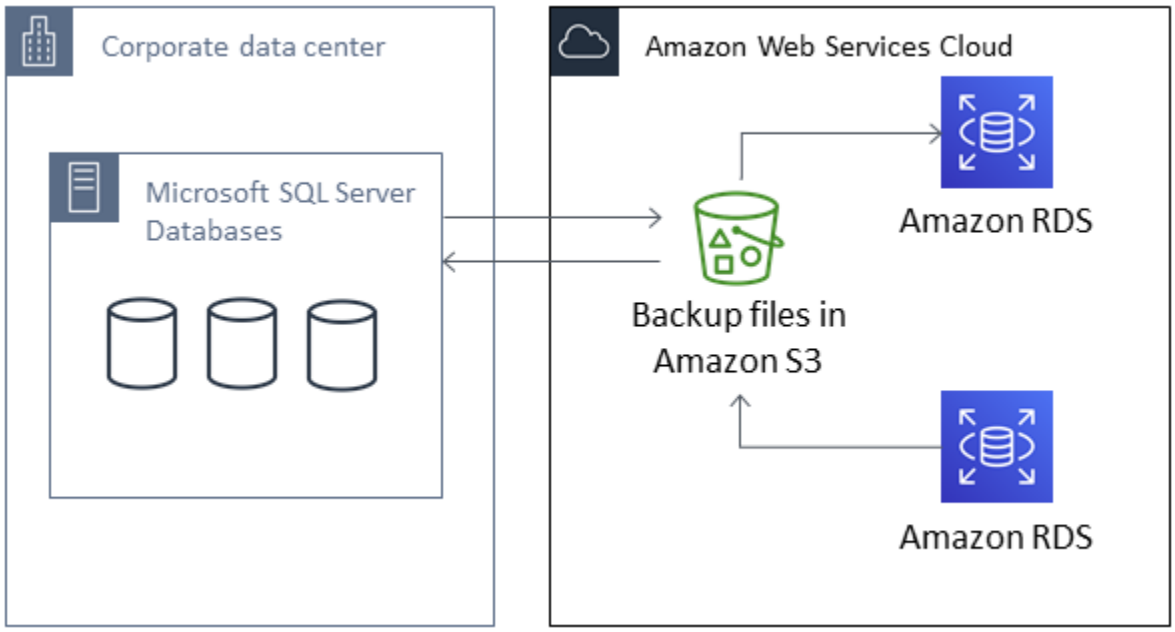
Importing and exporting SQL Server databases using native backup and restore

Amazon RDS supports native backup and restore for Microsoft SQL Server databases using full backup files (.bak files). When you use RDS, you access files stored in Amazon S3 rather than using the local file system on the database server.

For example, you can create a full backup from your local server, store it on S3, and then restore it onto an existing Amazon RDS DB instance. You can also make backups from RDS, store them on S3, and then restore them wherever you want.

Native backup and restore is available in all AWS Regions for Single-AZ and Multi-AZ DB instances, including Multi-AZ DB instances with read replicas. Native backup and restore is available for all editions of Microsoft SQL Server supported on Amazon RDS.

The following diagram shows the supported scenarios.



Using native .bak files to back up and restore databases is usually the fastest way to back up and restore databases. There are many additional advantages to using native backup and restore. For example, you can do the following:

- Migrate databases to or from Amazon RDS.
- Move databases between RDS for SQL Server DB instances.
- Migrate data, schemas, stored procedures, triggers, and other database code inside .bak files.

- Backup and restore single databases, instead of entire DB instances.
- Create copies of databases for development, testing, training, and demonstrations.
- Store and transfer backup files with Amazon S3, for an added layer of protection for disaster recovery.
- Create native backups of databases that have Transparent Data Encryption (TDE) turned on, and restore those backups to on-premises databases. For more information, see Support for Transparent Data Encryption in SQL Server.
- Restore native backups of on-premises databases that have TDE turned on to RDS for SQL Server DB instances. For more information, see Support for Transparent Data Encryption in SQL Server.

Contents

- Limitations and recommendations
- Setting up for native backup and restore
  - Manually creating an IAM role for native backup and restore
- Using native backup and restore
  - Backing up a database
    - Usage
    - Examples
  - Restoring a database
    - Usage
    - Examples
  - Restoring a log
    - Usage
    - Examples
  - Finishing a database restore
    - Usage
  - Working with partially restored databases
    - Dropping a partially restored database
    - Snapshot restore and point-in-time recovery behavior for partially restored databases
  - Canceling a task
    - Usage

- Tracking the status of tasks
  - Usage
  - Examples
  - Response
- Compressing backup files
- Troubleshooting
- Importing and exporting SQL Server data using other methods
  - Importing data into RDS for SQL Server by using a snapshot
    - Import the data
      - Generate and Publish Scripts Wizard
      - Import and Export Wizard
      - Bulk copy
  - Exporting data from RDS for SQL Server
    - SQL Server Import and Export Wizard
    - SQL Server Generate and Publish Scripts Wizard and bcp utility
- Using BCP utility from Linux to import and export data
  - Prerequisites
  - Installing SQL Server command-line tools on Linux
  - Exporting data from RDS for SQL Server
    - Basic export syntax
    - Export example
  - Importing data to RDS for SQL Server
    - Basic import syntax
    - Import example
  - Common BCP options
  - Best practices and considerations
  - Troubleshooting common issues

Limitations and recommendations

The following are some limitations to using native backup and restore:

- You can't back up to, or restore from, an Amazon S3 bucket in a different AWS Region from your Amazon RDS DB instance.
- You can't restore a database with the same name as an existing database. Database names are unique.
- We strongly recommend that you don't restore backups from one time zone to a different time zone. If you restore backups from one time zone to a different time zone, you must audit your queries and applications for the effects of the time zone change.
- RDS for Microsoft SQL Server has a size limit of 5 TB per file. For native backups of larger databases, you can use multifile backup.
- The maximum database size that can be backed up to S3 depends on the available memory, CPU, I/O, and network resources on the DB instance. The larger the database, the more memory the backup agent consumes.
- You can't back up to or restore from more than 10 backup files at the same time.
- A differential backup is based on the last full backup. For differential backups to work, you can't take a snapshot between the last full backup and the differential backup. If you want a differential backup, but a manual or automated snapshot exists, then do another full backup before proceeding with the differential backup.
- Differential and log restores aren't supported for databases with files that have their file_guid (unique identifier) set to `NULL`.
- You can run up to two backup or restore tasks at the same time.
- You can't perform native log backups from SQL Server on Amazon RDS.
- RDS supports native restores of databases up to 64 TiB. Native restores of databases on SQL Server Express Edition are limited to 10 GB.
- You can't do a native backup during the maintenance window, or any time Amazon RDS is in the process of taking a snapshot of the database. If a native backup task overlaps with the RDS daily backup window, the native backup task is canceled.
- On Multi-AZ DB instances, you can only natively restore databases that are backed up in the full recovery model.
- Calling the RDS procedures for native backup and restore within a transaction isn't supported.
- Use a symmetric encryption AWS KMS key to encrypt your backups. Amazon RDS doesn't support asymmetric KMS keys. For more information, see Creating symmetric encryption KMS keys in the *AWS Key Management Service Developer Guide*.

- Native backup files are encrypted with the specified KMS key using the "Encryption-Only" crypto mode. When you are restoring encrypted backup files, be aware that they were encrypted with the "Encryption-Only" crypto mode.
- You can't restore a database that contains a FILESTREAM file group.
- Amazon S3 server-side encryption with AWS KMS (SSE-KMS) is supported through your S3 bucket's default encryption configuration when you pass `@enable_bucket_default_encryption=1` to the backup stored procedure. By default, the restore supports the S3 object's server-side encryption.

  When you provide a KMS key to a stored procedure, any native backup and restores are encrypted and decrypted on the client-side with the KMS key. AWS stores the backups in the S3 bucket with SSE-S3 when `@enable_bucket_default_encryption=0` or with your S3 bucket's configured default encryption key when `@enable_bucket_default_encryption=1`.
- When using S3 Access Points, the access point cannot be configured to use an RDS internal VPC.
- For highest performance, we recommend using directory buckets or access points for directory buckets if they are available in your region.

If your database can be offline while the backup file is created, copied, and restored, we recommend that you use native backup and restore to migrate it to RDS. If your on-premises database can't be offline, we recommend that you use the AWS Database Migration Service to migrate your database to Amazon RDS. For more information, see What is AWS Database Migration Service?

Native backup and restore isn't intended to replace the data recovery capabilities of the cross-region snapshot copy feature. We recommend that you use snapshot copy to copy your database snapshot to another AWS Region for cross-region disaster recovery in Amazon RDS. For more information, see Copying a DB snapshot for Amazon RDS.

Setting up for native backup and restore

To set up for native backup and restore, you need three components:

1. An Amazon S3 bucket to store your backup files.

   You must have an S3 bucket to use for your backup files and then upload backups you want to migrate to RDS. If you already have an Amazon S3 bucket, you can use that. If you don't, you can

create a bucket. Alternatively, you can choose to have a new bucket created for you when you add the SQLSERVER_BACKUP_RESTORE option by using the AWS Management Console.

For information on using S3, see the Amazon Simple Storage Service User Guide

2. An AWS Identity and Access Management (IAM) role to access the bucket.

   If you already have an IAM role, you can use that. You can choose to have a new IAM role created for you when you add the SQLSERVER_BACKUP_RESTORE option by using the AWS Management Console. Alternatively, you can create a new one manually.

   If you want to create a new IAM role manually, take the approach discussed in the next section. Do the same if you want to attach trust relationships and permissions policies to an existing IAM role.

3. The SQLSERVER_BACKUP_RESTORE option added to an option group on your DB instance.

   To enable native backup and restore on your DB instance, you add the SQLSERVER_BACKUP_RESTORE option to an option group on your DB instance. For more information and instructions, see Support for native backup and restore in SQL Server.

Manually creating an IAM role for native backup and restore

If you want to manually create a new IAM role to use with native backup and restore, you can do so. In this case, you create a role to delegate permissions from the Amazon RDS service to your Amazon S3 bucket. When you create an IAM role, you attach a trust relationship and a permissions policy. The trust relationship allows RDS to assume this role. The permissions policy defines the actions this role can perform. For more information about creating the role, see Creating a role to delegate permissions to an AWS service.

For the native backup and restore feature, use trust relationships and permissions policies similar to the examples in this section. In the following example, we use the service principal name `rds.amazonaws.com` as an alias for all service accounts. In the other examples, we specify an Amazon Resource Name (ARN) to identify another account, user, or role that we're granting access to in the trust policy.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based trust relationships to limit the service's permissions to a specific resource. This is the most effective way to protect against the confused deputy problem.

You might use both global condition context keys and have the `aws:SourceArn` value contain the account ID. In this case, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same statement.

- Use `aws:SourceArn` if you want cross-service access for a single resource.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

In the trust relationship, make sure to use the `aws:SourceArn` global condition context key with the full ARN of the resources accessing the role. For native backup and restore, make sure to include both the DB option group and the DB instances, as shown in the following example.

Example of trust relationship with global condition context key for native backup and restore

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": [

 "arn:aws:rds:Region:0123456789:db:db_instance_identifier",
                        "arn:aws:rds:Region:0123456789:og:option_group_name"
                    ],
                    "aws:SourceAccount": "0123456789"
                }
            }
        }
    ]
}
```

The following example uses an ARN to specify a resource. For more information on using ARNs, see Amazon resource names (ARNs).

Example of permissions policy for native backup and restore without encryption support

JSON

```
{
    "Version":"2012-10-17",
    "Statement":
    [
        {
        "Effect": "Allow",
        "Action":
            [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        {
        "Effect": "Allow",
        "Action":
            [
                "s3:GetObjectAttributes",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Example permissions policy for native backup and restore with encryption support

If you want to encrypt your backup files, include an encryption key in your permissions policy. For more information about encryption keys, see Getting started in the *AWS Key Management Service Developer Guide*.

Note

You must use a symmetric encryption KMS key to encrypt your backups. Amazon RDS doesn't support asymmetric KMS keys. For more information, see Creating symmetric encryption KMS keys in the *AWS Key Management Service Developer Guide*.
The IAM role must also be a key user and key administrator for the KMS key, that is, it must be specified in the key policy. For more information, see Creating symmetric encryption KMS keys in the *AWS Key Management Service Developer Guide*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToKey",
      "Effect": "Allow",
      "Action": [
        "kms:DescribeKey",
        "kms:GenerateDataKey",
        "kms:Encrypt",
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/key-id"
    },
    {
      "Sid": "AllowAccessToS3",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
    },
    {
      "Sid": "GetS3Info",
      "Effect": "Allow",
      "Action": [
        "s3:GetObjectAttributes",
```

```
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

Example permissions policy for native backup and restore using access points without encryption support

The actions required to use S3 access points are the same as for S3 buckets. The resource path is updated to match the S3 access point ARN pattern.

> **Note**
>
> Access points must be configured to use **Network origin: Internet** as RDS does not publish private VPCs. S3 traffic from RDS instances does not go through the public internet since it goes through private VPCs.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
                ],
            "Resource": [
            "arn:aws:s3:us-east-1:111122223333:accesspoint/amzn-s3-demo-ap",
            "arn:aws:s3:::underlying-bucket"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectAttributes",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
                ],
                "Resource": [
                "arn:aws:s3:us-east-1:111122223333:accesspoint/amzn-s3-demo-ap/
*",
                    "arn:aws:s3:::underlying-bucket/*"
                    ]
                }
            ]
}
```

Example permissions policy for native backup and restore using access points for directory buckets without encryption support

Directory buckets use a different, session-based authorization mechanism than general purpose buckets, so the only required permission for native backup restore is the bucket-level “s3express:CreateSession” permission. To configure object-level access, you must use access points for directory buckets.

> **Note**
>
> Access points must be configured to use **Network origin: Internet** as RDS does not publish private VPCs. S3 traffic from RDS instances does not go through the public internet since it goes through private VPCs.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":
    [
        {
```

```
        "Effect": "Allow",
        "Action": "s3express:CreateSession",
        "Resource":
            [
                "arn:aws:s3express:us-east-1:111122223333:accesspoint/amzn-s3-
demo-accesspoint--use1-az6--xa-s3",
                "arn:aws:s3express:us-east-1:111122223333:bucket/amzn-s3-demo-
bucket--use1-az6--x-s3"
            ]
        }
    ]
}
```

Using native backup and restore

After you have enabled and configured native backup and restore, you can start using it. First, you connect to your Microsoft SQL Server database, and then you call an Amazon RDS stored procedure to do the work. For instructions on connecting to your database, see Connecting to your Microsoft SQL Server DB instance.

Some of the stored procedures require that you provide an Amazon Resource Name (ARN) to your Amazon S3 bucket and file. The format for your ARN is `arn:aws:s3:::bucket_name/file_name.extension`. Amazon S3 doesn't require an account number or AWS Region in ARNs.

If you also provide an optional KMS key, the format for the ARN of the key is `arn:aws:kms:region:account-id:key/key-id`. For more information, see Amazon resource names (ARNs) and AWS service namespaces. You must use a symmetric encryption KMS key to encrypt your backups. Amazon RDS doesn't support asymmetric KMS keys. For more information, see Creating symmetric encryption KMS keys in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> Whether or not you use a KMS key, the native backup and restore tasks enable server-side Advanced Encryption Standard (AES) 256-bit encryption through SSE-S3 by default for files uploaded to S3. Passing in `@enable_bucket_default_encryption=1` to the backup stored procedure uses your S3 bucket's configured default encryption key.

For instructions on how to call each stored procedure, see the following topics:

- Backing up a database
- Restoring a database
- Restoring a log
- Finishing a database restore
- Working with partially restored databases
- Canceling a task
- Tracking the status of tasks

Backing up a database

To back up your database, use the `rds_backup_database` stored procedure.

> **Note**
>
> You can't back up a database during the maintenance window, or while Amazon RDS is taking a snapshot.

Usage

```
exec msdb.dbo.rds_backup_database
 @source_db_name='database_name',
 @s3_arn_to_backup_to='arn:aws:s3:::bucket_name/file_name.extension',
 [@kms_master_key_arn='arn:aws:kms:region:account-id:key/key-id'],
 [@overwrite_s3_backup_file=0|1],
 [@block_size=512|1024|2048|4096|8192|16384|32768|65536],
        [@max_transfer_size=n],
        [@buffer_count=n],
 [@type='DIFFERENTIAL|FULL'],
 [@number_of_files=n],
 [@enable_bucket_default_encryption=0|1];
```

The following parameters are required:

- `@source_db_name` – The name of the database to back up.
- `@s3_arn_to_backup_to` – The ARN indicating the Amazon S3 bucket, access point, directory bucket, or access point for directory bucket to use for the backup, plus the name of the backup file.

  The file can have any extension, but `.bak` is usually used. Note that access point ARNs must be of the format `arn:aws:s3:us-east-1:111122223333:access-point-name/object/key`.

The following parameters are optional:

- `@kms_master_key_arn` – The ARN for the symmetric encryption KMS key to use to encrypt the item.
  - You can't use the default encryption key. If you use the default key, the database won't be backed up.

  - If you don't specify a KMS key identifier, the backup file won't be encrypted. For more information, see Encrypting Amazon RDS resources.
  - When you specify a KMS key, client-side encryption is used.
  - Amazon RDS doesn't support asymmetric KMS keys. For more information, see Creating symmetric encryption KMS keys in the *AWS Key Management Service Developer Guide*.
- `@overwrite_s3_backup_file` – A value that indicates whether to overwrite an existing backup file.
  - `0` – Doesn't overwrite an existing file. This value is the default.

    Setting `@overwrite_s3_backup_file` to 0 returns an error if the file already exists.
  - `1` – Overwrites an existing file that has the specified name, even if it isn't a backup file.
- `@type` – The type of backup.
  - `DIFFERENTIAL` – Makes a differential backup.
  - `FULL` – Makes a full backup. This value is the default.

  A differential backup is based on the last full backup. For differential backups to work, you can't take a snapshot between the last full backup and the differential backup. If you want a differential backup, but a snapshot exists, then do another full backup before proceeding with the differential backup.

  You can look for the last full backup or snapshot using the following example SQL query:

```
select top 1
database_name
,  backup_start_date
,  backup_finish_date
from    msdb.dbo.backupset
where   database_name='mydatabase'
and     type = 'D'
order by backup_start_date desc;
```

- `@number_of_files` – The number of files into which the backup will be divided (chunked). The maximum number is 10.
  - Multifile backup is supported for both full and differential backups.
  - If you enter a value of 1 or omit the parameter, a single backup file is created.

Provide the prefix that the files have in common, then suffix that with an asterisk (*). The asterisk can be anywhere in the *file_name* part of the S3 ARN. The asterisk is replaced by a series of alphanumeric strings in the generated files, starting with 1-of-*number_of_files*.

For example, if the file names in the S3 ARN are backup*.bak and you set @number_of_files=4, the backup files generated are backup1-of-4.bak, backup2-of-4.bak, backup3-of-4.bak, and backup4-of-4.bak.

- If any of the file names already exists, and @overwrite_s3_backup_file is 0, an error is returned.
- Multifile backups can only have one asterisk in the *file_name* part of the S3 ARN.
- Single-file backups can have any number of asterisks in the *file_name* part of the S3 ARN. Asterisks aren't removed from the generated file name.

- @block_size – Block size (in bytes) specifying the physical block size for backup operations. Valid values are 512, 1024, 2048, 4096, 8192, 16384, 32768, and 65536
- @max_transfer_size – Maximum transfer size denotes the upper limit of data volume (in bytes) transmitted per I/O operation during the backup process. Valid values are multiples of 65536 bytes (64 KB) up to 4194304 bytes (4 MB).
- @buffer_count – Total number of I/O buffers to be use for the backup process.
- @enable_bucket_default_encryption – A value that indicates whether to use the S3 bucket's default encryption configuration for server-side encryption in S3. Directory buckets always use the bucket's default encryption configuration regardless of this setting.
  - 0 – Server-side encryption uses Advanced Encryption Standard (AES) 256-bit encryption through SSE-S3.
  - 1 – Server-side encryption uses your S3 bucket's configured default encryption.

Examples

Example of differential backup

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup1.bak',
@overwrite_s3_backup_file=1,
@type='DIFFERENTIAL';
```

Example of full backup with client-side encryption

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup1.bak',
@kms_master_key_arn='arn:aws:kms:us-east-1:123456789012:key/AKIAIOSFODNN7EXAMPLE',
@overwrite_s3_backup_file=1,
@type='FULL';
```

Example of multifile backup

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup*.bak',
@number_of_files=4;
```

Example of multifile differential backup

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup*.bak',
@type='DIFFERENTIAL',
@number_of_files=4;
```

Example of multifile backup with encryption

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup*.bak',
@kms_master_key_arn='arn:aws:kms:us-east-1:123456789012:key/AKIAIOSFODNN7EXAMPLE',
@number_of_files=4;
```

Example of multifile backup with S3 overwrite

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup*.bak',
@overwrite_s3_backup_file=1,
@number_of_files=4;
```

Example of backup with block size

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup*.bak',
@block_size=512;
```

Example of multifile backup with `@max_transfer_size` and `@buffer_count`

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup*.bak',
@number_of_files=4,
@max_transfer_size=4194304,
@buffer_count=10;
```

Example of single-file backup with the @number_of_files parameter

This example generates a backup file named `backup*.bak`.

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup*.bak',
@number_of_files=1;
```

Example of full backup with server-side encryption

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:::mybucket/backup*.bak',
@overwrite_s3_backup_file=1,
@type='FULL',
@enable_bucket_default_encryption=1;
```

Example of full backup using an access point

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:us-east-1:111122223333:accesspoint/my-access-point/
object/backup1.bak',
```

```
@overwrite_s3_backup_file=1,
@type='FULL';
```

Example of full backup using an access point for a directory bucket

```
exec msdb.dbo.rds_backup_database
@source_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3express:us-east-1:123456789012:accesspoint/my-access-point--use1-az6--xa-s3/object/backup1.bak',
@overwrite_s3_backup_file=1,
@type='FULL';
```

Restoring a database

To restore your database, call the `rds_restore_database` stored procedure. Amazon RDS creates an initial snapshot of the database after the restore task is complete and the database is open.

Usage

```
exec msdb.dbo.rds_restore_database
 @restore_db_name='database_name',
 @s3_arn_to_restore_from='arn:aws:s3:::bucket_name/file_name.extension',
 @with_norecovery=0|1,
 [@keep_cdc=0|1],
 [@data_file_volume='D:|H:|I:|J:'],
 [@log_file_volume='D:|H:|I:|J:'],
 [@kms_master_key_arn='arn:aws:kms:region:account-id:key/key-id'],
        [@block_size=512|1024|2048|4096|8192|16384|32768|65536],
        [@max_transfer_size=n],
        [@buffer_count=n],
 [@type='DIFFERENTIAL|FULL'];
```

The following parameters are required:

- `@restore_db_name` – The name of the database to restore. Database names are unique. You can't restore a database with the same name as an existing database.
- `@s3_arn_to_restore_from` – The ARN indicating the Amazon S3 prefix and names of the backup files used to restore the database.
  - For a single-file backup, provide the entire file name.

- For a multifile backup, provide the prefix that the files have in common, then suffix that with an asterisk (*).
  - If using a directory bucket, the ARN must end with `/*` due to differences for directory buckets.
- If `@s3_arn_to_restore_from` is empty, the following error message is returned: S3 ARN prefix cannot be empty.

The following parameter is required for differential restores, but optional for full restores:

- `@with_norecovery` – The recovery clause to use for the restore operation.
  - Set it to `0` to restore with RECOVERY. In this case, the database is online after the restore.
  - Set it to `1` to restore with NORECOVERY. In this case, the database remains in the RESTORING state after restore task completion. With this approach, you can do later differential restores.
  - For DIFFERENTIAL restores, specify `0` or `1`.
  - For `FULL` restores, this value defaults to `0`.

The following parameters are optional:

- `@keep_cdc` – Indicates whether to retain Change Data Capture (CDC) configuration on the restored database. Set to `1` to enable KEEP_CDC, `0` to disable. The default value is `0`.
- `@data_file_volume` – Specifies the drive letter for database data files. The default value is `D:`.
- `@log_file_volume` – Specifies the drive letter for database log files The default value is `D:`.
- `@kms_master_key_arn` – If you encrypted the backup file, the KMS key to use to decrypt the file.

  When you specify a KMS key, client-side encryption is used.
- `@type` – The type of restore. Valid types are `DIFFERENTIAL` and `FULL`. The default value is `FULL`.
- `@block_size` – Block size (in bytes) specifying the physical block size for backup operations. Valid values are 512, 1024, 2048, 4096, 8192, 16384, 32768, and 65536
- `@max_transfer_size` – Maximum transfer size denotes the upper limit of data volume (in bytes) transmitted per I/O operation during the backup process. Valid values are multiples of 65536 bytes (64 KB) up to 4194304 bytes (4 MB).
- `@buffer_count` – Total number of I/O buffers to be use for the backup process.

Note

For differential restores, either the database must be in the RESTORING state or a task must already exist that restores with NORECOVERY.
You can't restore later differential backups while the database is online.
You can't submit a restore task for a database that already has a pending restore task with RECOVERY.
Full restores with both NORECOVERY and KEEP_CDC aren't supported.
All native restores aren't supported on instances that have cross-region read replicas.
For supported configurations, restoring a database on a Multi-AZ instance with read replicas is similar to restoring a database on a Multi-AZ instance. You don't have to take any additional actions to restore a database on a replica.

Examples

Example of single-file restore

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak';
```

Example of multifile restore

To avoid errors when restoring multiple files, make sure that all the backup files have the same prefix, and that no other files use that prefix.

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup*';
```

Example of full database restore with RECOVERY

The following three examples perform the same task, full restore with RECOVERY.

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak';
```

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
[@type='DIFFERENTIAL|FULL'];
```

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
@type='FULL',
@with_norecovery=0;
```

Example of full database restore with encryption

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
@kms_master_key_arn='arn:aws:kms:us-east-1:123456789012:key/AKIAIOSFODNN7EXAMPLE';
```

Example of restore with block size

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
@block_size=512;
```

Example of multifile restore with @max_transfer_size and @buffer_count

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup*',
@max_transfer_size=4194304,
@buffer_count=10;
```

Example of full database restore with NORECOVERY

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
@type='FULL',
```

```
@with_norecovery=1;
```

Example of differential restore with NORECOVERY

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
@type='DIFFERENTIAL',
@with_norecovery=1;
```

Example of differential restore with RECOVERY

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
@type='DIFFERENTIAL',
@with_norecovery=0;
```

Example of full database restore with RECOVERY using an access point

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_backup_to='arn:aws:s3:us-east-1:111122223333:accesspoint/my-access-point/
object/backup1.bak',
@with_norecovery=0;
```

Example of full database restore with KEEP_CDC

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
@keep_cdc=1;
```

Restoring a log

To restore your log, call the `rds_restore_log` stored procedure.

Usage

```
exec msdb.dbo.rds_restore_log
```

```
@restore_db_name='database_name',
@s3_arn_to_restore_from='arn:aws:s3:::bucket_name/log_file_name.extension',
[@kms_master_key_arn='arn:aws:kms:region:account-id:key/key-id'],
[@with_norecovery=0|1],
[@keep_cdc=0|1],
[@stopat='datetime'],
[@block_size=512|1024|2048|4096|8192|16384|32768|65536],
       [@max_transfer_size=n],
       [@buffer_count=n];
```

The following parameters are required:

- `@restore_db_name` – The name of the database whose log to restore.
- `@s3_arn_to_restore_from` – The ARN indicating the Amazon S3 prefix and name of the log file used to restore the log. The file can have any extension, but `.trn` is usually used.

  If `@s3_arn_to_restore_from` is empty, the following error message is returned: S3 ARN prefix cannot be empty.

The following parameters are optional:

- `@keep_cdc` – Indicates whether to retain Change Data Capture (CDC) configuration on the restored database. Set to 1 to enable KEEP_CDC, 0 to disable. The default value is 0.
- `@kms_master_key_arn` – If you encrypted the log, the KMS key to use to decrypt the log.
- `@with_norecovery` – The recovery clause to use for the restore operation. This value defaults to 1.
  - Set it to `0` to restore with RECOVERY. In this case, the database is online after the restore. You can't restore further log backups while the database is online.
  - Set it to 1 to restore with NORECOVERY. In this case, the database remains in the RESTORING state after restore task completion. With this approach, you can do later log restores.
- `@stopat` – A value that specifies that the database is restored to its state at the date and time specified (in datetime format). Only transaction log records written before the specified date and time are applied to the database.

  If this parameter isn't specified (it is NULL), the complete log is restored.
- `@block_size` – Block size (in bytes) specifying the physical block size for backup operations. Valid values are 512, 1024, 2048, 4096, 8192, 16384, 32768, and 65536

- `@max_transfer_size` – Maximum transfer size denotes the upper limit of data volume (in bytes) transmitted per I/O operation during the backup process. Valid values are multiples of 65536 bytes (64 KB) up to 4194304 bytes (4 MB).
- `@buffer_count` – Total number of I/O buffers to be use for the backup process.

> **Note**
>
> For log restores, either the database must be in a state of restoring or a task must already exist that restores with NORECOVERY.
> You can't restore log backups while the database is online.
> You can't submit a log restore task on a database that already has a pending restore task with RECOVERY.

Examples

Example of log restore

```
exec msdb.dbo.rds_restore_log
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/mylog.trn';
```

Example of log restore with encryption

```
exec msdb.dbo.rds_restore_log
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/mylog.trn',
@kms_master_key_arn='arn:aws:kms:us-east-1:123456789012:key/AKIAIOSFODNN7EXAMPLE';
```

Example of log restore with NORECOVERY

The following two examples perform the same task, log restore with NORECOVERY.

```
exec msdb.dbo.rds_restore_log
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/mylog.trn',
@with_norecovery=1;
```

```
exec msdb.dbo.rds_restore_log
```

```
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/mylog.trn';
```

Example of restore with block size

```
exec msdb.dbo.rds_restore_log
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/mylog.trn',
@block_size=512;
```

Example of log restore with RECOVERY

```
exec msdb.dbo.rds_restore_log
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/mylog.trn',
@with_norecovery=0;
```

Example of log restore with STOPAT clause

```
exec msdb.dbo.rds_restore_log
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/mylog.trn',
@with_norecovery=0,
@stopat='2019-12-01 03:57:09';
```

Example of log restore with KEEP_CDC

```
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::mybucket/backup1.bak',
@keep_cdc=1;
```

Finishing a database restore

If the last restore task on the database was performed using `@with_norecovery=1`, the database is now in the RESTORING state. Open this database for normal operation by using the `rds_finish_restore` stored procedure.

Usage

```
exec msdb.dbo.rds_finish_restore @db_name='database_name';
```

> **Note**
>
> To use this approach, the database must be in the RESTORING state without any pending restore tasks.
> To finish restoring the database, use the master login. Or use the user login that most recently restored the database or log with NORECOVERY.

Working with partially restored databases

Dropping a partially restored database

To drop a partially restored database (left in the RESTORING state), use the `rds_drop_database` stored procedure.

```
exec msdb.dbo.rds_drop_database @db_name='database_name';
```

> **Note**
>
> You can't submit a DROP database request for a database that already has a pending restore or finish restore task.
> To drop the database, use the master login. Or use the user login that most recently restored the database or log with NORECOVERY.

Snapshot restore and point-in-time recovery behavior for partially restored databases

Partially restored databases in the source instance (left in the RESTORING state) are dropped from the target instance during snapshot restore and point-in-time recovery.

Canceling a task

To cancel a backup or restore task, call the `rds_cancel_task` stored procedure.

> **Note**
>
> You can't cancel a FINISH_RESTORE task.

Usage

```
exec msdb.dbo.rds_cancel_task @task_id=ID_number;
```

The following parameter is required:

- `@task_id` – The ID of the task to cancel. You can get the task ID by calling `rds_task_status`.

Tracking the status of tasks

To track the status of your backup and restore tasks, call the `rds_task_status` stored procedure. If you don't provide any parameters, the stored procedure returns the status of all tasks. The status for tasks is updated approximately every two minutes. Task history is retained for 36 days.

Usage

```
exec msdb.dbo.rds_task_status
 [@db_name='database_name'],
 [@task_id=ID_number];
```

The following parameters are optional:

- `@db_name` – The name of the database to show the task status for.
- `@task_id` – The ID of the task to show the task status for.

Examples

Example of listing the status for a specific task

```
exec msdb.dbo.rds_task_status @task_id=5;
```

Example of listing the status for a specific database and task

```
exec msdb.dbo.rds_task_status
@db_name='my_database',
@task_id=5;
```

Example of listing all tasks and their statuses on a specific database

```
exec msdb.dbo.rds_task_status @db_name='my_database';
```

Example of listing all tasks and their statuses on the current instance

```
exec msdb.dbo.rds_task_status;
```

Response

The `rds_task_status` stored procedure returns the following columns.

Column	Description
`task_id`	The ID of the task.

`task_type`	Task type depending on the input parameters, as follows:
	• For backup tasks:
	  • BACKUP_DB – Full database backup
	  • BACKUP_DB_DIFFERENTIAL – Differential database backup
	• For restore tasks:
	  • RESTORE_DB – Full database restore with RECOVERY
	  • RESTORE_DB_NORECOVERY – Full database restore with NORECOVERY
	  • RESTORE_DB_DIFFERENTIAL – Differential database restore with R ECOVERY
	  • RESTORE_DB_DIFFERENTIAL_NORECOVERY – Differential database restore with NORECOVERY
	  • RESTORE_DB_LOG – Log restore with RECOVERY
	  •

Column	Description
	RESTORE_DB_LOG_NORECOVERY – Log restore with NORECOVERY • For tasks that finish a restore: • FINISH_RESTORE – Finish restore and open database Amazon RDS creates an initial snapshot of the database after it is open on completion of the following restore tasks: • RESTORE_DB • RESTORE_DB_DIFFERENTIAL • RESTORE_DB_LOG • FINISH_RESTORE
`database_name`	The name of the database that the task is associated with.
`% complete`	The progress of the task as a percent value.
`duration (mins)`	The amount of time spent on the task, in minutes.

Column	Description
lifecycle	The status of the task. The possible statuses are the following: • CREATED – As soon as you call rds_backup_database or rds_restore_database , a task is created and the status is set to CREATED. • IN_PROGRESS – After a backup or restore task starts, the status is set to IN_PROGRESS . It can take up to 5 minutes for the status to change from CREATED to IN_PROGRESS . • SUCCESS – After a backup or restore task completes, the status is set to SUCCESS. • ERROR – If a backup or restore task fails, the status is set to ERROR. For more information about the error, see the task_info column. • CANCEL_REQUESTED – As soon as you call rds_cancel_task , the status of the task is set to CANCEL_REQUESTED . • CANCELLED – After a task is successfully canceled, the status of the task is set to CANCELLED .
task_info	Additional information about the task. If an error occurs while backing up or restoring a database, this column contains information about the error. For a list of possible errors, and mitigation strategies, see Troubleshooting.
last_updated	The date and time that the task status was last updated. The status is updated after every 5 percent of progress.
created_at	The date and time that the task was created.
S3_object_arn	The ARN indicating the Amazon S3 prefix and the name of the file that is being backed up or restored.

Column	Description
`overwrite_s3_backup_file`	The value of the `@overwrite_s3_backup_file` parameter specified when calling a backup task. For more information, see Backing up a database.
`KMS_master_key_arn`	The ARN for the KMS key used for encryption (for backup) and decryption (for restore).
`filepath`	Not applicable to native backup and restore tasks.
`overwrite_file`	Not applicable to native backup and restore tasks.

Compressing backup files

To save space in your Amazon S3 bucket, you can compress your backup files. For more information about compressing backup files, see Backup compression in the Microsoft documentation.

Compressing your backup files is supported for the following database editions:

- Microsoft SQL Server Enterprise Edition
- Microsoft SQL Server Standard Edition

To verify the compression option for your backup files, run the following code:

```
exec rdsadmin.dbo.rds_show_configuration 'S3 backup compression';
```

To turn on compression for your backup files, run the following code:

```
exec rdsadmin.dbo.rds_set_configuration 'S3 backup compression', 'true';
```

To turn off compression for your backup files, run the following code:

```
exec rdsadmin.dbo.rds_set_configuration 'S3 backup compression', 'false';
```

Troubleshooting

The following are issues you might encounter when you use native backup and restore.

Issue	Troubleshooting suggestions
Database backup/restore option is not enabled yet or is in the process of being enabled. Please try again later.	Make sure that you have added the SQLSERVER_BACKUP_RESTORE option to the DB option group associated with your DB instance. For more information, see Adding the native backup and restore option.
The EXECUTE permission was denied on the object '*rds_backup_database* ', database 'msdb', schema 'dbo'.	Make sure that you are using the master user when executing the stored procedure. If you encounter this error even after being logged in as the master user, it might be due to the admin user permissions being misaligned. To reset the master user, use the AWS Management Console. See Resetting the db_owner role membership for master user for Amazon RDS for SQL Server.
The EXECUTE permission was denied on the object '*rds_restore_database* ', database 'msdb', schema 'dbo'.	Make sure that you are using the master user when executing the stored procedure. If you encounter this error even after being logged in as the master user, it might be due to the admin user permissions being misaligned. To reset the master user, use the AWS Management Console. See Resetting the db_owner role membership for master user for Amazon RDS for SQL Server.
Access Denied	The backup or restore process can't access the backup file. This is usually caused by issues like the following: • Referencing the incorrect bucket. Referencing the bucket using an incorrect format. Referencing a file name without using the ARN. • Incorrect permissions on the bucket file. For example, if it is created by a different account that is trying to access it now, add the correct permissions. •

Issue	Troubleshooting suggestions
	An IAM policy that is incorrect or incomplete. Your IAM role must include all the necessary elements, including, for example, the correct version. These are highlighted in Importing and exporting SQL Server databases using native backup and restore.
BACKUP DATABASE WITH COMPRESSION isn't supported on <edition_name> Edition	Compressing your backup files is only supported for Microsoft SQL Server Enterprise Edition and Standard Edition. For more information, see Compressing backup files.
Key <ARN> does not exist	You attempted to restore an encrypted backup, but didn't provide a valid encryption key. Check your encryption key and retry. For more information, see Restoring a database.
Please reissue task with correct type and overwrite property	If you attempt to back up your database and provide the name of a file that already exists, but set the overwrite property to false, the save operation fails. To fix this error, either provide the name of a file that doesn't already exist, or set the overwrite property to true. For more information, see Backing up a database. It's also possible that you intended to restore your database, but called the `rds_backup_database` stored procedure accidentally. In that case, call the `rds_restore_database` stored procedure instead. For more information, see Restoring a database. If you intended to restore your database and called the `rds_restore_database` stored procedure, make sure that you provided the name of a valid backup file. For more information, see Using native backup and restore.

Issue	Troubleshooting suggestions
Please specify a bucket that is in the same region as RDS instance	You can't back up to, or restore from, an Amazon S3 bucket in a different AWS Region from your Amazon RDS DB instance. You can use Amazon S3 replication to copy the backup file to the correct AWS Region. For more information, see Cross-Region replication in the Amazon S3 documentation.
The specified bucket does not exist	Verify that you have provided the correct ARN for your bucket and file, in the correct format. For more information, see Using native backup and restore.
User <ARN> is not authorized to perform <kms action> on resource <ARN>	You requested an encrypted operation, but didn't provide correct AWS KMS permissions. Verify that you have the correct permissio ns, or add them. For more information, see Setting up for native backup and restore.
The Restore task is unable to restore from more than 10 backup file(s). Please reduce the number of files matched and try again.	Reduce the number of files that you're trying to restore from. You can make each individual file larger if necessary.
Database '*database_name* ' already exists. Two databases that differ only by case or accent are not allowed. Choose a different database name.	You can't restore a database with the same name as an existing database. Database names are unique.

Importing and exporting SQL Server data using other methods

Following, you can find information about using snapshots to import your Microsoft SQL Server data to Amazon RDS. You can also find information about using snapshots to export your data from an RDS DB instance running SQL Server.

If your scenario supports it, it's easier to move data in and out of Amazon RDS by using the native backup and restore functionality. For more information, see Importing and exporting SQL Server databases using native backup and restore.

> **Note**
>
> Amazon RDS for Microsoft SQL Server doesn't support importing data into the `msdb` database.

Importing data into RDS for SQL Server by using a snapshot

To import data into a SQL Server DB instance by using a snapshot

1. Create a DB instance. For more information, see Creating an Amazon RDS DB instance.
2. Stop applications from accessing the destination DB instance.

   If you prevent access to your DB instance while you are importing data, data transfer is faster. Additionally, you don't need to worry about conflicts while data is being loaded if other applications cannot write to the DB instance at the same time. If something goes wrong and you have to roll back to an earlier database snapshot, the only changes that you lose are the imported data. You can import this data again after you resolve the issue.

   For information about controlling access to your DB instance, see Controlling access with security groups.
3. Create a snapshot of the target database.

   If the target database is already populated with data, we recommend that you take a snapshot of the database before you import the data. If something goes wrong with the data import or you want to discard the changes, you can restore the database to its previous state by using the snapshot. For information about database snapshots, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

> **Note**
>
> When you take a database snapshot, I/O operations to the database are suspended for a moment (milliseconds) while the backup is in progress.

4. Disable automated backups on the target database.

   Disabling automated backups on the target DB instance improves performance while you are importing your data because Amazon RDS doesn't log transactions when automatic backups are disabled. However, there are some things to consider. Automated backups are required to perform a point-in-time recovery. Thus, you can't restore the database to a specific point in time while you are importing data. Additionally, any automated backups that were created on the DB instance are erased unless you choose to retain them.

   Choosing to retain the automated backups can help protect you against accidental deletion of data. Amazon RDS also saves the database instance properties along with each automated backup to make it easy to recover. Using this option lets you restore a deleted database instance to a specified point in time within the backup retention period even after deleting it. Automated backups are automatically deleted at the end of the specified backup window, just as they are for an active database instance.

   You can also use previous snapshots to recover the database, and any snapshots that you have taken remain available. For information about automated backups, see Introduction to backups.

5. Disable foreign key constraints, if applicable.

   If you need to disable foreign key constraints, you can do so with the following script.

```
--Disable foreign keys on all tables
    DECLARE @table_name SYSNAME;
    DECLARE @cmd NVARCHAR(MAX);
    DECLARE table_cursor CURSOR FOR SELECT name FROM sys.tables;

    OPEN table_cursor;
    FETCH NEXT FROM table_cursor INTO @table_name;

    WHILE @@FETCH_STATUS = 0 BEGIN
      SELECT @cmd = 'ALTER TABLE '+QUOTENAME(@table_name)+' NOCHECK CONSTRAINT
 ALL';
```

```
      EXEC (@cmd);
      FETCH NEXT FROM table_cursor INTO @table_name;
    END

    CLOSE table_cursor;
    DEALLOCATE table_cursor;

    GO
```

6. Drop indexes, if applicable.
7. Disable triggers, if applicable.

   If you need to disable triggers, you can do so with the following script.

```
--Disable triggers on all tables
    DECLARE @enable BIT = 0;
    DECLARE @trigger SYSNAME;
    DECLARE @table SYSNAME;
    DECLARE @cmd NVARCHAR(MAX);
    DECLARE trigger_cursor CURSOR FOR SELECT trigger_object.name trigger_name,
     table_object.name table_name
    FROM sysobjects trigger_object
    JOIN sysobjects table_object ON trigger_object.parent_obj = table_object.id
    WHERE trigger_object.type = 'TR';

    OPEN trigger_cursor;
    FETCH NEXT FROM trigger_cursor INTO @trigger, @table;

    WHILE @@FETCH_STATUS = 0 BEGIN
      IF @enable = 1
         SET @cmd = 'ENABLE ';
      ELSE
         SET @cmd = 'DISABLE ';

      SET @cmd = @cmd + ' TRIGGER dbo.'+QUOTENAME(@trigger)+' ON
 dbo.'+QUOTENAME(@table)+' ';
      EXEC (@cmd);
      FETCH NEXT FROM trigger_cursor INTO @trigger, @table;
    END

    CLOSE trigger_cursor;
    DEALLOCATE trigger_cursor;
```

```
    GO
```

8. Query the source SQL Server instance for any logins that you want to import to the destination DB instance.

   SQL Server stores logins and passwords in the `master` database. Because Amazon RDS doesn't grant access to the `master` database, you cannot directly import logins and passwords into your destination DB instance. Instead, you must query the `master` database on the source SQL Server instance to generate a data definition language (DDL) file. This file should include all logins and passwords that you want to add to the destination DB instance. This file also should include role memberships and permissions that you want to transfer.

   For information about querying the `master` database, see Transfer logins and passwords between instances of SQL Server in the Microsoft Knowledge Base.

   The output of the script is another script that you can run on the destination DB instance. The script in the Knowledge Base article has the following code:

   ```
   p.type IN
   ```

   Every place `p.type` appears, use the following code instead:

   ```
   p.type = 'S'
   ```

9. Import the data using the method in Import the data.
10. Grant applications access to the target DB instance.

    When your data import is complete, you can grant access to the DB instance to those applications that you blocked during the import. For information about controlling access to your DB instance, see Controlling access with security groups.

11. Enable automated backups on the target DB instance.

    For information about automated backups, see Introduction to backups.

12. Enable foreign key constraints.

    If you disabled foreign key constraints earlier, you can now enable them with the following script.

    ```
    --Enable foreign keys on all tables
    ```

```
    DECLARE @table_name SYSNAME;
    DECLARE @cmd NVARCHAR(MAX);
    DECLARE table_cursor CURSOR FOR SELECT name FROM sys.tables;

    OPEN table_cursor;
    FETCH NEXT FROM table_cursor INTO @table_name;

    WHILE @@FETCH_STATUS = 0 BEGIN
      SELECT @cmd = 'ALTER TABLE '+QUOTENAME(@table_name)+' CHECK CONSTRAINT ALL';
      EXEC (@cmd);
      FETCH NEXT FROM table_cursor INTO @table_name;
    END

    CLOSE table_cursor;
    DEALLOCATE table_cursor;
```

13. Enable indexes, if applicable.
14. Enable triggers, if applicable.

    If you disabled triggers earlier, you can now enable them with the following script.

```
--Enable triggers on all tables
    DECLARE @enable BIT = 1;
    DECLARE @trigger SYSNAME;
    DECLARE @table SYSNAME;
    DECLARE @cmd NVARCHAR(MAX);
    DECLARE trigger_cursor CURSOR FOR SELECT trigger_object.name trigger_name,
     table_object.name table_name
    FROM sysobjects trigger_object
    JOIN sysobjects table_object ON trigger_object.parent_obj = table_object.id
    WHERE trigger_object.type = 'TR';

    OPEN trigger_cursor;
    FETCH NEXT FROM trigger_cursor INTO @trigger, @table;

    WHILE @@FETCH_STATUS = 0 BEGIN
      IF @enable = 1
         SET @cmd = 'ENABLE ';
      ELSE
         SET @cmd = 'DISABLE ';

      SET @cmd = @cmd + ' TRIGGER dbo.'+QUOTENAME(@trigger)+' ON
 dbo.'+QUOTENAME(@table)+' ';
```

```
    EXEC (@cmd);
    FETCH NEXT FROM trigger_cursor INTO @trigger, @table;
END

CLOSE trigger_cursor;
DEALLOCATE trigger_cursor;
```

Import the data

Microsoft SQL Server Management Studio is a graphical SQL Server client that is included in all Microsoft SQL Server editions except the Express Edition. SQL Server Management Studio Express is available from Microsoft as a free download. To find this download, see the Microsoft website.

> **Note**
>
> SQL Server Management Studio is available only as a Windows-based application.

SQL Server Management Studio includes the following tools, which are useful in importing data to a SQL Server DB instance:

- Generate and Publish Scripts Wizard
- Import and Export Wizard
- Bulk copy

Generate and Publish Scripts Wizard

The Generate and Publish Scripts Wizard creates a script that contains the schema of a database, the data itself, or both. You can generate a script for a database in your local SQL Server deployment. You can then run the script to transfer the information that it contains to an Amazon RDS DB instance.

> **Note**
>
> For databases of 1 GiB or larger, it's more efficient to script only the database schema. You then use the Import and Export Wizard or the bulk copy feature of SQL Server to transfer the data.

For detailed information about the Generate and Publish Scripts Wizard, see the Microsoft SQL Server documentation.

In the wizard, pay particular attention to the advanced options on the **Set Scripting Options** page to ensure that everything you want your script to include is selected. For example, by default, database triggers are not included in the script.

When the script is generated and saved, you can use SQL Server Management Studio to connect to your DB instance and then run the script.

Import and Export Wizard

The Import and Export Wizard creates a special Integration Services package, which you can use to copy data from your local SQL Server database to the destination DB instance. The wizard can filter which tables and even which tuples within a table are copied to the destination DB instance.

> **Note**
>
> The Import and Export Wizard works well for large datasets, but it might not be the fastest way to remotely export data from your local deployment. For an even faster way, consider the SQL Server bulk copy feature.

For detailed information about the Import and Export Wizard, see the Microsoft SQL Server documentation.

In the wizard, on the **Choose a Destination** page, do the following:

- For **Server Name**, type the name of the endpoint for your DB instance.
- For the server authentication mode, choose **Use SQL Server Authentication**.
- For **User name** and **Password**, type the credentials for the master user that you created for the DB instance.

Bulk copy

The SQL Server bulk copy feature is an efficient means of copying data from a source database to your DB instance. Bulk copy writes the data that you specify to a data file, such as an ASCII file. You can then run bulk copy again to write the contents of the file to the destination DB instance.

This section uses the **bcp** utility, which is included with all editions of SQL Server. For detailed information about bulk import and export operations, see the Microsoft SQL Server documentation.

> **Note**
>
> Before you use bulk copy, you must first import your database schema to the destination DB instance. The Generate and Publish Scripts Wizard, described earlier in this topic, is an excellent tool for this purpose.

The following command connects to the local SQL Server instance. It generates a tab-delimited file of a specified table in the C:\ root directory of your existing SQL Server deployment. The table is specified by its fully qualified name, and the text file has the same name as the table that is being copied.

```
bcp dbname.schema_name.table_name out C:\table_name.txt -n -S localhost -U username -
P password -b 10000
```

The preceding code includes the following options:

- `-n` specifies that the bulk copy uses the native data types of the data to be copied.
- `-S` specifies the SQL Server instance that the *bcp* utility connects to.
- `-U` specifies the user name of the account to log in to the SQL Server instance.
- `-P` specifies the password for the user specified by `-U`.
- `-b` specifies the number of rows per batch of imported data.

> **Note**
>
> There might be other parameters that are important to your import situation. For example, you might need the `-E` parameter that pertains to identity values. For more information; see the full description of the command line syntax for the **bcp** utility in the Microsoft SQL Server documentation.

For example, suppose that a database named `store` that uses the default schema, dbo, contains a table named `customers`. The user account `admin`, with the password `insecure`, copies 10,000 rows of the `customers` table to a file named `customers.txt`.

```
bcp store.dbo.customers out C:\customers.txt -n -S localhost -U admin -P insecure -b
 10000
```

After you generate the data file, you can upload the data to your DB instance by using a similar command. Beforehand, create the database and schema on the target DB instance. Then use the `in` argument to specify an input file instead of `out` to specify an output file. Instead of using localhost to specify the local SQL Server instance, specify the endpoint of your DB instance. If you use a port other than 1433, specify that too. The user name and password are the master user and password for your DB instance. The syntax is as follows.

```
bcp dbname.schema_name.table_name
     in C:\table_name.txt -n -S endpoint,port -U master_user_name -
P master_user_password -b 10000
```

To continue the previous example, suppose that the master user name is `admin`, and the password is `insecure`. The endpoint for the DB instance is `rds.ckz2kqd4qsn1.us-east-1.rds.amazonaws.com`, and you use port 4080. The command is as follows.

```
bcp store.dbo.customers in C:\customers.txt -n -S rds.ckz2kqd4qsn1.us-
east-1.rds.amazonaws.com,4080 -U admin -P insecure -b 10000
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Exporting data from RDS for SQL Server

You can choose one of the following options to export data from an RDS for SQL Server DB instance:

- **Native database backup using a full backup file (.bak)** – Using .bak files to backup databases is heavily optimized, and is usually the fastest way to export data. For more information, see Importing and exporting SQL Server databases using native backup and restore.

- **SQL Server Import and Export Wizard** – For more information, see SQL Server Import and Export Wizard.
- **SQL Server Generate and Publish Scripts Wizard and bcp utility** – For more information, see SQL Server Generate and Publish Scripts Wizard and bcp utility.

SQL Server Import and Export Wizard

You can use the SQL Server Import and Export Wizard to copy one or more tables, views, or queries from your RDS for SQL Server DB instance to another data store. This choice is best if the target data store is not SQL Server. For more information, see SQL Server Import and Export Wizard in the SQL Server documentation.

The SQL Server Import and Export Wizard is available as part of Microsoft SQL Server Management Studio. This graphical SQL Server client is included in all Microsoft SQL Server editions except the Express Edition. SQL Server Management Studio is available only as a Windows-based application. SQL Server Management Studio Express is available from Microsoft as a free download. To find this download, see the Microsoft website.

To use the SQL Server Import and Export Wizard to export data

1. In SQL Server Management Studio, connect to your RDS for SQL Server DB instance. For details on how to do this, see Connecting to your Microsoft SQL Server DB instance.
2. In **Object Explorer**, expand **Databases**, open the context (right-click) menu for the source database, choose **Tasks**, and then choose **Export Data**. The wizard appears.
3. On the **Choose a Data Source** page, do the following:
    a. For **Data source**, choose **`SQL Server Native Client 11.0`**.
    b. Verify that the **Server name** box shows the endpoint of your RDS for SQL Server DB instance.
    c. Select **Use SQL Server Authentication**. For **User name** and **Password**, type the master user name and password of your DB instance.
    d. Verify that the **Database** box shows the database from which you want to export data.
    e. Choose **Next**.
4. On the **Choose a Destination** page, do the following:
    a. For **Destination**, choose **`SQL Server Native Client 11.0`**.

> **Note**
>
> Other target data sources are available. These include .NET Framework data providers, OLE DB providers, SQL Server Native Client providers, ADO.NET providers, Microsoft Office Excel, Microsoft Office Access, and the Flat File source. If you choose to target one of these data sources, skip the remainder of step 4. For details on the connection information to provide next, see Choose a destination in the SQL Server documentation.

b. For **Server name**, type the server name of the target SQL Server DB instance.

c. Choose the appropriate authentication type. Type a user name and password if necessary.

d. For **Database**, choose the name of the target database, or choose **New** to create a new database to contain the exported data.

If you choose **New**, see Create database in the SQL Server documentation for details on the database information to provide.

e. Choose **Next**.

5. On the **Table Copy or Query** page, choose **Copy data from one or more tables or views** or **Write a query to specify the data to transfer**. Choose **Next**.
6. If you chose **Write a query to specify the data to transfer**, you see the **Provide a Source Query** page. Type or paste in a SQL query, and then choose **Parse** to verify it. Once the query validates, choose **Next**.
7. On the **Select Source Tables and Views** page, do the following:

a. Select the tables and views that you want to export, or verify that the query you provided is selected.

b. Choose **Edit Mappings** and specify database and column mapping information. For more information, see Column mappings in the SQL Server documentation.

c. (Optional) To see a preview of data to be exported, select the table, view, or query, and then choose **Preview**.

d. Choose **Next**.

8. On the **Run Package** page, verify that **Run immediately** is selected. Choose **Next**.
9. On the **Complete the Wizard** page, verify that the data export details are as you expect. Choose **Finish**.

10. On the **The execution was successful** page, choose **Close**.

SQL Server Generate and Publish Scripts Wizard and bcp utility

You can use the SQL Server Generate and Publish Scripts Wizard to create scripts for an entire database or just selected objects. You can run these scripts on a target SQL Server DB instance to recreate the scripted objects. You can then use the bcp utility to bulk export the data for the selected objects to the target DB instance. This choice is best if you want to move a whole database (including objects other than tables) or large quantities of data between two SQL Server DB instances. For a full description of the bcp command-line syntax, see bcp utility in the Microsoft SQL Server documentation.

The SQL Server Generate and Publish Scripts Wizard is available as part of Microsoft SQL Server Management Studio. This graphical SQL Server client is included in all Microsoft SQL Server editions except the Express Edition. SQL Server Management Studio is available only as a Windows-based application. SQL Server Management Studio Express is available from Microsoft as a free download.

To use the SQL Server Generate and Publish Scripts Wizard and the bcp utility to export data

1. In SQL Server Management Studio, connect to your RDS for SQL Server DB instance. For details on how to do this, see Connecting to your Microsoft SQL Server DB instance.
2. In **Object Explorer**, expand the **Databases** node and select the database you want to script.
3. Follow the instructions in Generate and publish scripts Wizard in the SQL Server documentation to create a script file.
4. In SQL Server Management Studio, connect to your target SQL Server DB instance.
5. With the target SQL Server DB instance selected in **Object Explorer**, choose **Open** on the **File** menu, choose **File**, and then open the script file.
6. If you have scripted the entire database, review the CREATE DATABASE statement in the script. Make sure that the database is being created in the location and with the parameters that you want. For more information, see CREATE DATABASE in the SQL Server documentation.
7. If you are creating database users in the script, check to see if server logins exist on the target DB instance for those users. If not, create logins for those users; the scripted commands to create the database users fail otherwise. For more information, see Create a login in the SQL Server documentation.

8. Choose **!Execute** on the SQL Editor menu to run the script file and create the database objects. When the script finishes, verify that all database objects exist as expected.
9. Use the bcp utility to export data from the RDS for SQL Server DB instance into files. Open a command prompt and type the following command.

   ```
   bcp database_name.schema_name.table_name out data_file -n -S aws_rds_sql_endpoint -
   U username -P password
   ```

   The preceding code includes the following options:

   - *table_name* is the name of one of the tables that you've recreated in the target database and now want to populate with data.
   - *data_file* is the full path and name of the data file to be created.
   - `-n` specifies that the bulk copy uses the native data types of the data to be copied.
   - `-S` specifies the SQL Server DB instance to export from.
   - `-U` specifies the user name to use when connecting to the SQL Server DB instance.
   - `-P` specifies the password for the user specified by `-U`.

   The following shows an example command.

   ```
   bcp world.dbo.city out C:\Users\JohnDoe\city.dat -n -S sql-jdoe.1234abcd.us-
   west-2.rds.amazonaws.com,1433 -U JohnDoe -P ClearTextPassword
   ```

   Repeat this step until you have data files for all of the tables you want to export.
10. Prepare your target DB instance for bulk import of data by following the instructions at Basic guidelines for bulk importing data in the SQL Server documentation.
11. Decide on a bulk import method to use after considering performance and other concerns discussed in About bulk import and bulk export operations in the SQL Server documentation.
12. Bulk import the data from the data files that you created using the bcp utility. To do so, follow the instructions at either Import and export bulk data by using the bcp utility or Import bulk data by using BULK INSERT or OPENROWSET(BULK...) in the SQL Server documentation, depending on what you decided in step 11.

Using BCP utility from Linux to import and export data

The BCP (Bulk Copy Program) utility provides an efficient way to transfer large amounts of data between your RDS for SQL Server DB instance and data files. You can use BCP from Linux environments to perform bulk data operations, making it useful for data migration, ETL processes, and regular data transfers.

BCP supports both importing data from files into SQL Server tables and exporting data from SQL Server tables to files. This is particularly effective for transferring structured data in various formats including delimited text files.

Prerequisites

Before using BCP with your RDS for SQL Server DB instance from Linux, ensure you have the following:

- A Linux environment with network connectivity to your RDS for SQL Server DB instance
- Microsoft SQL Server command-line tools installed on your Linux system, including:
  - sqlcmd - SQL Server command-line query tool
  - bcp - Bulk Copy Program utility
- Valid credentials for your RDS for SQL Server DB instance
- Network access configured through security groups to allow connections on the SQL Server port (typically 1433)
- Appropriate database permissions for the operations you want to perform

Installing SQL Server command-line tools on Linux

To use BCP from Linux, you need to install the Microsoft SQL Server command-line tools. For detailed installation instructions for your specific Linux distribution, see the following Microsoft documentation:

- Install sqlcmd and bcp the SQL Server command-line tools on Linux
- bcp utility - Complete reference for the BCP utility

After installation, ensure the tools are available in your PATH by running:

```
bcp -v
```

```
sqlcmd -?
```

Exporting data from RDS for SQL Server

You can use BCP to export data from your RDS for SQL Server DB instance to files on your Linux system. This is useful for creating backups, data analysis, or preparing data for migration.

Basic export syntax

The basic syntax for exporting data using BCP is:

```
bcp database.schema.table out output_file -S server_name -U username -P password
 [options]
```

Where:

- `database.schema.table` - The fully qualified table name
- `output_file` - The path and name of the output file
- `server_name` - Your RDS for SQL Server endpoint
- `username` - Your database username
- `password` - Your database password

Export example

The following example exports data from a table named `customers` in the `sales` database:

```
bcp sales.dbo.customers out /home/user/customers.txt \
    -S mydb.cluster-abc123.us-east-1.rds.amazonaws.com \
    -U admin \
    -P mypassword \
    -c \
    -t "|" \
    -r "\n"
```

This command:

- Exports data from the `customers` table
- Saves the output to `/home/user/customers.txt`

- Uses character format (`-c`)
- Uses pipe (|) as the field delimiter (`-t "|"`)
- Uses newline as the row delimiter (`-r "\n"`)

Importing data to RDS for SQL Server

You can use BCP to import data from files on your Linux system into your RDS for SQL Server DB instance. This is useful for data migration, loading test data, or regular data updates.

Basic import syntax

The basic syntax for importing data using BCP is:

```
bcp database.schema.table in input_file -S server_name -U username -P password
 [options]
```

Where:

- `database.schema.table` - The fully qualified destination table name
- `input_file` - The path and name of the input file
- `server_name` - Your RDS for SQL Server endpoint
- `username` - Your database username
- `password` - Your database password

Import example

The following example imports data from a file into a table named `customers`:

```
bcp sales.dbo.customers in /home/user/customers.txt \
    -S mydb.cluster-abc123.us-east-1.rds.amazonaws.com \
    -U admin \
    -P mypassword \
    -c \
    -t "|" \
    -r "\n" \
    -b 1000
```

This command:

- Imports data into the `customers` table
- Reads data from `/home/user/customers.txt`
- Uses character format (`-c`)
- Uses pipe (|) as the field delimiter (`-t "|"`)
- Uses newline as the row delimiter (`-r "\n"`)
- Processes data in batches of 1000 rows (`-b 1000`)

Common BCP options

BCP provides numerous options to control data formatting and transfer behavior. The following table describes commonly used options:

Option	Description
`-c`	Uses character data type for all columns
`-n`	Uses native database data types
`-t`	Specifies the field delimiter (default is tab)
`-r`	Specifies the row delimiter (default is newline)
`-b`	Specifies the batch size for bulk operations
`-F`	Specifies the first row to export or import
`-L`	Specifies the last row to export or import
`-e`	Specifies an error file to capture rejected rows
`-f`	Specifies a format file for data formatting
`-q`	Uses quoted identifiers for object names

Best practices and considerations

When using BCP with RDS for SQL Server from Linux, consider the following best practices:

- **Use batch processing** – For large datasets, use the `-b` option to process data in batches. T his improves performance and allows for better error recovery.
- **Handle errors gracefully** – Use the `-e` option to capture error information and rejected rows in a separate file for analysis.
- **Choose appropriate data formats** – Use character format (`-c`) for cross-platform compatibility or native format (`-n`) for better performance when both source and destination are SQL Server.
- **Secure your credentials** – Avoid putting passwords directly in command lines. Consider using environment variables or configuration files with appropriate permissions.
- **Test with small datasets** – Before processing large amounts of data, test your BCP commands with smaller datasets to verify formatting and connectivity.
- **Monitor network connectivity** – Ensure stable network connections, especially for large data transfers. Consider using tools like `screen` or `tmux` for long-running operations.
- **Validate data integrity** – After data transfer, verify row counts and sample data to ensure the operation completed successfully.

Troubleshooting common issues

The following table describes common issues you might encounter when using BCP from Linux and their solutions:

Issue	Solution
Connection timeout or network errors	Verify your Amazon RDS endpoint, security group settings, and network connectivity. Make sure the SQL Server port (typically 1433) is accessible from your Linux system.
Authentication failures	Verify your username and password. Make sure the database user has appropriate permissions for the operations you're performing.
Data format errors	Check your field and row delimiters. Make sure the data format matches what BCP expects. Use format files for complex data structures.

Issue	Solution
Permission denied errors	Make sure your database user has `INSERT` permissions for imports or `SELECT` permissions for exports on the target tables.
Large file handling issues	Use batch processing with the `-b` option. Consider splitting large files into smaller chunks for better performance and error recovery.
Character encoding problems	Ensure your data files use compatible character encoding. Use the `-c` option for character format or specify appropriate code pages.

Working with read replicas for Microsoft SQL Server in Amazon RDS

You usually use read replicas to configure replication between Amazon RDS DB instances. For general information about read replicas, see Working with DB instance read replicas.

In this section, you can find specific information about working with read replicas on Amazon RDS for SQL Server.

- Synchronizing database users and objects with a SQL Server read replica
- Troubleshooting a SQL Server read replica problem

Configuring read replicas for SQL Server

Before a DB instance can serve as a source instance for replication, you must enable automatic backups on the source DB instance. To do so, you set the backup retention period to a value other than 0. Setting this type of deployment also enforces that automatic backups are enabled.

Creating a SQL Server read replica doesn't require an outage for the primary DB instance. Amazon RDS sets the necessary parameters and permissions for the source DB instance and the read replica without any service interruption. A snapshot is taken of the source DB instance, and this snapshot becomes the read replica. No outage occurs when you delete a read replica.

You can create up to 15 read replicas from one source DB instance. For replication to operate effectively, we recommend that you configure each read replica with the same amount of compute and storage resources as the source DB instance. If you scale the source DB instance, also scale the read replicas.

The SQL Server DB engine version of the source DB instance and all of its read replicas must be the same. Amazon RDS upgrades the primary immediately after upgrading the read replicas, regardless of the maintenance window. For more information about upgrading the DB engine version, see Upgrades of the Microsoft SQL Server DB engine.

For a read replica to receive and apply changes from the source, it should have sufficient compute and storage resources. If a read replica reaches compute, network, or storage resource capacity, the read replica stops receiving or applying changes from its source. You can modify the storage and CPU resources of a read replica independently from its source and other read replicas.

For more information about how to create a read replica, see Creating a read replica.

Read replica limitations with SQL Server

The following limitations apply to SQL Server read replicas on Amazon RDS:

- Read replicas are only available on the SQL Server Enterprise Edition (EE) engine.
- Read replicas are available for SQL Server versions 2016–2022.
- You can create up to 15 read replicas from one source DB instance. Replication might lag when your source DB instance has more than 5 read replicas.
- Read replicas are only available for DB instances running on DB instance classes with four or more vCPUs.
- A read replica supports up to 100 databases depending on the instance class type and availability mode. You must create databases on the source DB instance to automatically replicate them to the read replicas. You can't choose individual databases to replicate. For more information, see Limitations for Microsoft SQL Server DB instances.
- You can't drop a database from a read replica. To drop a database, drop it from the source DB instance with the `rds_drop_database` stored procedure. For more information, see Dropping a database in an Amazon RDS for Microsoft SQL Server DB instance.
- If the source DB instance uses Transparent Data Encryption (TDE) to encrypt data, the read replica also automatically configures TDE.

  If the source DB instance uses a KMS key to encrypt data, read replicas in the same region use the same KMS key. For cross-region read replicas, you must specify a KMS key from the read replica's region when creating the read replica. You can't change the KMS key for a read replica.
- Read replicas have the same time zone and collation as the source DB instance, regardless of Availabilty Zone they're created in.
- The following aren't supported on Amazon RDS for SQL Server:
  - Backup retention of read replicas
  - Point-in-time recovery from read replicas
  - Manual snapshots of read replicas
  - Multi-AZ read replicas
  - Creating read replicas of read replicas
  - Synchronization of user logins to read replicas

- Amazon RDS for SQL Server doesn't intervene to mitigate high replica lag between a source DB instance and its read replicas. Make sure that the source DB instance and its read replicas are sized properly, in terms of computing power and storage, to suit their operational load.
- You can replicate between the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions, but not into or out of AWS GovCloud (US) Regions.

Option considerations for RDS for SQL Server replicas

Before you create an RDS for SQL Server replica, consider the following requirements, restrictions, and recommendations:

- If your SQL Server replica is in the same Region as its source DB instance, make sure that it belongs to the same option group as the source DB instance. Modifications to the source option group or source option group membership propagate to replicas. These changes are applied to the replicas immediately after they are applied to the source DB instance, regardless of the replica's maintenance window.

  For more information about option groups, see Working with option groups.
- When you create a SQL Server cross-Region replica, Amazon RDS creates a dedicated option group for it.

  You can't remove an SQL Server cross-Region replica from its dedicated option group. No other DB instances can use the dedicated option group for a SQL Server cross-Region replica.

  The following options are replicated options. To add replicated options to a SQL Server cross-Region replica, add it to the source DB instance's option group. The option is also installed on all of the source DB instance's replicas.

  - `TDE`

  The following options are non-replicated options. You can add or remove non-replicated options from a dedicated option group.

  - `MSDTC`
  - `SQLSERVER_AUDIT`
  - To enable the `SQLSERVER_AUDIT` option on cross-Region read replica, add the `SQLSERVER_AUDIT` option on the dedicated option group on the cross-region read replica and the source instance's option group. By adding the `SQLSERVER_AUDIT` option on the source instance of SQL Server cross-Region read replica, you can create Server Level Audit Object

and Server Level Audit Specifications on each of the cross-Region read replicas of the source instance. To allow the cross-Region read replicas access to upload the completed audit logs to an Amazon S3 bucket, add the `SQLSERVER_AUDIT` option to the dedicated option group and configure the option settings. The Amazon S3 bucket that you use as a target for audit files must be in the same Region as the cross-Region read replica. You can modify the option setting of the `SQLSERVER_AUDIT` option for each cross region read replica independently so each can access an Amazon S3 bucket in their respective Region.

The following options are not supported for read replicas.

- `SSRS`
- `SSAS`
- `SSIS`

The following options are partially supported for cross-Region read replicas.

- `SQLSERVER_BACKUP_RESTORE`
- The source DB instance of a SQL Server cross-Region replica can have the `SQLSERVER_BACKUP_RESTORE` option, but you can not perform native restores on the source DB instance until you delete all its cross-Region replicas. Any existing native restore tasks will be cancelled during the creation of a cross-Region replica. You can't add the `SQLSERVER_BACKUP_RESTORE` option to a dedicated option group.

  For more information on native backup and restore, see Importing and exporting SQL Server databases using native backup and restore

When you promote a SQL Server cross-Region read replica, the promoted replica behaves the same as other SQL Server DB instances, including the management of its options. For more information about option groups, see Working with option groups.

Synchronizing database users and objects with a SQL Server read replica

Any logins, custom server roles, SQL agent jobs, or other server-level objects that exist in the primary DB instance at the time of creating a read replica are expected to be present in the newly created read replica. However, any server-level objects that are created in the primary DB instance after the creation of the read replica will not be automatically replicated, and you must create them manually in the read replica.

The database users are automatically replicated from the primary DB instance to the read replica. As the read replica database is in read-only mode, the security identifier (SID) of the database user cannot be updated in the database. Therefore, when creating SQL logins in the read replica, it's essential to ensure that the SID of that login matches the SID of the corresponding SQL login in the primary DB instance. If you don't synchronize the SIDs of the SQL logins, they won't be able to access the database in the read replica. Windows Active Directory (AD) Authenticated Logins do not experience this issue because the SQL Server obtains the SID from the Active Directory.

To synchronize a SQL login from the primary DB instance to the read replica

1. Connect to the primary DB instance.
2. Create a new SQL login in the primary DB instance.

```
USE [master]
GO
CREATE LOGIN TestLogin1
WITH PASSWORD = 'REPLACE WITH PASSWORD';
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

3. Create a new database user for the SQL login in the database.

```
USE [REPLACE WITH YOUR DB NAME]
GO
CREATE USER TestLogin1 FOR LOGIN TestLogin1;
GO
```

4. Check the SID of the newly created SQL login in primary DB instance.

```
SELECT name, sid FROM sys.server_principals WHERE name =  'TestLogin1';
```

5. Connect to the read replica. Create the new SQL login.

```
CREATE LOGIN TestLogin1 WITH PASSWORD = 'REPLACE WITH PASSWORD', SID=REPLACE WITH
 sid FROM STEP #4;
```

Alternately, if you have access to the read replica database, you can fix the orphaned user as follows:

1. Connect to the read replica.
2. Identify the orphaned users in the database.

```
USE [REPLACE WITH YOUR DB NAME]
GO
EXEC sp_change_users_login 'Report';
GO
```

3. Create a new SQL login for the orphaned database user.

```
CREATE LOGIN TestLogin1 WITH PASSWORD = 'REPLACE WITH PASSWORD', SID=REPLACE WITH
 sid FROM STEP #2;
```

Example:

```
CREATE LOGIN TestLogin1 WITH PASSWORD = 'TestPa$$word#1',
 SID=0x1A2B3C4D5E6F7G8H9I0J1K2L3M4N5O6P;
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Troubleshooting a SQL Server read replica problem

You can monitor replication lag in Amazon CloudWatch by viewing the Amazon RDS `ReplicaLag` metric. For information about replication lag time, see Monitoring read replication.

If replication lag is too long, you can use the following query to get information about the lag.

```
SELECT AR.replica_server_name
     , DB_NAME (ARS.database_id) 'database_name'
     , AR.availability_mode_desc
     , ARS.synchronization_health_desc
     , ARS.last_hardened_lsn
     , ARS.last_redone_lsn
     , ARS.secondary_lag_seconds
```

```
FROM sys.dm_hadr_database_replica_states ARS
INNER JOIN sys.availability_replicas AR ON ARS.replica_id = AR.replica_id
--WHERE DB_NAME(ARS.database_id) = 'database_name'
ORDER BY AR.replica_server_name;
```

Multi-AZ deployments for Amazon RDS for Microsoft SQL Server

Multi-AZ deployments provide increased availability, data durability, and fault tolerance for DB instances. In the event of planned database maintenance or unplanned service disruption, Amazon RDS automatically fails over to the up-to-date secondary DB instance. This functionality lets database operations resume quickly without manual intervention. The primary and standby instances use the same endpoint, whose physical network address transitions to the secondary replica as part of the failover process. You don't have to reconfigure your application when a failover occurs.

Amazon RDS supports Multi-AZ deployments for Microsoft SQL Server by using either SQL Server Database Mirroring (DBM), Always On Availability Groups (AGs), or block level replication. Amazon RDS monitors and maintains the health of your Multi-AZ deployment. If problems occur, RDS automatically repairs unhealthy DB instances, reestablishes synchronization, and initiates failovers. Failover only occurs if the standby and primary are fully in sync. You don't have to manage anything.

When you set up SQL Server Multi-AZ, RDS automatically configures all databases on the instance to use DBM, AGs, or block level replication. Amazon RDS handles the primary, the witness, and the secondary DB instance for you when you configure DBM or AGs. For block level replication, RDS handles the primary and the secondary DB instances. Because configuration is automatic, RDS selects DBM, Always On AGs, or block level replication based on the version of SQL Server that you deploy.

Amazon RDS supports Multi-AZ with Always On AGs for the following SQL Server versions and editions:

- SQL Server 2022:
  - Standard Edition
  - Enterprise Edition
- SQL Server 2019:
  - Standard Edition 15.00.4073.23 and higher
  - Enterprise Edition
- SQL Server 2017:
  - Standard Edition 14.00.3401.7 and higher

  - Enterprise Edition 14.00.3049.1 and higher
- SQL Server 2016: Enterprise Edition 13.00.5216.0 and higher

Amazon RDS supports Multi-AZ with DBM for the following SQL Server versions and editions, except for the versions noted previously:

- SQL Server 2019: Standard Edition 15.00.4043.16
- SQL Server 2017: Standard and Enterprise Editions
- SQL Server 2016: Standard and Enterprise Editions

Amazon RDS supports Multi-AZ with block level replication for SQL Server 2022 Web Edition 16.00.4215.2 and above.

> **Note**
>
> Only new DB instances created with 16.00.4215.2 or higher support Multi-AZ deployments with block level replication. The following restrictions apply for existing SQL Server 2022 Web Edition instances:
>
> - For existing instances on version 16.00.4215.2, you must restore a snapshot to a new instance with the same or higher minor version to enable block level replication.
> - SQL Server 2022 Web instances with an older minor version can be upgraded to minor version 16.00.4215.2 or higher to enable block level replication.

You can use the following SQL query to determine whether your SQL Server DB instance is Single-AZ, Multi-AZ with DBM, or Multi-AZ with Always On AGs. This query does not apply for Multi-AZ deployments on SQL Server Web Edition.

```
SELECT CASE WHEN dm.mirroring_state_desc IS NOT NULL THEN 'Multi-AZ (Mirroring)'
    WHEN dhdrs.group_database_id IS NOT NULL THEN 'Multi-AZ (AlwaysOn)'
    ELSE 'Single-AZ'
    END 'high_availability'
FROM sys.databases sd
LEFT JOIN sys.database_mirroring dm ON sd.database_id = dm.database_id
LEFT JOIN sys.dm_hadr_database_replica_states dhdrs ON sd.database_id =
 dhdrs.database_id AND dhdrs.is_local = 1
```

```
WHERE DB_NAME(sd.database_id) = 'rdsadmin';
```

The output resembles the following:

```
high_availability
Multi-AZ (AlwaysOn)
```

Adding Multi-AZ to a Microsoft SQL Server DB instance

When you create a new SQL Server DB instance using the AWS Management Console, you can add Multi-AZ with Database Mirroring (DBM), Always On AGs or block level replication. You do so by choosing **Yes (Mirroring / Always On / Block Level Replication)** from **Multi-AZ deployment**. For more information, see Creating an Amazon RDS DB instance.

When you modify an existing SQL Server DB instance using the console, you can add Multi-AZ with DBM, AGs, or block level replication by choosing **Yes (Mirroring / Always On / Block Level Replication)** from **Multi-AZ deployment** on the **Modify DB instance** page. For more information, see Modifying an Amazon RDS DB instance.

> **Note**
>
> If your DB instance is running Database Mirroring (DBM)—not Always On Availability Groups (AGs)—you might need to disable in-memory optimization before you add Multi-AZ. Disable in-memory optimization with DBM before you add Multi-AZ if your DB instance runs SQL Server 2016 or 2017 Enterprise Edition and has in-memory optimization enabled. If your DB instance is running AGs or block level replication for SQL Server Web Editions, it doesn't require this step.

Removing Multi-AZ from a Microsoft SQL Server DB instance

When you modify an existing SQL Server DB instance using the AWS Management Console, you can remove Multi-AZ with DBM, AGs, or block level replication. You can do this by choosing **No (Mirroring / Always On / Block Level Replication)** from **Multi-AZ deployment** on the **Modify DB instance** page. For more information, see Modifying an Amazon RDS DB instance.

Microsoft SQL Server Multi-AZ deployment limitations, notes, and recommendations

The following are some limitations when working with Multi-AZ deployments on RDS for SQL Server DB instances:

- Cross-Region Multi-AZ isn't supported.
- Stopping an RDS for SQL Server DB instance in a multi-AZ deployment isn't supported.
- You can't configure the secondary DB instance to accept database read activity.
- Multi-AZ with Always On Availability Groups (AGs) supports in-memory optimization.
- Multi-AZ with Always On Availability Groups (AGs) doesn't support Kerberos authentication for the availability group listener. This is because the listener has no Service Principal Name (SPN).
- Multi-AZ with block level replication is currently only supported for SQL Server Web Edition instances.
- You can't rename a database on a SQL Server DB instance that is in a SQL Server Multi-AZ deployment. If you need to rename a database on such an instance, first turn off Multi-AZ for the DB instance, then rename the database. Finally, turn Multi-AZ back on for the DB instance.
- You can only restore Multi-AZ DB instances that are backed up using the full recovery model.
- Multi-AZ deployments have a limit of 10,000 SQL Server Agent jobs.

  If you need a higher limit, request an increase by contacting Support. Open the AWS Support Center page, sign in if necessary, and choose **Create case**. Choose **Service limit increase**. Complete and submit the form.
- You can't have an offline database on a SQL Server DB instance that is in a SQL Server Multi-AZ deployment.
- RDS for SQL Server doesn't replicate MSDB database permissions to the secondary instance. If you need these permissions on the secondary instance, you must recreate them manually.
- Volume metrics are not available for the secondary host of the instance using block level replication.

The following are some notes about working with Multi-AZ deployments on RDS for SQL Server DB instances:

- Amazon RDS exposes the Always On AGs availability group listener endpoint. The endpoint is visible in the console, and is returned by the `DescribeDBInstances` API operation as an entry in the endpoints field.
- Amazon RDS supports availability group multisubnet failovers.
- To use SQL Server Multi-AZ with a SQL Server DB instance in a virtual private cloud (VPC), first create a DB subnet group that has subnets in at least two distinct Availability Zones. Then assign the DB subnet group to the primary replica of the SQL Server DB instance.
- When a DB instance is modified to be a Multi-AZ deployment, during the modification it has a status of **modifying**. Amazon RDS creates the standby, and makes a backup of the primary DB instance. After the process is complete, the status of the primary DB instance becomes **available**.
- Multi-AZ deployments maintain all databases on the same node. If a database on the primary host fails over, all your SQL Server databases fail over as one atomic unit to your standby host. Amazon RDS provisions a new healthy host, and replaces the unhealthy host.
- Multi-AZ with DBM, AGs, or block level replication supports a single standby replica.
- Users, logins, and permissions are automatically replicated for you on the secondary. You don't need to recreate them. User-defined server roles are replicated in DB instances that use Always On AGs or block level replication for Multi-AZ deployments.
- In Multi-AZ deployments, RDS for SQL Server creates SQL Server logins to allow Always On AGs or Database Mirroring. RDS creates logins with the following pattern, `db_<dbiResourceId>_node1_login`, `db_<dbiResourceId>_node2_login`, and `db_<dbiResourceId>_witness_login`.
- RDS for SQL Server creates a SQL Server login to allow access to read replicas. RDS creates a login with the following pattern, `db_<readreplica_dbiResourceId>_node_login`.
- In Multi-AZ deployments, SQL Server Agent jobs are replicated from the primary host to the secondary host when the job replication feature is turned on. For more information, see Turning on SQL Server Agent job replication.
- You might observe elevated latencies compared to a standard DB instance deployment (in a single Availability Zone) because of the synchronous data replication.
- Failover times are affected by the time it takes to complete the recovery process. Large transactions increase the failover time.
- In SQL Server Multi-AZ deployments, reboot with failover reboots only the primary DB instance. After the failover, the primary DB instance becomes the new secondary DB instance. Parameters might not be updated for Multi-AZ instances. For reboot without failover, both the primary and

secondary DB instances reboot, and parameters are updated after the reboot. If the DB instance is unresponsive, we recommend reboot without failover.

The following are some recommendations for working with Multi-AZ deployments on RDS for Microsoft SQL Server DB instances:

- For databases used in production or preproduction, we recommend the following options:
  - Multi-AZ deployments for high availability
  - "Provisioned IOPS" for fast, consistent performance
  - "Memory optimized" rather than "General purpose"
- You can't select the Availability Zone (AZ) for the secondary instance, so when you deploy application hosts, take this into account. Your database might fail over to another AZ, and the application hosts might not be in the same AZ as the database. For this reason, we recommend that you balance your application hosts across all AZs in the given AWS Region.
- For best performance, don't enable Database Mirroring, Always On AGs, or block level replication during a large data load operation. If you want your data load to be as fast as possible, finish loading data before you convert your DB instance to a Multi-AZ deployment.
- Applications that access the SQL Server databases should have exception handling that catches connection errors. The following code sample shows a try/catch block that catches a communication error. In this example, the `break` statement exits the `while` loop if the connection is successful, but retries up to 10 times if an exception is thrown.

```
int RetryMaxAttempts = 10;
int RetryIntervalPeriodInSeconds = 1;
int iRetryCount = 0;
while (iRetryCount < RetryMaxAttempts)
{
   using (SqlConnection connection = new SqlConnection(DatabaseConnString))
   {
      using (SqlCommand command = connection.CreateCommand())
      {
         command.CommandText = "INSERT INTO SOME_TABLE VALUES ('SomeValue');";
         try
         {
            connection.Open();
            command.ExecuteNonQuery();
            break;
         }
```

```
         catch (Exception ex)
         {
            Logger(ex.Message);
            iRetryCount++;
         }
         finally {
            connection.Close();
         }
      }
   }
   Thread.Sleep(RetryIntervalPeriodInSeconds * 1000);
}
```

- Don't use the `Set Partner Off` command when working with Multi-AZ instances using DBM or AGs. This command is not supported on instances using block level replication. For example, don't do the following.

```
--Don't do this
ALTER DATABASE db1 SET PARTNER off
```

- Don't set the recovery mode to `simple`. For example, don't do the following.

```
--Don't do this
ALTER DATABASE db1 SET RECOVERY simple
```

- Don't use the DEFAULT_DATABASE parameter when creating new logins on Multi-AZ DB instances unless using block level replication for high availability, because these settings can't be applied to the standby mirror. For example, don't do the following.

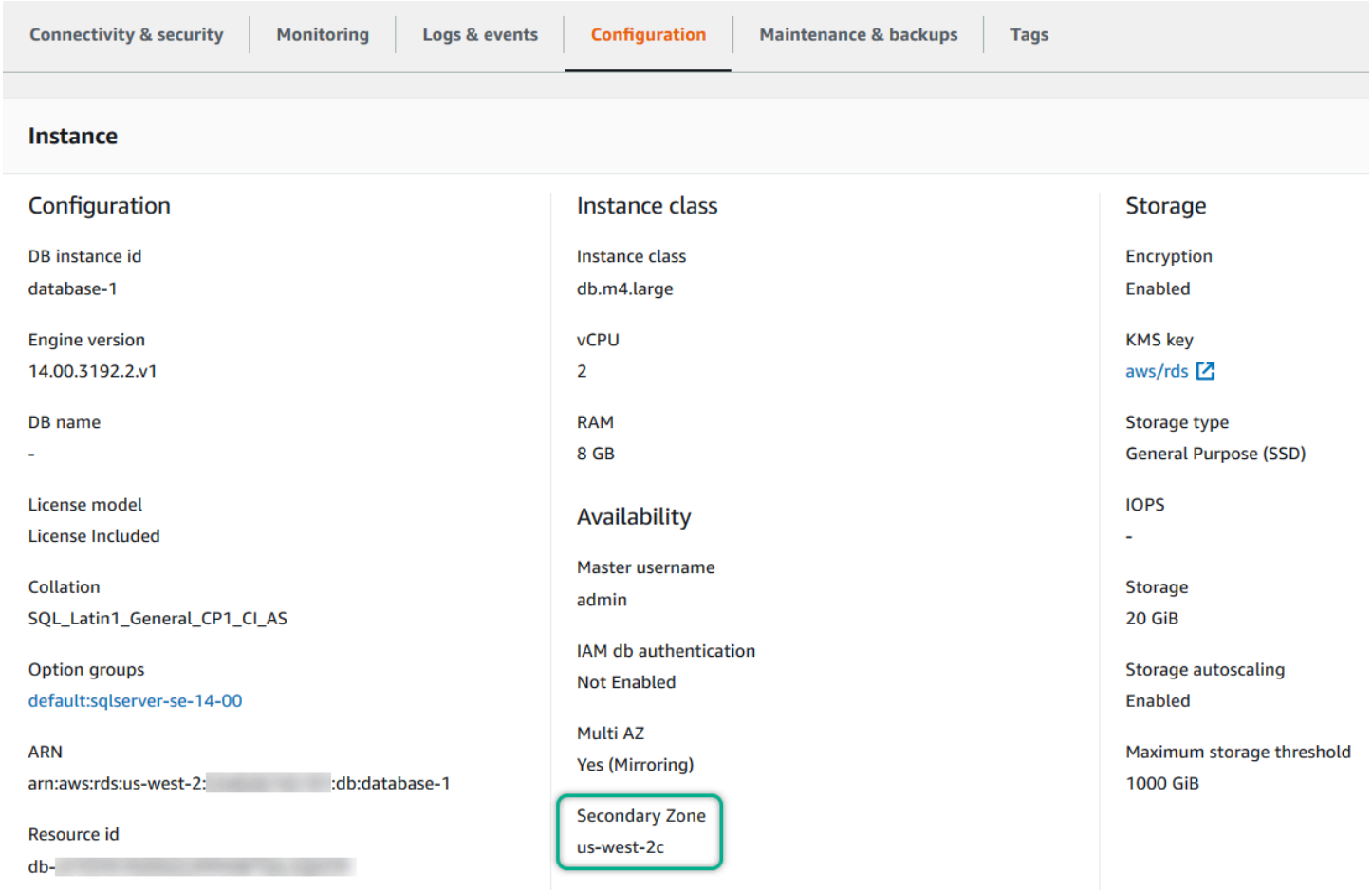
```
--Don't do this
CREATE LOGIN [test_dba] WITH PASSWORD=foo, DEFAULT_DATABASE=[db2]
```

  Also, don't do the following.

```
--Don't do this
ALTER LOGIN [test_dba] WITH DEFAULT_DATABASE=[db3]
```

Determining the location of the secondary

You can determine the location of the secondary replica by using the AWS Management Console. You need to know the location of the secondary if you are setting up your primary DB instance in a VPC.



You can also view the Availability Zone of the secondary using the AWS CLI command `describe-db-instances` or RDS API operation `DescribeDBInstances`. The output shows the secondary AZ where the standby mirror is located.

Migrating from Database Mirroring to Always On Availability Groups

In version 14.00.3049.1 of Microsoft SQL Server Enterprise Edition, Always On Availability Groups (AGs) are enabled by default.

To migrate from Database Mirroring (DBM) to AGs, first check your version. If you are using a DB instance with a version prior to Enterprise Edition 13.00.5216.0, modify the instance to patch it

to 13.00.5216.0 or later. If you are using a DB instance with a version prior to Enterprise Edition 14.00.3049.1, modify the instance to patch it to 14.00.3049.1 or later.

If you want to upgrade a mirrored DB instance to use AGs, run the upgrade first, modify the instance to remove Multi-AZ, and then modify it again to add Multi-AZ. This converts your instance to use Always On AGs.

Additional features for Microsoft SQL Server on Amazon RDS

In the following sections, you can find information about augmenting Amazon RDS instances running the Microsoft SQL Server DB engine.

Topics

- Using Password Policy for SQL Server logins on RDS for SQL Server
- Integrating an Amazon RDS for SQL Server DB instance with Amazon S3
- Using Database Mail on Amazon RDS for SQL Server
- Instance store support for the tempdb database on Amazon RDS for SQL Server
- Using extended events with Amazon RDS for Microsoft SQL Server
- Access to transaction log backups with RDS for SQL Server

Using Password Policy for SQL Server logins on RDS for SQL Server

Amazon RDS allows you to set the password policy for your Amazon RDS DB instance running Microsoft SQL Server. Use this to set complexity, length, and lockout requirements for logins that use SQL Server Authentication to authenticate to your DB instance.

Key terms

Login

In SQL Server, a server-level principal that can authenticate to a database instance is referred to as a **login**. Other database engines might refer to this principal as a *user*. In RDS for SQL Server, a login can authenticate using SQL Server Authentication or Windows Authentication.

SQL Server login

A login that uses a username and password to authenticate using SQL Server Authentication is a SQL Server login. The password policy you configure through DB parameters only applies to SQL Server logins.

Windows login

A login that is based on a Windows principal and authenticates using Windows Authentication is a Windows login. You can configure the password policy for your Windows logins in Active Directory. For more information, see Working with Active Directory with RDS for SQL Server.

Enabling and disabling policy for each login

Each SQL Server login has flags for `CHECK_POLICY` and `CHECK_EXPIRATION`. By default, new logins are created with `CHECK_POLICY` set to `ON` and `CHECK_EXPIRATION` set to `OFF`.

If `CHECK_POLICY` is enabled for a login, RDS for SQL Server validates the password against the complexity and minimum length requirements. Lockout policies also apply. An example T-SQL statement to enable `CHECK_POLICY` and `CHECK_EXPIRATION`:

```
ALTER LOGIN [master_user] WITH CHECK_POLICY = ON, CHECK_EXPIRATION = ON;
```

If `CHECK_EXPIRATION` is enabled, passwords are subject to password age policies. The T-SQL statement to check if `CHECK_POLICY` and `CHECK_EXPIRATION` are set:

```
SELECT name, is_policy_checked, is_expiration_checked FROM sys.sql_logins;
```

Password policy parameters

All password policy parameters are dynamic and do not require DB reboot to take effect. The following table lists the DB parameters you can set to modify the password policy for SQL Server logins:

DB parameter	Description	Allowed Values	Default Value	

rds.password_complexity_enabled	Password complexity requirements must be satisfied when creating or changing passwords for SQL Server logins. The following constraints must be met:
	• The password must include characters from three of the following categories:
	  • Latin lowercase letter (a through z)
	  • Latin uppercase letter (A through Z)
	  • Non-alpha numeric
		0,1	0	

DB parameter	Description	Allowed Values	Default Value	
	characters such as: exclamation point (!), dollar sign ($), number sign (#), or percent (%). • The password doesn't contain the account name of the user.			
rds.password_min_length	The minimum number of characters required in a password for a SQL Server login.	0-14	0	
rds.password_min_age	The minimum number of days a SQL Server login password must be used before the user can change it. Passwords can be changed immediately when set to 0.	0-998	0	

DB parameter	Description	Allowed Values	Default Value	
rds.password_max_age	The maximum number of days a SQL Server login password can be used after which the user is required to change it. Passwords never expire when set to 0.	0-999	42	
rds.password_lockout_threshold	The number of consecutive failed login attempts that cause a SQL Server login to become locked out.	0-999	0	
rds.password_lockout_duration	The number of minutes a locked out SQL Server login must wait before being unlocked.	1-60	10	

DB parameter	Description	Allowed Values	Default Value	
rds.password_lockout_reset_counter_after	The number of minutes that must elapse after a failed login attempt before the failed login attempt counter is reset to 0.	1-60	10	

> **Note**
>
> For more information about SQL Server password policy, see Password Policy.
> The password complexity and minimum length policies also apply to DB users in contained databases. For more information, see Contained Databases.

The following constraints apply to the password policy parameters:

- The `rds.password_min_age` parameter must be less than `rds.password_max_age` parameter, unless `rds.password_max_age` is set to 0
- The `rds.password_lockout_reset_counter_after` parameter must be less than or equal to the `rds.password_lockout_duration` parameter.
- If `rds.password_lockout_threshold` is set to 0, `rds.password_lockout_duration` and `rds.password_lockout_reset_counter_after` do not apply.

Considerations for existing logins

After modifying the password policy on an instance, existing passwords for logins are **not** retroactively evaluated against the new password complexity and length requirements. Only new passwords are validated against the new policy.

SQL Server **does** evaluate existing passwords for age requirements.

It is possible for passwords to expire immediately once a password policy is modified. For example, if a login has `CHECK_EXPIRATION` enabled and its password was last changed 100 days ago and you set the `rds.password_max_age` parameter to 5 days, the password immediately expires and the login needs to change their password at their next attempt to log in.

> **Note**
>
> RDS for SQL Server doesn't support password history policies. History policies prevent logins from reusing previously used passwords.

Considerations for Multi-AZ deployments

The failed login attempt counter and lockout state for Multi-AZ instances does not replicate between nodes. In the event of a login being locked out when a Multi-AZ instance fails over, it is possible for the login to be already unlocked on the new node.

Password considerations for the master login

When you create an RDS for SQL Server DB instance, the master user password is not evaluated against the password policy. A new master password is also not evaluated against the password when performing operations to the master user, specifically when setting `MasterUserPassword` in the `ModifyDBInstance` command. In both cases, you can set a password for the master user that does not satisfy your password policy, and the operation still succeeds. If the policy is not satisfied, RDS attempts to raise an RDS event, with the recommendation to set a strong password. Take care to only use strong passwords for the master user.

RDS attempts to generate the following event messages when the master user password does not meet the password policy requirements:

- The master user was created, but the password doesn't meet the minimum length requirement of your password policy. Consider using a stronger password.
- The master user was created, but the password doesn't meet the complexity requirement of your password policy. Consider using a stronger password.
- The master user password was reset, but the password doesn't meet the minimum length requirement of your password policy. Consider using a stronger password.
- The master user password was reset, but the password doesn't meet the complexity requirement of your password policy. Consider using a stronger password.

By default, the master user is created with `CHECK_POLICY` and `CHECK_EXPIRATION` set to `OFF`. To apply the password policy to the master user, you must manually enable these flags for the master user after DB instance creation. After you enable these flags, modify the master user password directly in SQL Server (eg. via T-SQL statements or SSMS) to validate the new password against the password policy.

> **Note**
>
> If the master user gets locked out, you can unlock the user by resetting the master user password using the `ModifyDBInstance` command.

Modifying the master user password

You can modify the master user password by using the ModifyDBInstance command.

> **Note**
>
> When you reset the master user password, RDS resets various permissions for the master user and the master user might lose certain permissions. Resetting the master user password also unlocks the master user, if it was locked out.

RDS validates the new master user password and attempts to emit an RDS event if the password does not satisfy the policy. RDS sets the password even if it does not satisfy the password policy.

Integrating an Amazon RDS for SQL Server DB instance with Amazon S3

You can transfer files between a DB instance running Amazon RDS for SQL Server and an Amazon S3 bucket. By doing this, you can use Amazon S3 with SQL Server features such as BULK INSERT. For example, you can download .csv, .xml, .txt, and other files from Amazon S3 to the DB instance host and import the data from `D:\S3\` into the database. All files are stored in `D:\S3\` on the DB instance.

The following limitations apply:

> **Note**
>
> Traffic between the RDS host and S3 routes through VPC endpoints in RDS internal VPCs for all SQL Server features that use S3. This traffic doesn't use the RDS instance endpoint ENI. S3 bucket policies can't restrict RDS traffic by networking conditions.

- Files in the `D:\S3` folder are deleted on the standby replica after a failover on Multi-AZ instances. For more information, see Multi-AZ limitations for S3 integration.
- The DB instance and the S3 bucket must be in the same AWS Region.
- If you run more than one S3 integration task at a time, the tasks run sequentially, not in parallel.

  > **Note**
  >
  > S3 integration tasks share the same queue as native backup and restore tasks. At maximum, you can have only two tasks in progress at any time in this queue. Therefore, two running native backup and restore tasks will block any S3 integration tasks.

- You must re-enable the S3 integration feature on restored instances. S3 integration isn't propagated from the source instance to the restored instance. Files in `D:\S3` are deleted on a restored instance.
- Downloading to the DB instance is limited to 100 files. In other words, there can't be more than 100 files in `D:\S3\`.
- Only files without file extensions or with the following file extensions are supported for download: .abf, .asdatabase, .bcp, .configsettings, .csv, .dat, .deploymentoptions, .deploymenttargets, .fr and .xmla.

- The S3 bucket must have the same owner as the related AWS Identity and Access Management (IAM) role. Therefore, cross-account S3 integration isn't supported.
- The S3 bucket can't be open to the public.
- The file size for uploads from RDS to S3 is limited to 50 GB per file.
- The file size for downloads from S3 to RDS is limited to the maximum supported by S3.

Topics

- Prerequisites for integrating RDS for SQL Server with S3
- Enabling RDS for SQL Server integration with S3
- Transferring files between RDS for SQL Server and Amazon S3
- Listing files on the RDS DB instance
- Deleting files on the RDS DB instance
- Monitoring the status of a file transfer task
- Canceling a task
- Multi-AZ limitations for S3 integration
- Disabling RDS for SQL Server integration with S3

For more information on working with files in Amazon S3, see Getting started with Amazon Simple Storage Service.

Prerequisites for integrating RDS for SQL Server with S3

Before you begin, find or create the S3 bucket that you want to use. Also, add permissions so that the RDS DB instance can access the S3 bucket. To configure this access, you create both an IAM policy and an IAM role.

Console

To create an IAM policy for access to Amazon S3

1. In the IAM Management Console, choose **Policies** in the navigation pane.
2. Create a new policy, and use the **Visual editor** tab for the following steps.
3. For **Service**, enter **S3** and then choose the **S3** service.

4. For **Actions**, choose the following to grant the access that your DB instance requires:

   - `ListAllMyBuckets` – required
   - `ListBucket` – required
   - `GetBucketAcl` – required
   - `GetBucketLocation` – required
   - `GetObject` – required for downloading files from S3 to `D:\S3\`
   - `PutObject` – required for uploading files from `D:\S3\` to S3
   - `ListMultipartUploadParts` – required for uploading files from `D:\S3\` to S3
   - `AbortMultipartUpload` – required for uploading files from `D:\S3\` to S3

5. For **Resources**, the options that display depend on which actions you choose in the previous step. You might see options for **bucket**, **object**, or both. For each of these, add the appropriate Amazon Resource Name (ARN).

   For **bucket**, add the ARN for the bucket that you want to use. For example, if your bucket is named *`amzn-s3-demo-bucket`*, set the ARN to `arn:aws:s3:::`*`amzn-s3-demo-bucket`*.

   For **object**, enter the ARN for the bucket and then choose one of the following:

   - To grant access to all files in the specified bucket, choose **Any** for both **Bucket name** and **Object name**.
   - To grant access to specific files or folders in the bucket, provide ARNs for the specific buckets and objects that you want SQL Server to access.

6. Follow the instructions in the console until you finish creating the policy.

   The preceding is an abbreviated guide to setting up a policy. For more detailed instructions on creating IAM policies, see Creating IAM policies in the *IAM User Guide.*

To create an IAM role that uses the IAM policy from the previous procedure

1. In the IAM Management Console, choose **Roles** in the navigation pane.
2. Create a new IAM role, and choose the following options as they appear in the console:

   - **AWS service**
   - **RDS**
   - **RDS – Add Role to Database**

Then choose **Next:Permissions** at the bottom.

3. For **Attach permissions policies**, enter the name of the IAM policy that you previously created. Then choose the policy from the list.
4. Follow the instructions in the console until you finish creating the role.

   The preceding is an abbreviated guide to setting up a role. If you want more detailed instructions on creating roles, see IAM roles in the *IAM User Guide.*

AWS CLI

To grant Amazon RDS access to an Amazon S3 bucket, use the following process:

1. Create an IAM policy that grants Amazon RDS access to an S3 bucket.
2. Create an IAM role that Amazon RDS can assume on your behalf to access your S3 buckets.

   For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.
3. Attach the IAM policy that you created to the IAM role that you created.

To create the IAM policy

Include the appropriate actions to grant the access your DB instance requires:

- `ListAllMyBuckets` – required
- `ListBucket` – required
- `GetBucketAcl` – required
- `GetBucketLocation` – required
- `GetObject` – required for downloading files from S3 to `D:\S3\`
- `PutObject` – required for uploading files from `D:\S3\` to S3
- `ListMultipartUploadParts` – required for uploading files from `D:\S3\` to S3
- `AbortMultipartUpload` – required for uploading files from `D:\S3\` to S3

1. The following AWS CLI command creates an IAM policy named `rds-s3-integration-policy` with these options. It grants access to a bucket named *`amzn-s3-demo-bucket`*.

Example

For Linux, macOS, or Unix:

```
aws iam create-policy \
  --policy-name rds-s3-integration-policy \
  --policy-document '{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": "s3:ListAllMyBuckets",
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "s3:ListBucket",
                    "s3:GetBucketAcl",
                    "s3:GetBucketLocation"
                ],
                "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "s3:GetObject",
                    "s3:PutObject",
                    "s3:ListMultipartUploadParts",
                    "s3:AbortMultipartUpload"
                ],
                "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/key_prefix/*"
            }
        ]
    }'
```

For Windows:

Make sure to change the line endings to the ones supported by your interface (^ instead of \). Also, in Windows, you must escape all double quotes with a \. To avoid the need to escape the quotes in the JSON, you can save it to a file instead and pass that in as a parameter.

First, create the `policy.json` file with the following permission policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:ListAllMyBuckets",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketACL",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/key_prefix/*"
        }
    ]
}
```

Then use the following command to create the policy:

```
aws iam create-policy ^
     --policy-name rds-s3-integration-policy ^
     --policy-document file://file_path/assume_role_policy.json
```

2. After the policy is created, note the Amazon Resource Name (ARN) of the policy. You need the ARN for a later step.

To create the IAM role

- The following AWS CLI command creates the `rds-s3-integration-role` IAM role for this purpose.

  Example

  For Linux, macOS, or Unix:

```
aws iam create-role \
    --role-name rds-s3-integration-role \
    --assume-role-policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
             "Service": "rds.amazonaws.com"
           },
          "Action": "sts:AssumeRole"
        }
      ]
    }'
```

  For Windows:

  Make sure to change the line endings to the ones supported by your interface (^ instead of \). Also, in Windows, you must escape all double quotes with a \. To avoid the need to escape the quotes in the JSON, you can save it to a file instead and pass that in as a parameter.

  First, create the `assume_role_policy.json` file with the following policy:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Then use the following command to create the IAM role:

```
aws iam create-role ^
     --role-name rds-s3-integration-role ^
     --assume-role-policy-document file://file_path/assume_role_policy.json
```

Example of using the global condition context key to create the IAM role

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based policies to limit the service's permissions to a specific resource. This is the most effective way to protect against the confused deputy problem.

You might use both global condition context keys and have the `aws:SourceArn` value contain the account ID. In this case, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

- Use `aws:SourceArn` if you want cross-service access for a single resource.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

In the policy, make sure to use the `aws:SourceArn` global condition context key with the full Amazon Resource Name (ARN) of the resources accessing the role. For S3 integration, make sure to include the DB instance ARNs, as shown in the following example.

For Linux, macOS, or Unix:

```
aws iam create-role \
    --role-name rds-s3-integration-role \
    --assume-role-policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
             "Service": "rds.amazonaws.com"
           },
          "Action": "sts:AssumeRole",
                "Condition": {
                    "StringEquals": {

 "aws:SourceArn":"arn:aws:rds:Region:my_account_ID:db:db_instance_identifier"
                    }
                }
        }
      ]
    }'
```

For Windows:

Add the global condition context key to `assume_role_policy.json`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "rds.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
```

```
            "aws:SourceArn":"arn:aws:rds:Region:my_account_ID:db:db_instance_identifier"
                    }
                }
            }
        ]
}
```

To attach the IAM policy to the IAM role

- The following AWS CLI command attaches the policy to the role named `rds-s3-integration-role`. Replace *your-policy-arn* with the policy ARN that you noted in a previous step.

  Example

  For Linux, macOS, or Unix:

  ```
  aws iam attach-role-policy \
      --policy-arn your-policy-arn \
      --role-name rds-s3-integration-role
  ```

  For Windows:

  ```
  aws iam attach-role-policy ^
      --policy-arn your-policy-arn ^
      --role-name rds-s3-integration-role
  ```

Enabling RDS for SQL Server integration with S3

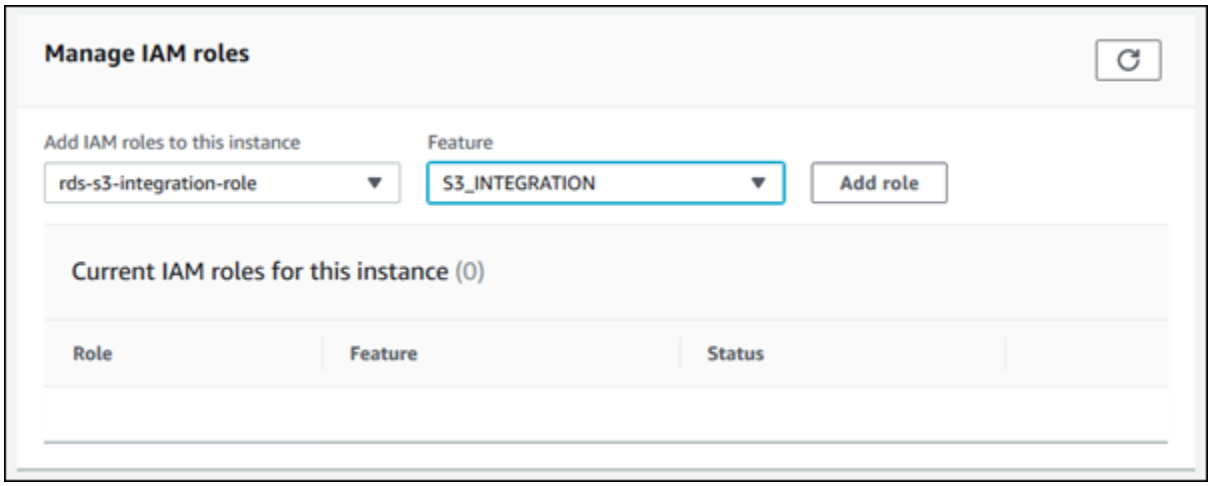
In the following section, you can find how to enable Amazon S3 integration with Amazon RDS for SQL Server. To work with S3 integration, your DB instance must be associated with the IAM role that you previously created before you use the `S3_INTEGRATION` feature-name parameter.

> **Note**
>
> To add an IAM role to a DB instance, the status of the DB instance must be **available**.

Console

To associate your IAM role with your DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose the RDS for SQL Server DB instance name to display its details.
3. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the IAM role to add for **Add IAM roles to this instance**.
4. For **Feature**, choose **S3_INTEGRATION**.



5. Choose **Add role**.

AWS CLI

To add the IAM role to the RDS for SQL Server DB instance

- The following AWS CLI command adds your IAM role to an RDS for SQL Server DB instance named *mydbinstance*.

  Example

  For Linux, macOS, or Unix:

```
aws rds add-role-to-db-instance \
    --db-instance-identifier mydbinstance \
    --feature-name S3_INTEGRATION \
    --role-arn your-role-arn
```

For Windows:

```
aws rds add-role-to-db-instance ^
    --db-instance-identifier mydbinstance ^
    --feature-name S3_INTEGRATION ^
    --role-arn your-role-arn
```

Replace *your-role-arn* with the role ARN that you noted in a previous step. `S3_INTEGRATION` must be specified for the `--feature-name` option.

Transferring files between RDS for SQL Server and Amazon S3

You can use Amazon RDS stored procedures to download and upload files between Amazon S3 and your RDS DB instance. You can also use Amazon RDS stored procedures to list and delete files on the RDS instance.

The files that you download from and upload to S3 are stored in the `D:\S3` folder. This is the only folder that you can use to access your files. You can organize your files into subfolders, which are created for you when you include the destination folder during download.

Some of the stored procedures require that you provide an Amazon Resource Name (ARN) to your S3 bucket and file. The format for your ARN is `arn:aws:s3:::`*amzn-s3-demo-bucket*`/file_name`. Amazon S3 doesn't require an account number or AWS Region in ARNs.

S3 integration tasks run sequentially and share the same queue as native backup and restore tasks. At maximum, you can have only two tasks in progress at any time in this queue. It can take up to five minutes for the task to begin processing.

Downloading files from an Amazon S3 bucket to a SQL Server DB instance

To download files from an S3 bucket to an RDS for SQL Server DB instance, use the Amazon RDS stored procedure `msdb.dbo.rds_download_from_s3` with the following parameters.

Parameter name	Data type	Default	Required	Description
`@s3_arn_of_file`	NVARCHAR	–	Required	The S3 ARN of the file to download, for example: `arn:aws:s3:::` *amzn-*

Parameter name	Data type	Default	Required	Description
				s3-demo-bucket / mydata.csv
@rds_file_path	NVARCHAR	–	Optional	The file path for the RDS instance. If not specified, the file path is D:\S3*<filename in s3>*. RDS supports absolute paths and relative paths. If you want to create a subfolder, include it in the file path.
@overwrite_file	INT	0	Optional	Overwrite the existing file: 0 = Don't overwrite 1 = Overwrite

You can download files without a file extension and files with the following file extensions: .bcp, .csv, .dat, .fmt, .info, .lst, .tbl, .txt, and .xml.

Note

Files with the .ispac file extension are supported for download when SQL Server Integration Services is enabled. For more information on enabling SSIS, see SQL Server Integration Services.
Files with the following file extensions are supported for download when SQL Server Analysis Services is
enabled: .abf, .asdatabase, .configsettings, .deploymentoptions, .deploymenttargets, and .xmla. For more information on enabling SSAS, see SQL Server Analysis Services.

The following example shows the stored procedure to download files from S3.

```
exec msdb.dbo.rds_download_from_s3
     @s3_arn_of_file='arn:aws:s3:::amzn-s3-demo-bucket/bulk_data.csv',
     @rds_file_path='D:\S3\seed_data\data.csv',
     @overwrite_file=1;
```

The example `rds_download_from_s3` operation creates a folder named `seed_data` in `D:\S3\`, if the folder doesn't exist yet. Then the example downloads the source file `bulk_data.csv` from S3 to a new file named `data.csv` on the DB instance. If the file previously existed, it's overwritten because the `@overwrite_file` parameter is set to 1.

Uploading files from a SQL Server DB instance to an Amazon S3 bucket

To upload files from an RDS for SQL Server DB instance to an S3 bucket, use the Amazon RDS stored procedure `msdb.dbo.rds_upload_to_s3` with the following parameters.

Parameter name	Data type	Default	Required	Description
`@s3_arn_of_file`	NVARCHAR	–	Required	The S3 ARN of the file to be created in S3, for example: `arn:aws:s3:::` *`amzn-s3-demo-bucket`* `/mydata.csv`
`@rds_file_path`	NVARCHAR	–	Required	The file path of the file to upload to S3. Absolute and relative paths are supported.
`@overwrite_file`	INT	–	Optional	Overwrite the existing file: 0 = Don't overwrite 1 = Overwrite

The following example uploads the file named `data.csv` from the specified location in `D:\S3\seed_data\` to a file `new_data.csv` in the S3 bucket specified by the ARN.

```
exec msdb.dbo.rds_upload_to_s3
  @rds_file_path='D:\S3\seed_data\data.csv',
  @s3_arn_of_file='arn:aws:s3:::amzn-s3-demo-bucket/new_data.csv',
  @overwrite_file=1;
```

If the file previously existed in S3, it's overwritten because the @overwrite_file parameter is set to 1.

Listing files on the RDS DB instance

To list the files available on the DB instance, use both a stored procedure and a function. First, run the following stored procedure to gather file details from the files in `D:\S3\`.

```
exec msdb.dbo.rds_gather_file_details;
```

The stored procedure returns the ID of the task. Like other tasks, this stored procedure runs asynchronously. As soon as the status of the task is SUCCESS, you can use the task ID in the `rds_fn_list_file_details` function to list the existing files and directories in D:\S3\, as shown following.

```
SELECT * FROM msdb.dbo.rds_fn_list_file_details(TASK_ID);
```

The `rds_fn_list_file_details` function returns a table with the following columns.

Output parameter	Description
`filepath`	Absolute path of the file (for example, `D:\S3\mydata.csv`)
`size_in_bytes`	File size (in bytes)
`last_modified_utc`	Last modification date and time in UTC format
`is_directory`	Option that indicates whether the item is a directory (`true`/`false`)

Deleting files on the RDS DB instance

To delete the files available on the DB instance, use the Amazon RDS stored procedure `msdb.dbo.rds_delete_from_filesystem` with the following parameters.

Parameter name	Data type	Default	Required	Description
`@rds_file_path`	NVARCHAR	–	Required	The file path of the file to delete. Absolute and relative paths are supported.
`@force_delete`	INT	0	Optional	To delete a directory, this flag must be included and set to 1. 1 = delete a directory This parameter is ignored if you are deleting a file.

To delete a directory, the `@rds_file_path` must end with a backslash (\) and `@force_delete` must be set to 1.

The following example deletes the file `D:\S3\delete_me.txt`.

```
exec msdb.dbo.rds_delete_from_filesystem
    @rds_file_path='D:\S3\delete_me.txt';
```

The following example deletes the directory `D:\S3\example_folder\`.

```
exec msdb.dbo.rds_delete_from_filesystem
    @rds_file_path='D:\S3\example_folder\',
    @force_delete=1;
```

Monitoring the status of a file transfer task

To track the status of your S3 integration task, call the `rds_fn_task_status` function. It takes two parameters. The first parameter should always be `NULL` because it doesn't apply to S3 integration. The second parameter accepts a task ID.

To see a list of all tasks, set the first parameter to `NULL` and the second parameter to `0`, as shown in the following example.

```
SELECT * FROM msdb.dbo.rds_fn_task_status(NULL,0);
```

To get a specific task, set the first parameter to `NULL` and the second parameter to the task ID, as shown in the following example.

```
SELECT * FROM msdb.dbo.rds_fn_task_status(NULL,42);
```

The `rds_fn_task_status` function returns the following information.

Output parameter	Description
`task_id`	The ID of the task.
`task_type`	For S3 integration, tasks can have the following task types: • `DOWNLOAD_FROM_S3` • `UPLOAD_TO_S3` • `LIST_FILES_ON_DISK` • `DELETE_FILES_ON_DISK`
`database_name`	Not applicable to S3 integration tasks.
`% complete`	The progress of the task as a percentage.
`duration(mins)`	The amount of time spent on the task, in minutes.
`lifecycle`	The status of the task. Possible statuses are the following:

Output parameter	Description
	• CREATED – After you call one of the S3 integration stored procedures, a task is created and the status is set to CREATED. • IN_PROGRESS – After a task starts, the status is set to IN_PROGRESS . It can take up to five minutes for the status to change from CREATED to IN_PROGRESS . • SUCCESS – After a task completes, the status is set to SUCCESS. • ERROR – If a task fails, the status is set to ERROR. For more information about the error, see the task_info column. • CANCEL_REQUESTED – After you call rds_cancel_task , the status of the task is set to CANCEL_REQUESTED . • CANCELLED – After a task is successfully canceled, the status of the task is set to CANCELLED .
task_info	Additional information about the task. If an error occurs during processing, this column contains information about the error.
last_updated	The date and time that the task status was last updated.
created_at	The date and time that the task was created.
S3_object_arn	The ARN of the S3 object downloaded from or uploaded to.
overwrite_S3_backup_file	Not applicable to S3 integration tasks.
KMS_master_key_arn	Not applicable to S3 integration tasks.

Output parameter	Description
`filepath`	The file path on the RDS DB instance.
`overwrite_file`	An option that indicates if an existing file is overwritten.
`task_metadata`	Not applicable to S3 integration tasks.

Canceling a task

To cancel S3 integration tasks, use the `msdb.dbo.rds_cancel_task` stored procedure with the `task_id` parameter. Delete and list tasks that are in progress can't be cancelled. The following example shows a request to cancel a task.

```
exec msdb.dbo.rds_cancel_task @task_id = 1234;
```

To get an overview of all tasks and their task IDs, use the `rds_fn_task_status` function as described in Monitoring the status of a file transfer task.

Multi-AZ limitations for S3 integration

On Multi-AZ instances, files in the `D:\S3` folder are deleted on the standby replica after a failover. A failover can be planned, for example, during DB instance modifications such as changing the instance class or upgrading the engine version. Or a failover can be unplanned, during an outage of the primary.

> **Note**
>
> We don't recommend using the `D:\S3` folder for file storage. The best practice is to upload created files to Amazon S3 to make them durable, and download files when you need to import data.

To determine the last failover time, you can use the `msdb.dbo.rds_failover_time` stored procedure. For more information, see Determining the last failover time for Amazon RDS for SQL Server.

Example of no recent failover

This example shows the output when there is no recent failover in the error logs. No failover has happened since 2020-04-29 23:59:00.01.

Therefore, all files downloaded after that time that haven't been deleted using the `rds_delete_from_filesystem` stored procedure are still accessible on the current host. Files downloaded before that time might also be available.

errorlog_available_from	recent_failover_time
2020-04-29 23:59:00.0100000	null

Example of recent failover

This example shows the output when there is a failover in the error logs. The most recent failover was at 2020-05-05 18:57:51.89.

All files downloaded after that time that haven't been deleted using the `rds_delete_from_filesystem` stored procedure are still accessible on the current host.

errorlog_available_from	recent_failover_time
2020-04-29 23:59:00.0100000	2020-05-05 18:57:51.8900000

Disabling RDS for SQL Server integration with S3

Following, you can find how to disable Amazon S3 integration with Amazon RDS for SQL Server. Files in `D:\S3\` aren't deleted when disabling S3 integration.

> **Note**
>
> To remove an IAM role from a DB instance, the status of the DB instance must be `available`.

Console

To disassociate your IAM role from your DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose the RDS for SQL Server DB instance name to display its details.
3. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the IAM role to remove.
4. Choose **Delete**.

AWS CLI

To remove the IAM role from the RDS for SQL Server DB instance

- The following AWS CLI command removes the IAM role from a RDS for SQL Server DB instance named *mydbinstance*.

  Example

  For Linux, macOS, or Unix:

```
aws rds remove-role-from-db-instance \
    --db-instance-identifier mydbinstance \
    --feature-name S3_INTEGRATION \
    --role-arn your-role-arn
```

  For Windows:

```
aws rds remove-role-from-db-instance ^
    --db-instance-identifier mydbinstance ^
    --feature-name S3_INTEGRATION ^
    --role-arn your-role-arn
```

  Replace *your-role-arn* with the appropriate IAM role ARN for the `--feature-name` option.

Using Database Mail on Amazon RDS for SQL Server

You can use Database Mail to send email messages to users from your Amazon RDS on SQL Server database instance. The messages can contain files and query results. Database Mail includes the following components:

- **Configuration and security objects** – These objects create profiles and accounts, and are stored in the `msdb` database.
- **Messaging objects** – These objects include the sp_send_dbmail stored procedure used to send messages, and data structures that hold information about messages. They're stored in the `msdb` database.
- **Logging and auditing objects** – Database Mail writes logging information to the `msdb` database and the Microsoft Windows application event log.
- **Database Mail executable** – `DatabaseMail.exe` reads from a queue in the `msdb` database and sends email messages.

RDS supports Database Mail for all SQL Server versions on the Web, Standard, and Enterprise Editions.

Limitations

The following limitations apply to using Database Mail on your SQL Server DB instance:

- Database Mail isn't supported for SQL Server Express Edition.
- Modifying Database Mail configuration parameters isn't supported. To see the preset (default) values, use the sysmail_help_configure_sp stored procedure.
- File attachments aren't fully supported. For more information, see Working with file attachments.
- The maximum file attachment size is 1 MB.
- Database Mail requires additional configuration on Multi-AZ DB instances. For more information, see Considerations for Multi-AZ deployments.
- Configuring SQL Server Agent to send email messages to predefined operators isn't supported.

Enabling Database Mail

Use the following process to enable Database Mail for your DB instance:

1. Create a new parameter group.
2. Modify the parameter group to set the `database mail xps` parameter to 1.
3. Associate the parameter group with the DB instance.

Creating the parameter group for Database Mail

Create a parameter group for the `database mail xps` parameter that corresponds to the SQL Server edition and version of your DB instance.

> **Note**
>
> You can also modify an existing parameter group. Follow the procedure in Modifying the parameter that enables Database Mail.

Console

The following example creates a parameter group for SQL Server Standard Edition 2016.

To create the parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
4. In the **Create parameter group** pane, do the following:
   a. For **Parameter group family**, choose **sqlserver-se-13.0**.
   b. For **Group name**, enter an identifier for the parameter group, such as **`dbmail-sqlserver-se-13`**.
   c. For **Description**, enter **`Database Mail XPs`**.
5. Choose **Create**.

CLI

The following example creates a parameter group for SQL Server Standard Edition 2016.

To create the parameter group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name dbmail-sqlserver-se-13 \
    --db-parameter-group-family "sqlserver-se-13.0" \
    --description "Database Mail XPs"
```

  For Windows:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name dbmail-sqlserver-se-13 ^
    --db-parameter-group-family "sqlserver-se-13.0" ^
    --description "Database Mail XPs"
```

Modifying the parameter that enables Database Mail

Modify the `database mail xps` parameter in the parameter group that corresponds to the SQL Server edition and version of your DB instance.

To enable Database Mail, set the `database mail xps` parameter to 1.

Console

The following example modifies the parameter group that you created for SQL Server Standard Edition 2016.

To modify the parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose the parameter group, such as **dbmail-sqlserver-se-13**.
4. Under **Parameters**, filter the parameter list for **`mail`**.
5. Choose **database mail xps**.

6. Choose **Edit parameters**.
7. Enter **1**.
8. Choose **Save changes**.

CLI

The following example modifies the parameter group that you created for SQL Server Standard Edition 2016.

To modify the parameter group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name dbmail-sqlserver-se-13 \
    --parameters "ParameterName='database mail
 xps',ParameterValue=1,ApplyMethod=immediate"
```

  For Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name dbmail-sqlserver-se-13 ^
    --parameters "ParameterName='database mail
 xps',ParameterValue=1,ApplyMethod=immediate"
```

Associating the parameter group with the DB instance

You can use the AWS Management Console or the AWS CLI to associate the Database Mail parameter group with the DB instance.

Console

You can associate the Database Mail parameter group with a new or existing DB instance.

- For a new DB instance, associate it when you launch the instance. For more information, see Creating an Amazon RDS DB instance.

- For an existing DB instance, associate it by modifying the instance. For more information, see Modifying an Amazon RDS DB instance.

CLI

You can associate the Database Mail parameter group with a new or existing DB instance.

To create a DB instance with the Database Mail parameter group

- Specify the same DB engine type and major version as you used when creating the parameter group.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.m5.2xlarge \
    --engine sqlserver-se \
    --engine-version 13.00.5426.0.v1 \
    --allocated-storage 100 \
    --manage-master-user-password \
    --master-username admin \
    --storage-type gp2 \
    --license-model li
    --db-parameter-group-name dbmail-sqlserver-se-13
```

  For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-se ^
    --engine-version 13.00.5426.0.v1 ^
    --allocated-storage 100 ^
    --manage-master-user-password ^
    --master-username admin ^
    --storage-type gp2 ^
    --license-model li ^
    --db-parameter-group-name dbmail-sqlserver-se-13
```

To modify a DB instance and associate the Database Mail parameter group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-parameter-group-name dbmail-sqlserver-se-13 \
    --apply-immediately
```

  For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-parameter-group-name dbmail-sqlserver-se-13 ^
    --apply-immediately
```

Configuring Database Mail

You perform the following tasks to configure Database Mail:

1. Create the Database Mail profile.
2. Create the Database Mail account.
3. Add the Database Mail account to the Database Mail profile.
4. Add users to the Database Mail profile.

> **Note**
>
> To configure Database Mail, make sure that you have `execute` permission on the stored procedures in the `msdb` database.

Creating the Database Mail profile

To create the Database Mail profile, you use the sysmail_add_profile_sp stored procedure. The following example creates a profile named `Notifications`.

To create the profile

- Use the following SQL statement.

```
USE msdb
GO

EXECUTE msdb.dbo.sysmail_add_profile_sp
    @profile_name         = 'Notifications',
    @description          = 'Profile used for sending outgoing notifications using
 Amazon SES.';
GO
```

Creating the Database Mail account

To create the Database Mail account, you use the sysmail_add_account_sp stored procedure. The following example creates an account named SES on an RDS for SQL Server DB instance in a private VPC, using Amazon Simple Email Service.

Using Amazon SES requires the following parameters:

- `@email_address` – An Amazon SES verified identity. For more information, see Verified identities in Amazon SES.
- `@mailserver_name` – An Amazon SES SMTP endpoint. For more information, see Connecting to an Amazon SES SMTP endpoint.
- `@username` – An Amazon SES SMTP user name. For more information, see Obtaining Amazon SES SMTP credentials.

  Don't use an AWS Identity and Access Management user name.
- `@password` – An Amazon SES SMTP password. For more information, see Obtaining Amazon SES SMTP credentials.

To create the account

- Use the following SQL statement.

```
USE msdb
GO

EXECUTE msdb.dbo.sysmail_add_account_sp
    @account_name        = 'SES',
    @description         = 'Mail account for sending outgoing notifications.',
    @email_address       = 'nobody@example.com',
    @display_name        = 'Automated Mailer',
    @mailserver_name     = 'vpce-0a1b2c3d4e5f-01234567.email-smtp.us-
west-2.vpce.amazonaws.com',
    @port                = 587,
    @enable_ssl          = 1,
    @username            = 'Smtp_Username',
    @password            = 'Smtp_Password';
GO
```

> **Note**
>
> Specify credentials other than the prompts shown here as a security best practice.

Adding the Database Mail account to the Database Mail profile

To add the Database Mail account to the Database Mail profile, you use the sysmail_add_profileaccount_sp stored procedure. The following example adds the SES account to the `Notifications` profile.

To add the account to the profile

- Use the following SQL statement.

```
USE msdb
GO

EXECUTE msdb.dbo.sysmail_add_profileaccount_sp
    @profile_name        = 'Notifications',
    @account_name        = 'SES',
```

```
    @sequence_number     = 1;
GO
```

Adding users to the Database Mail profile

To grant permission for an `msdb` database principal to use a Database Mail profile, you use the sysmail_add_principalprofile_sp stored procedure. A *principal* is an entity that can request SQL Server resources. The database principal must map to a SQL Server authentication user, a Windows Authentication user, or a Windows Authentication group.

The following example grants public access to the `Notifications` profile.

To add a user to the profile

- Use the following SQL statement.

```
USE msdb
GO

EXECUTE msdb.dbo.sysmail_add_principalprofile_sp
    @profile_name       = 'Notifications',
    @principal_name     = 'public',
    @is_default         = 1;
GO
```

Amazon RDS stored procedures and functions for Database Mail

Microsoft provides stored procedures for using Database Mail, such as creating, listing, updating, and deleting accounts and profiles. In addition, RDS provides the stored procedures and functions for Database Mail shown in the following table.

Procedure/Function	Description
rds_fn_sysmail_allitems	Shows sent messages, including those submitted by other users.
rds_fn_sysmail_event_log	Shows events, including those for messages submitted by other users.

Procedure/Function	Description
rds_fn_sysmail_mailattachments	Shows attachments, including those to messages submitted by other users.
rds_sysmail_control	Starts and stops the mail queue (DatabaseMail.exe process).
rds_sysmail_delete_mailitems_sp	Deletes email messages sent by all users from the Database Mail internal tables.

Sending email messages using Database Mail

You use the sp_send_dbmail stored procedure to send email messages using Database Mail.

Usage

```
EXEC msdb.dbo.sp_send_dbmail
@profile_name = 'profile_name',
@recipients = 'recipient1@example.com[; recipient2; ... recipientn]',
@subject = 'subject',
@body = 'message_body',
[@body_format = 'HTML'],
[@file_attachments = 'file_path1; file_path2; ... file_pathn'],
[@query = 'SQL_query'],
[@attach_query_result_as_file = 0|1]';
```

The following parameters are required:

- `@profile_name` – The name of the Database Mail profile from which to send the message.
- `@recipients` – The semicolon-delimited list of email addresses to which to send the message.
- `@subject` – The subject of the message.
- @body – The body of the message. You can also use a declared variable as the body.

The following parameters are optional:

- `@body_format` – This parameter is used with a declared variable to send email in HTML format.
- `@file_attachments` – The semicolon-delimited list of message attachments. File paths must be absolute paths.

- `@query` – A SQL query to run. The query results can be attached as a file or included in the body of the message.
- `@attach_query_result_as_file` – Whether to attach the query result as a file. Set to 0 for no, 1 for yes. The default is 0.

Examples

The following examples demonstrate how to send email messages.

Example of sending a message to a single recipient

```
USE msdb
GO

EXEC msdb.dbo.sp_send_dbmail
     @profile_name       = 'Notifications',
     @recipients         = 'nobody@example.com',
     @subject            = 'Automated DBMail message - 1',
     @body               = 'Database Mail configuration was successful.';
GO
```

Example of sending a message to multiple recipients

```
USE msdb
GO

EXEC msdb.dbo.sp_send_dbmail
     @profile_name       = 'Notifications',
     @recipients         = 'recipient1@example.com;recipient2@example.com',
     @subject            = 'Automated DBMail message - 2',
     @body               = 'This is a message.';
GO
```

Example of sending a SQL query result as a file attachment

```
USE msdb
GO

EXEC msdb.dbo.sp_send_dbmail
     @profile_name       = 'Notifications',
     @recipients         = 'nobody@example.com',
```

```
     @subject            = 'Test SQL query',
     @body               = 'This is a SQL query test.',
     @query              = 'SELECT * FROM abc.dbo.test',
     @attach_query_result_as_file = 1;
GO
```

Example of sending a message in HTML format

```
USE msdb
GO

DECLARE @HTML_Body as NVARCHAR(500) = 'Hi, <h4> Heading </h4> </br> See the report. <b>
 Regards </b>';

EXEC msdb.dbo.sp_send_dbmail
     @profile_name       = 'Notifications',
     @recipients         = 'nobody@example.com',
     @subject            = 'Test HTML message',
     @body               = @HTML_Body,
     @body_format        = 'HTML';
GO
```

Example of sending a message using a trigger when a specific event occurs in the database

```
USE AdventureWorks2017
GO
IF OBJECT_ID ('Production.iProductNotification', 'TR') IS NOT NULL
DROP TRIGGER Purchasing.iProductNotification
GO

CREATE TRIGGER iProductNotification ON Production.Product
   FOR INSERT
   AS
   DECLARE @ProductInformation nvarchar(255);
   SELECT
   @ProductInformation = 'A new product, ' + Name + ', is now available for $' +
 CAST(StandardCost AS nvarchar(20)) + '!'
   FROM INSERTED i;

EXEC msdb.dbo.sp_send_dbmail
     @profile_name       = 'Notifications',
     @recipients         = 'nobody@example.com',
     @subject            = 'New product information',
```

```
    @body                = @ProductInformation;
GO
```

Viewing messages, logs, and attachments

You use RDS stored procedures to view messages, event logs, and attachments.

To view all email messages

- Use the following SQL query.

```
SELECT * FROM msdb.dbo.rds_fn_sysmail_allitems(); --WHERE sent_status='sent' or
 'failed' or 'unsent'
```

To view all email event logs

- Use the following SQL query.

```
SELECT * FROM msdb.dbo.rds_fn_sysmail_event_log();
```

To view all email attachments

- Use the following SQL query.

```
SELECT * FROM msdb.dbo.rds_fn_sysmail_mailattachments();
```

Deleting messages

You use the `rds_sysmail_delete_mailitems_sp` stored procedure to delete messages.

> **Note**
>
> RDS automatically deletes mail table items when DBMail history data reaches 1 GB in size, with a retention period of at least 24 hours.
> If you want to keep mail items for a longer period, you can archive them. For more information, see Create a SQL Server Agent job to archive Database Mail messages and event logs in the Microsoft documentation.

To delete all email messages

- Use the following SQL statement.

```
DECLARE @GETDATE datetime
SET @GETDATE = GETDATE();
EXECUTE msdb.dbo.rds_sysmail_delete_mailitems_sp @sent_before = @GETDATE;
GO
```

To delete all email messages with a particular status

- Use the following SQL statement to delete all failed messages.

```
DECLARE @GETDATE datetime
SET @GETDATE = GETDATE();
EXECUTE msdb.dbo.rds_sysmail_delete_mailitems_sp @sent_status = 'failed';
GO
```

Starting and stopping mail queue

Use the following instructions to start and stop the DB mail queue:

Topics

- Starting the mail queue
- Stopping the mail queue

Starting the mail queue

You use the `rds_sysmail_control` stored procedure to start the Database Mail process.

> **Note**
>
> Enabling Database Mail automatically starts the mail queue.

To start the mail queue

- Use the following SQL statement.

```
EXECUTE msdb.dbo.rds_sysmail_control start;
GO
```

Stopping the mail queue

You use the `rds_sysmail_control` stored procedure to stop the Database Mail process.

To stop the mail queue

- Use the following SQL statement.

```
EXECUTE msdb.dbo.rds_sysmail_control stop;
GO
```

Working with file attachments

The following file attachment extensions aren't supported in Database Mail messages from RDS on SQL
Server: .ade, .adp, .apk, .appx, .appxbundle, .bat, .bak, .cab, .chm, .cmd, .com, .cpl, .dll, .dmg, .exe, .hta, .inf1 and .wsh.

Database Mail uses the Microsoft Windows security context of the current user to control access to files. Users who log in with SQL Server Authentication can't attach files using the `@file_attachments` parameter with the `sp_send_dbmail` stored procedure. Windows doesn't allow SQL Server to provide credentials from a remote computer to another remote computer. Therefore, Database Mail can't attach files from a network share when the command is run from a computer other than the computer running SQL Server.

However, you can use SQL Server Agent jobs to attach files. For more information on SQL Server Agent, see Using SQL Server Agent for Amazon RDS and SQL Server Agent in the Microsoft documentation.

Considerations for Multi-AZ deployments

When you configure Database Mail on a Multi-AZ DB instance, the configuration isn't automatically propagated to the secondary. We recommend converting the Multi-AZ instance to a Single-AZ instance, configuring Database Mail, and then converting the DB instance back to Multi-AZ. Then both the primary and secondary nodes have the Database Mail configuration.

If you create a read replica from your Multi-AZ instance that has Database Mail configured, the replica inherits the configuration, but without the password to the SMTP server. Update the Database Mail account with the password.

Removing the SMTP (port 25) restriction

By default, AWS blocks outbound traffic on SMTP (port 25) for RDS for SQL Server DB instances. This is done to prevent spam based on the elastic network interface owner's policies. You can remove this restriction if needed. For more information, see How do I remove the restriction on port 25 from my Amazon EC2 instance or Lambda function?.

Instance store support for the tempdb database on Amazon RDS for SQL Server

An *instance store* provides temporary block-level storage for your DB instance. This storage is located on disks that are physically attached to the host computer. These disks have Non-Volatile Memory Express (NVMe) instance storage that is based on solid-state drives (SSDs). This storage is optimized for low latency, very high random I/O performance, and high sequential read throughput.

By placing `tempdb` data files and `tempdb` log files on the instance store, you can achieve lower read and write latencies compared to standard storage based on Amazon EBS.

> **Note**
>
> SQL Server database files and database log files aren't placed on the instance store.

Enabling the instance store

When RDS provisions DB instances with one of the following instance classes, the `tempdb` database is automatically placed onto the instance store:

- db.m5d
- db.r5d
- db.x2iedn

To enable the instance store, do one of the following:

- Create a SQL Server DB instance using one of these instance types. For more information, see Creating an Amazon RDS DB instance.
- Modify an existing SQL Server DB instance to use one of them. For more information, see Modifying an Amazon RDS DB instance.

The instance store is available in all AWS Regions where one or more of these instance types are supported. For more information on the `db.m5d` and `db.r5d` instance classes, see DB instance classes. For more information on the instance classes supported by Amazon RDS for SQL Server, see DB instance class support for Microsoft SQL Server.

File location and size considerations

On instances without an instance store, RDS stores the `tempdb` data and log files in the `D:\rdsdbdata\DATA` directory. Both files start at 8 MB by default.

On instances with an instance store, RDS stores the `tempdb` data and log files in the `T:\rdsdbdata\DATA` directory.

When `tempdb` has only one data file (`tempdb.mdf`) and one log file (`templog.ldf`), `templog.ldf` starts at 8 MB by default and `tempdb.mdf` starts at 80% or more of the instance's storage capacity. Twenty percent of the storage capacity or 200 GB, whichever is less, is kept free to start. Multiple `tempdb` data files split the 80% disk space evenly, while log files always have an 8-MB initial size.

For example, if you modify your DB instance class from `db.m5.2xlarge` to `db.m5d.2xlarge`, the size of `tempdb` data files increases from 8 MB each to 234 GB in total.

> **Note**
>
> Besides the `tempdb` data and log files on the instance store (`T:\rdsdbdata\DATA`), you can still create extra `tempdb` data and log files on the data volume (`D:\rdsdbdata\DATA`). Those files always have an 8 MB initial size.

Backup considerations

You might need to retain backups for long periods, incurring costs over time. The `tempdb` data and log blocks can change very often depending on the workload. This can greatly increase the DB snapshot size.

When `tempdb` is on the instance store, snapshots don't include temporary files. This means that snapshot sizes are smaller and consume less of the free backup allocation compared to EBS-only storage.

Disk full errors

If you use all of the available space in the instance store, you might receive errors such as the following:

- The transaction log for database 'tempdb' is full due to 'ACTIVE_TRANSACTION'.

- Could not allocate space for object 'dbo.SORT temporary run storage: 140738941419520' in database 'tempdb' because the 'PRIMARY' filegroup is full. Create disk space by deleting unneeded files, dropping objects in the filegroup, adding additional files to the filegroup, or setting autogrowth on for existing files in the filegroup.

You can do one or more of the following when the instance store is full:

- Adjust your workload or the way you use `tempdb`.
- Scale up to use a DB instance class with more NVMe storage.
- Stop using the instance store, and use an instance class with only EBS storage.
- Use a mixed mode by adding secondary data or log files for `tempdb` on the EBS volume.

Removing the instance store

To remove the instance store, modify your SQL Server DB instance to use an instance type that doesn't support instance store, such as db.m5, db.r5, or db.x1e.

> **Note**
>
> When you remove the instance store, the temporary files are moved to the `D:\rdsdbdata\DATA` directory and reduced in size to 8 MB.

Using extended events with Amazon RDS for Microsoft SQL Server

You can use extended events in Microsoft SQL Server to capture debugging and troubleshooting information for Amazon RDS for SQL Server. Extended events replace SQL Trace and Server Profiler, which have been deprecated by Microsoft. Extended events are similar to profiler traces but with more granular control on the events being traced. Extended events are supported for SQL Server versions 2016 and later on Amazon RDS. For more information, see Extended events overview in the Microsoft documentation.

Extended events are turned on automatically for users with master user privileges in Amazon RDS for SQL Server.

Topics

- Limitations and recommendations
- Configuring extended events on RDS for SQL Server
- Considerations for Multi-AZ deployments
- Querying extended event files

Limitations and recommendations

When using extended events on RDS for SQL Server, the following limitations apply:

- Extended events are supported only for the Enterprise and Standard Editions.
- You can't alter default extended event sessions.
- Make sure to set the session memory partition mode to `NONE`.
- Session event retention mode can be either `ALLOW_SINGLE_EVENT_LOSS` or `ALLOW_MULTIPLE_EVENT_LOSS`.
- Event Tracing for Windows (ETW) targets aren't supported.
- Make sure that file targets are in the `D:\rdsdbdata\log` directory.
- For pair matching targets, set the `respond_to_memory_pressure` property to 1.
- Ring buffer target memory can't be greater than 4 MB.
- The following actions aren't supported:
  - `debug_break`
  - `create_dump_all_threads`

- `create_dump_single_threads`

- The `rpc_completed` event is supported on the following versions and later: 15.0.4083.2, 14.0.3370.1, 13.0.5865.1, 12.0.6433.1, 11.0.7507.2.

Configuring extended events on RDS for SQL Server

On RDS for SQL Server, you can configure the values of certain parameters of extended event sessions. The following table describes the configurable parameters.

Parameter name	Description
`xe_session_max_memory`	Specifies the maximum amount of memory to allocate t corresponds to the `max_memory` setting of the event s
`xe_session_max_event_size`	Specifies the maximum memory size allowed for large e `_size` setting of the event session.
`xe_session_max_dispatch_latency`	Specifies the amount of time that events are buffered in session targets. This value corresponds to the `max_disp`
`xe_file_target_size`	Specifies the maximum size of the file target. This value file target.
`xe_file_retention`	Specifies the retention time in days for files generated b

> **Note**
>
> Setting `xe_file_retention` to zero causes .xel files to be removed automatically after the lock on these files is released by SQL Server. The lock is released whenever an .xel file reaches the size limit set in `xe_file_target_size`.

You can use the `rdsadmin.dbo.rds_show_configuration` stored procedure to show the current values of these parameters. For example, use the following SQL statement to view the current setting of `xe_session_max_memory`.

```
exec rdsadmin.dbo.rds_show_configuration 'xe_session_max_memory'
```

You can use the `rdsadmin.dbo.rds_set_configuration` stored procedure to modify them. For example, use the following SQL statement to set `xe_session_max_memory` to 4 MB.

```
exec rdsadmin.dbo.rds_set_configuration 'xe_session_max_memory', 4
```

Considerations for Multi-AZ deployments

When you create an extended event session on a primary DB instance, it doesn't propagate to the standby replica. You can fail over and create the extended event session on the new primary DB instance. Or you can remove and then re-add the Multi-AZ configuration to propagate the extended event session to the standby replica. RDS stops all nondefault extended event sessions on the standby replica, so that these sessions don't consume resources on the standby. Because of this, after a standby replica becomes the primary DB instance, make sure to manually start the extended event sessions on the new primary.

> **Note**
>
> This approach applies to both Always On Availability Groups and Database Mirroring.

You can also use a SQL Server Agent job to track the standby replica and start the sessions if the standby becomes the primary. For example, use the following query in your SQL Server Agent job step to restart event sessions on a primary DB instance.

```
BEGIN
    IF (DATABASEPROPERTYEX('rdsadmin','Updateability')='READ_WRITE'
    AND DATABASEPROPERTYEX('rdsadmin','status')='ONLINE'
    AND (DATABASEPROPERTYEX('rdsadmin','Collation') IS NOT NULL OR
 DATABASEPROPERTYEX('rdsadmin','IsAutoClose')=1)
    )
    BEGIN
        IF NOT EXISTS (SELECT 1 FROM sys.dm_xe_sessions WHERE name='xe1')
            ALTER EVENT SESSION xe1 ON SERVER STATE=START
        IF NOT EXISTS (SELECT 1 FROM sys.dm_xe_sessions WHERE name='xe2')
            ALTER EVENT SESSION xe2 ON SERVER STATE=START
    END
END
```

This query restarts the event sessions `xe1` and `xe2` on a primary DB instance if these sessions are in a stopped state. You can also add a schedule with a convenient interval to this query.

Querying extended event files

You can either use SQL Server Management Studio or the `sys.fn_xe_file_target_read_file` function to view data from extended events that use file targets. For more information on this function, see sys.fn_xe_file_target_read_file (Transact-SQL) in the Microsoft documentation.

Extended event file targets can only write files to the `D:\rdsdbdata\log` directory on RDS for SQL Server.

As an example, use the following SQL query to list the contents of all files of extended event sessions whose names start with `xe`.

```
SELECT * FROM sys.fn_xe_file_target_read_file('d:\rdsdbdata\log\xe*', null,null,null);
```

Access to transaction log backups with RDS for SQL Server

With access to transaction log backups for RDS for SQL Server, you can list the transaction log backup files for a database and copy them to a target Amazon S3 bucket. By copying transaction log backups in an Amazon S3 bucket, you can use them in combination with full and differential database backups to perform point in time database restores. You use RDS stored procedures to set up access to transaction log backups, list available transaction log backups, and copy them to your Amazon S3 bucket.

Access to transaction log backups provides the following capabilities and benefits:

- List and view the metadata of available transaction log backups for a database on an RDS for SQL Server DB instance.
- Copy available transaction log backups from RDS for SQL Server to a target Amazon S3 bucket.
- Perform point-in-time restores of databases without the need to restore an entire DB instance. For more information on restoring a DB instance to a point in time, see Restoring a DB instance to a specified time for Amazon RDS.

Availability and support

Access to transaction log backups is supported in all AWS Regions. Access to transaction log backups is available for all editions and versions of Microsoft SQL Server supported on Amazon RDS.

Requirements

The following requirements must be met before enabling access to transaction log backups:

- Automated backups must be enabled on the DB instance and the backup retention must be set to a value of one or more days. For more information on enabling automated backups and configuring a retention policy, see Enabling automated backups.
- An Amazon S3 bucket must exist in the same account and Region as the source DB instance. Before enabling access to transaction log backups, choose an existing Amazon S3 bucket or create a new bucket to use for your transaction log backup files.
- An Amazon S3 bucket permissions policy must be configured as follows to allow Amazon RDS to copy transaction log files into it:

  1. Set the object account ownership property on the bucket to **Bucket Owner Preferred**.

2. Add the following policy. There will be no policy by default, so use the bucket Access Control Lists (ACL) to edit the bucket policy and add it.

The following example uses an ARN to specify a resource. We recommend using the `SourceArn` and `SourceAccount` global condition context keys in resource-based trust relationships to limit the service's permissions to a specific resource. For more information on working with ARNs, see Amazon resource names (ARNs) and Amazon Resource Names (ARNs) in Amazon RDS.

Example of an Amazon S3 permissions policy for access to transaction log backups

JSON

```
   {
   "Version":"2012-10-17",
   "Statement": [
       {
           "Sid": "Only allow writes to my bucket with bucket owner full
 control",
           "Effect": "Allow",
           "Principal": {
               "Service": "backups.rds.amazonaws.com"
           },
           "Action": "s3:PutObject",
           "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/{customer_path}/*",
           "Condition": {
               "StringEquals": {
                   "s3:x-amz-acl": "bucket-owner-full-control",
                   "aws:sourceAccount": "{customer_account}",
                   "aws:sourceArn": "{db_instance_arn}"
               }
           }
       }
   ]
}
```

- An AWS Identity and Access Management (IAM) role to access the Amazon S3 bucket. If you already have an IAM role, you can use that. You can choose to have a new IAM role created for you when you add the `SQLSERVER_BACKUP_RESTORE` option by using the AWS Management Console. Alternatively, you can create a new one manually. For more information on creating and

configuring an IAM role with SQLSERVER_BACKUP_RESTORE, see Manually creating an IAM role for native backup and restore.

- The SQLSERVER_BACKUP_RESTORE option must be added to an option group on your DB instance. For more information on adding the SQLSERVER_BACKUP_RESTORE option, see Support for native backup and restore in SQL Server.

> **Note**
>
> If your DB instance has storage encryption enabled, the AWS KMS (KMS) actions and key must be provided in the IAM role provided in the native backup and restore option group.

Optionally, if you intend to use the `rds_restore_log` stored procedure to perform point in time database restores, we recommend using the same Amazon S3 path for the native backup and restore option group and access to transaction log backups. This method ensures that when Amazon RDS assumes the role from the option group to perform the restore log functions, it has access to retrieve transaction log backups from the same Amazon S3 path.

- If the DB instance is encrypted, regardless of encryption type (AWS managed key or customer managed key), you must provide a customer managed KMS key in the IAM role and in the `rds_tlog_backup_copy_to_S3` stored procedure.

Limitations and recommendations

Access to transaction log backups has the following limitations and recommendations:

- You can list and copy up to the last seven days of transaction log backups for any DB instance that has backup retention configured between one to 35 days.
- The Amazon S3 bucket used for access to transaction log backups must exist in the same account and Region as the source DB instance. Cross-account and cross-region copy is not supported.
- Only one Amazon S3 bucket can be configured as a target to copy transaction log backups into. You can choose a new target Amazon S3 bucket with the `rds_tlog_copy_setup` stored procedure. For more information on choosing a new target Amazon S3 bucket, see Setting up access to transaction log backups.
- You cannot specify the KMS key when using the `rds_tlog_backup_copy_to_S3` stored procedure if your RDS instance is not enabled for storage encryption.

- Multi-account copying is not supported. The IAM role used for copying will only permit write access to Amazon S3 buckets within the owner account of the DB instance.
- Only two concurrent tasks of any type may be run on an RDS for SQL Server DB instance.
- Only one copy task can run for a single database at a given time. If you want to copy transaction log backups for multiple databases on the DB instance, use a separate copy task for each database.
- If you copy a transaction log backup that already exists with the same name in the Amazon S3 bucket, the existing transaction log backup will be overwritten.
- You can only run the stored procedures that are provided with access to transaction log backups on the primary DB instance. You can't run these stored procedures on an RDS for SQL Server read replica or on a secondary instance of a Multi-AZ DB cluster.
- If the RDS for SQL Server DB instance is rebooted while the `rds_tlog_backup_copy_to_S3` stored procedure is running, the task will automatically restart from the beginning when the DB instance is back online. Any transaction log backups that had been copied to the Amazon S3 bucket while the task was running before the reboot will be overwritten.
- The Microsoft SQL Server system databases and the `RDSAdmin` database cannot be configured for access to transaction log backups.
- Copying to buckets encrypted by SSE-KMS isn't supported.

Setting up access to transaction log backups

To set up access to transaction log backups, complete the list of requirements in the Requirements section, and then run the `rds_tlog_copy_setup` stored procedure. The procedure will enable the access to transaction log backups feature at the DB instance level. You don't need to run it for each individual database on the DB instance.

> **Important**
>
> The database user must be granted the `db_owner` role within SQL Server on each database to configure and use the access to transaction log backups feature.

Example usage:

```
exec msdb.dbo.rds_tlog_copy_setup
```

```
@target_s3_arn='arn:aws:s3:::amzn-s3-demo-bucket/myfolder';
```

The following parameter is required:

- `@target_s3_arn` – The ARN of the target Amazon S3 bucket to copy transaction log backups files to.

Example of setting an Amazon S3 target bucket:

```
exec msdb.dbo.rds_tlog_copy_setup @target_s3_arn='arn:aws:s3:::amzn-s3-demo-logging-bucket/mytestdb1';
```

To validate the configuration, call the `rds_show_configuration` stored procedure.

Example of validating the configuration:

```
exec rdsadmin.dbo.rds_show_configuration @name='target_s3_arn_for_tlog_copy';
```

To modify access to transaction log backups to point to a different Amazon S3 bucket, you can view the current Amazon S3 bucket value and re-run the `rds_tlog_copy_setup` stored procedure using a new value for the `@target_s3_arn`.

Example of viewing the existing Amazon S3 bucket configured for access to transaction log backups

```
exec rdsadmin.dbo.rds_show_configuration @name='target_s3_arn_for_tlog_copy';
```

Example of updating to a new target Amazon S3 bucket

```
exec msdb.dbo.rds_tlog_copy_setup @target_s3_arn='arn:aws:s3:::amzn-s3-demo-logging-bucket1/mynewfolder';
```

Listing available transaction log backups

With RDS for SQL Server, databases configured to use the full recovery model and a DB instance backup retention set to one or more days have transaction log backups automatically enabled. By enabling access to transaction log backups, up to seven days of those transaction log backups are made available for you to copy into your Amazon S3 bucket.

After you have enabled access to transaction log backups, you can start using it to list and copy available transaction log backup files.

Listing transaction log backups

To list all transaction log backups available for an individual database, call the `rds_fn_list_tlog_backup_metadata` function. You can use an `ORDER BY` or a `WHERE` clause when calling the function.

Example of listing and filtering available transaction log backup files

```
SELECT * from msdb.dbo.rds_fn_list_tlog_backup_metadata('mydatabasename');
SELECT * from msdb.dbo.rds_fn_list_tlog_backup_metadata('mydatabasename') WHERE
 rds_backup_seq_id = 3507;
SELECT * from msdb.dbo.rds_fn_list_tlog_backup_metadata('mydatabasename') WHERE
 backup_file_time_utc > '2022-09-15 20:44:01' ORDER BY backup_file_time_utc DESC;
```

100 %

Results | Messages

	db_name	db_id	family_guid	rds_backup_seq_id	backup_file_epoch	backup_file_time_utc	starting_lsn	ending_lsn	is_log_chain_broken	file_size_bytes	Error
1	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	43	1661846641	2022-08-30 08:04:01	5450000085730100001	5450000085731000001	0	35564	NULL
2	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	44	1661846941	2022-08-30 08:09:01	5450000085731000001	5450000085731900001	0	35473	NULL
3	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	45	1661847241	2022-08-30 08:14:01	5450000085731900001	5450000085732800001	0	35394	NULL
4	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	46	1661847541	2022-08-30 08:19:01	5450000085732800001	5450000085733700001	0	35374	NULL
5	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	47	1661847841	2022-08-30 08:24:01	5450000085733700001	5450000085734600001	0	35601	NULL
6	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	48	1661848142	2022-08-30 08:29:02	5450000085734600001	5450000085735500001	0	35470	NULL
7	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	49	1661848441	2022-08-30 08:34:01	5450000085735500001	5450000085736400001	0	35491	NULL
8	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	50	1661848741	2022-08-30 08:39:01	5450000085736400001	5450000085737300001	0	35520	NULL
9	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	51	1661849041	2022-08-30 08:44:01	5450000085737300001	5450000085738200001	0	35326	NULL
10	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	52	1661849341	2022-08-30 08:49:01	5450000085738200001	5450000085739100001	0	35407	NULL
11	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	53	1661849641	2022-08-30 08:54:01	5450000085739100001	5450000085740000001	0	35491	NULL
12	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	54	1661849941	2022-08-30 08:59:01	5450000085740000001	5450000085740900001	0	35438	NULL
13	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	55	1661850241	2022-08-30 09:04:01	5450000085740900001	5450000085741800001	0	35319	NULL
14	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	56	1661850541	2022-08-30 09:09:01	5450000085741800001	5450000085742700001	0	35270	NULL
15	tpcc	6	CD11CB3D-B5E4-46D9-B462-CE40CDA97E89	57	1661850841	2022-08-30 09:14:01	5450000085742700001	5450000085743600001	0	35476	NULL

The `rds_fn_list_tlog_backup_metadata` function returns the following output:

Column name	Data type	Description
db_name	sysname	The database name provided to list the transacti on log backups for.
db_id	int	The internal database identifier for the input parameter db_name.
family_guid	uniqueidentifier	The unique ID of the original database at creation. This value remains the same when the database is restored, even to a different database name.
rds_backu p_seq_id	int	The ID that RDS uses internally to maintain a sequence number for each transaction log backup file.
backup_fi le_epoch	bigint	The epoch time that a transaction backup file was generated.
backup_fi le_time_utc	datetime	The UTC time-converted value for the backup_fi le_epoch value.
starting_lsn	numeric(25,0)	The log sequence number of the first or oldest log record of a transaction log backup file.
ending_lsn	numeric(25,0)	The log sequence number of the last or next log record of a transaction log backup file.
is_log_ch ain_broken	bit	A boolean value indicating if the log chain is broken between the current transaction log backup file and the previous transaction log backup file.
file_size _bytes	bigint	The size of the transactional backup set in bytes.
Error	varchar(4000)	Error message if the rds_fn_list_tlog_b ackup_metadata function throws an exception. NULL if no exceptions.

Copying transaction log backups

To copy a set of available transaction log backups for an individual database to your Amazon S3 bucket, call the `rds_tlog_backup_copy_to_S3` stored procedure. The `rds_tlog_backup_copy_to_S3` stored procedure will initiate a new task to copy transaction log backups.

> **Note**
>
> The `rds_tlog_backup_copy_to_S3` stored procedure will copy the transaction log backups without validating against `is_log_chain_broken` attribute. For this reason, you should manually confirm an unbroken log chain before running the `rds_tlog_backup_copy_to_S3` stored procedure. For further explanation, see Validating the transaction log backup log chain.

Example usage of the `rds_tlog_backup_copy_to_S3` stored procedure

```
exec msdb.dbo.rds_tlog_backup_copy_to_S3
 @db_name='mydatabasename',
 [@kms_key_arn='arn:aws:kms:region:account-id:key/key-id'],
 [@backup_file_start_time='2022-09-01 01:00:15'],
 [@backup_file_end_time='2022-09-01 21:30:45'],
 [@starting_lsn=149000000112100001],
 [@ending_lsn=149000000120400001],
 [@rds_backup_starting_seq_id=5],
 [@rds_backup_ending_seq_id=10];
```

The following input parameters are available:

Parameter	Description
@db_name	The name of the database to copy transaction log backups for
@kms_key_arn	A customer managed KMS key. If you encrypt your DB instance with an AWS managed KMS key, you must create a customer managed key. If you encrypt your DB instance with a customer managed key, you can use the same KMS key ARN.

Parameter	Description
@backup_file_start_time	The UTC timestamp as provided from the [backup_file_time_utc] column of the rds_fn_list_tlog_backup_metadata function.
@backup_file_end_time	The UTC timestamp as provided from the [backup_file_time_utc] column of the rds_fn_list_tlog_backup_metadata function.
@starting_lsn	The log sequence number (LSN) as provided from the [starting_lsn] column of the rds_fn_list_tlog_backup_metadata function
@ending_lsn	The log sequence number (LSN) as provided from the [ending_lsn] column of the rds_fn_list_tlog_backup_metadata function.
@rds_backup_starting_seq_id	The sequence ID as provided from the [rds_backup_seq_id] column of the rds_fn_list_tlog_backup_metadata function.
@rds_backup_ending_seq_id	The sequence ID as provided from the [rds_backup_seq_id] column of the rds_fn_list_tlog_backup_metadata function.

You can specify a set of either the time, LSN, or sequence ID parameters. Only one set of parameters are required.

You can also specify just a single parameter in any of the sets. For example, by providing a value for only the backup_file_end_time parameter, all available transaction log backup files prior to that time within the seven-day limit will be copied to your Amazon S3 bucket.

Following are the valid input parameter combinations for the rds_tlog_backup_copy_to_S3 stored procedure.

Parameters provided	Expected result
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name = 'testdb1', @backup_file_start_time='2022-08-23 00:00:00', @backup_file_end_time='2022-08-30 00:00:00';`	Copies transaction log backups from the last seven days and exist between the provided range of `backup_file_start_time` and `backup_file_end_time` . In this example, the stored procedure will copy transaction log backups that were generated between '2022-08-23 00:00:00' and '2022-08-30 00:00:00'.
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name = 'testdb1', @backup_file_start_time='2022-08-23 00:00:00';`	Copies transaction log backups from the last seven days and starting from the provided `backup_file_start_time` . In this example, the stored procedure will copy transaction log backups from '2022-08-23 00:00:00' up to the

Parameters provided	Expected result	
	latest transaction log backup.	
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name = 'testdb1', @backup_file_end_time='2022-08-30 00:00:00';`	Copies transaction log backups from the last seven days up to the provided `backup_file_end_time` . In this example, the stored procedure will copy transaction log backups from '2022-08-23 00:00:00 up to '2022-08-30 00:00:00'.	

Parameters provided	Expected result	
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name='testdb1', @starting_lsn =14900000 00040007, @ending_lsn = 149000000 0050009;`	Copies transaction log backups that are available from the last seven days and are between the provided range of the `starting_lsn` and `ending_lsn` . In this example, the stored procedure will copy transacti on log backups from the last seven days with an LSN range between 149000000 0040007 and 149000000 0050009.	

Parameters provided	Expected result
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name='testdb1', @starting_lsn =14900000 00040007;`	Copies transacti on log backups that are available from the last seven days, beginning from the provided `starting_lsn` . In this example, the stored procedure will copy transaction log backups from LSN 149000000 0040007 up to the latest transaction log backup.
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name='testdb1', @ending_lsn =14900000 00050009;`	Copies transaction log backups that are available from the last seven days, up to the provided `ending_lsn` . In this example, the stored procedure will copy transacti on log backups beginning from the last seven days up to lsn 149000000 0050009.

Parameters provided	Expected result
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name='testdb1', @rds_backup_starting_seq_id= 2000, @rds_backup_ending_seq_id= 5000;`	Copies transaction log backups that are available from the last seven days, and exist between the provided range of `rds_backup_starting_seq_id` and `rds_backup_ending_seq_id` . In this example, the stored procedure will copy transaction log backups beginning from the last seven days and within the provided rds backup sequence id range, starting from seq_id 2000 up to seq_id 5000.

Parameters provided	Expected result	
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name='testdb1', @rds_backup_starting_seq_id= 2000;`	Copies transaction log backups that are available from the last seven days, beginning from the provided `rds_backup_starting_seq_id` . In this example, the stored procedure will copy transaction log backups beginning from seq_id 2000, up to the latest transaction log backup.	
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name='testdb1', @rds_backup_ending_seq_id= 5000;`	Copies transaction log backups that are available from the last seven days, up to the provided `rds_backup_ending_seq_id` . In this example, the stored procedure will copy transaction log backups beginning from the last seven days, up to seq_id 5000.	

Parameters provided	Expected result	
`exec msdb.dbo.rds_tlog_backup_copy_to_S3 @db_name='testdb1', @rds_backup_starting_seq_id= 2000; @rds_backup_ending_seq_id= 2000;`	Copies a single transaction log backup with the provided `rds_backup_starting_seq_id` , if available within the last seven days. In this example, the stored procedure will copy a single transaction log backup that has a seq_id of 2000, if it exists within the last seven days.	

Validating the transaction log backup log chain

Databases configured for access to transaction log backups must have automated backup retention enabled. Automated backup retention sets the databases on the DB instance to the FULL recovery model. To support point in time restore for a database, avoid changing the database recovery model, which can result in a broken log chain. We recommend keeping the database set to the FULL recovery model.

To manually validate the log chain before copying transaction log backups, call the `rds_fn_list_tlog_backup_metadata` function and review the values in the `is_log_chain_broken` column. A value of "1" indicates the log chain was broken between the current log backup and the previous log backup.

The following example shows a broken log chain in the output from the `rds_fn_list_tlog_backup_metadata` stored procedure.

rds_sequence_id	first_lsn	last_lsn	is_log_chain_broken
43	90023	90457	0
44	90457	90985	0
45	90987	92034	1

In a normal log chain, the log sequence number (LSN) value for first_lsn for given rds_sequence_id should match the value of last_lsn in the preceding rds_sequence_id. In the image, the rds_sequence_id of 45 has a first_lsn value 90987, which does not match the last_lsn value of 90985 for preceeding rds_sequence_id 44.

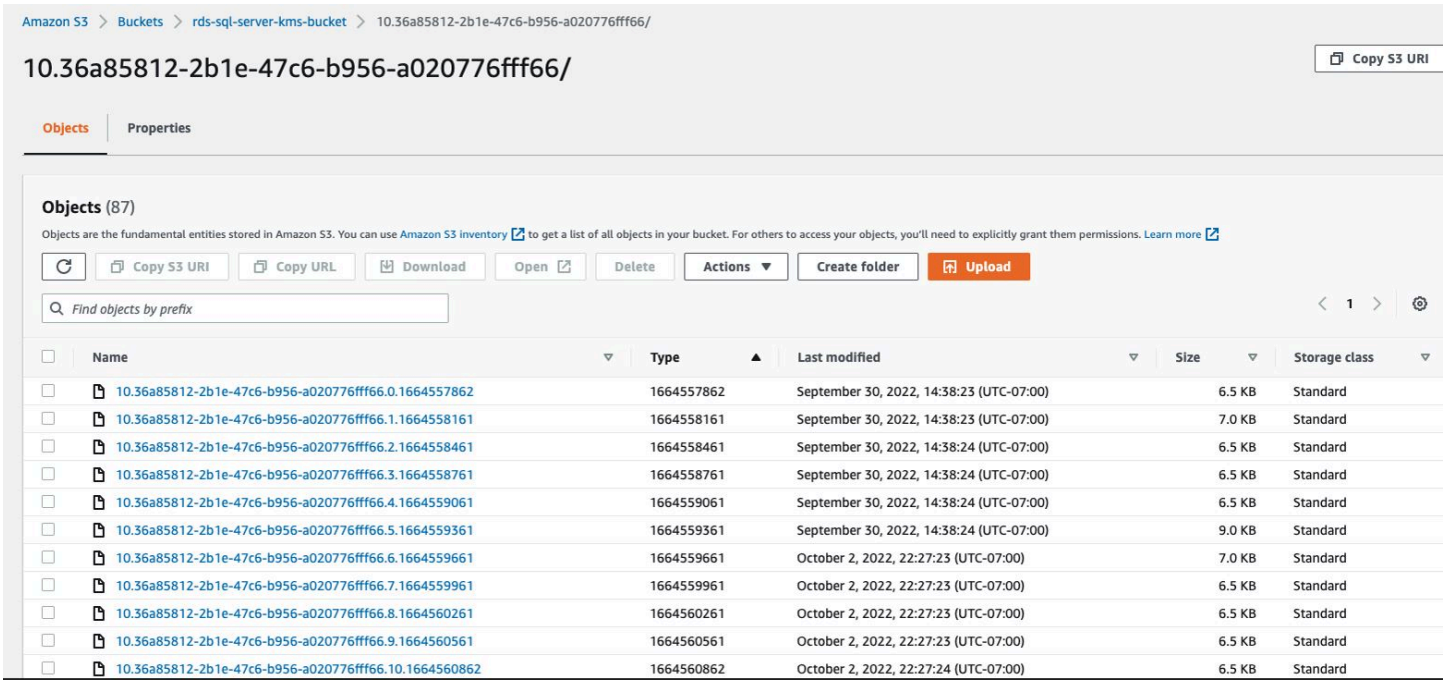
For more information about SQL Server transaction log architecture and log sequence numbers, see Transaction Log Logical Architecture in the Microsoft SQL Server documentation.

Amazon S3 bucket folder and file structure

Transaction log backups have the following standard structure and naming convention within an Amazon S3 bucket:

- A new folder is created under the `target_s3_arn` path for each database with the naming structure as `{db_id}.{family_guid}`.
- Within the folder, transaction log backups have a filename structure as `{db_id}.{family_guid}.{rds_backup_seq_id}.{backup_file_epoch}`.
- You can view the details of `family_guid,db_id,rds_backup_seq_id` and `backup_file_epoch` with the `rds_fn_list_tlog_backup_metadata`function.

The following example shows the folder and file structure of a set of transaction log backups within an Amazon S3 bucket.



Tracking the status of tasks

To track the status of your copy tasks, call the `rds_task_status` stored procedure. If you don't provide any parameters, the stored procedure returns the status of all tasks.

Example usage:

```
exec msdb.dbo.rds_task_status
  @db_name='database_name',
  @task_id=ID_number;
```

The following parameters are optional:

- `@db_name` – The name of the database to show the task status for.
- `@task_id` – The ID of the task to show the task status for.

Example of listing the status for a specific task ID:

```
exec msdb.dbo.rds_task_status @task_id=5;
```

Example of listing the status for a specific database and task:

```
exec msdb.dbo.rds_task_status@db_name='my_database',@task_id=5;
```

Example of listing all tasks and their status for a specific database:

```
exec msdb.dbo.rds_task_status @db_name='my_database';
```

Example of listing all tasks and their status on the current DB instance:

```
exec msdb.dbo.rds_task_status;
```

Canceling a task

To cancel a running task, call the `rds_cancel_task` stored procedure.

Example usage:

```
exec msdb.dbo.rds_cancel_task @task_id=ID_number;
```

The following parameter is required:

- `@task_id` – The ID of the task to cancel. You can view the task ID by calling the `rds_task_status` stored procedure.

For more information on viewing and canceling running tasks, see Importing and exporting SQL Server databases using native backup and restore.

Troubleshooting access to transaction log backups

The following are issues you might encounter when you use the stored procedures for access to transaction log backups.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_copy_setup	Backups are disabled on this DB instance. Enable DB instance backups with a retention of at least "1" and try again.	Automated backups are not enabled for the DB instance.	DB instance backup retention must be enabled with a retention of at least one day. For more information on enabling automated backups and configuring backup retention, see Backup retention period.
rds_tlog_copy_setup	Error running the rds_tlog_copy_setup stored procedure. Reconnect to the RDS endpoint and try again.	An internal error occurred.	Reconnect to the RDS endpoint and run the `rds_tlog_copy_setup` stored procedure again.
rds_tlog_copy_setup	Running the rds_tlog_backup_copy_setup stored procedure inside a transacti	The stored procedure was attempted within a transaction using `BEGIN` and `END`.	Avoid using `BEGIN` and `END` when running the `rds_tlog_copy_setup` stored procedure.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
	on is not supported . Verify that the session has no open transacti ons and try again.		
rds_tlog_ copy_setup	The S3 bucket name for the input parameter `@target_s 3_arn` should contain at least one character other than a space.	An incorrect value was provided for the input parameter `@target_s3_arn` .	Ensure the input parameter `@target_s3_arn` specifies the complete Amazon S3 bucket ARN.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_copy_setup	The SQLSERVER_BACKUP_RESTORE option isn't enabled or is in the process of being enabled. Enable the option or try again later.	The SQLSERVER_BACKUP_RESTORE option is not enabled on the DB instance or was just enabled and pending internal activation.	Enable the SQLSERVER_BACKUP_RESTORE option as specified in the Requirements section. Wait a few minutes and run the rds_tlog_copy_setup stored procedure again.
rds_tlog_copy_setup	The target S3 arn for the input parameter @target_s3_arn can't be empty or null.	An NULL value was provided for the input parameter @target_s3_arn , or the value wasn't provided.	Ensure the input parameter @target_s3_arn specifies the complete Amazon S3 bucket ARN.
rds_tlog_copy_setup	The target S3 arn for the input parameter @target_s3_arn must begin with arn:aws.	The input parameter @target_s3_arn was provide without arn:aws on the front.	Ensure the input parameter @target_s3_arn specifies the complete Amazon S3 bucket ARN.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_copy_setup	The target S3 ARN is already set to the provided value.	The `rds_tlog_copy_setup` stored procedure previously ran and was configured with an Amazon S3 bucket ARN.	To modify the Amazon S3 bucket value for access to transaction log backups, provide a different `target S3 ARN`.
rds_tlog_copy_setup	Unable to generate credentials for enabling Access to Transaction Log Backups. Confirm the S3 path ARN provided with `rds_tlog_copy_setup`, and try again later.	There was an unspecified error while generating credentials to enable access to transaction log backups.	Review your setup configuration and try again.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_copy_setup	You cannot run the rds_tlog_copy_setup stored procedure while there are pending tasks. Wait for the pending tasks to complete and try again.	Only two tasks may run at any time. There are pending tasks awaiting completion.	View pending tasks and wait for them to complete. For more information on monitoring task status, see Tracking the status of tasks.
rds_tlog_backup_copy_to_S3	A T-log backup file copy task has already been issued for database: %s with task Id: %d, please try again later.	Only one copy task may run at any time for a given database. There is a pending copy task awaiting completion.	View pending tasks and wait for them to complete. For more information on monitoring task status, see Tracking the status of tasks.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	At least one of these three parameter sets must be provided. SET-1:(@backup_file_start_time, @backup_file_end_time) | SET-2:(@starting_lsn, @ending_lsn) | SET-3:(@rds_backup_starting_seq_id, @rds_backup_ending_seq_id)	None of the three parameter sets were provided, or a provided parameter set is missing a required parameter.	You can specify either the time, lsn, or sequence ID parameters. One set from these three sets of parameters are required. For more information on required parameters, see Copying transaction log backups.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	Backups are disabled on your instance. Please enable backups and try again in some time.	Automated backups are not enabled for the DB instance.	For more information on enabling automated backups and configuring backup retention, see Backup retention period.
rds_tlog_backup_copy_to_S3	Cannot find the given database %s.	The value provided for input parameter `@db_name` does not match a database name on the DB instance.	Use the correct database name. To list all databases by name, run `SELECT * from sys.databases`
rds_tlog_backup_copy_to_S3	Cannot run the rds_tlog_backup_copy_to_S3 stored procedure for SQL Server system databases or the rdsadmin database.	The value provided for input parameter `@db_name` matches a SQL Server system database name or the RDSAdmin database.	The following databases are not allowed to be used with access to transaction log backups: `master, model, msdb, tempdb, RDSAdmin.`

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	Database name for the input parameter @db_name can't be empty or null.	The value provided for input parameter `@db_name` was empty or `NULL`.	Use the correct database name. To list all databases by name, run `SELECT * from sys.databases`
rds_tlog_backup_copy_to_S3	DB instance backup retention period must be set to at least 1 to run the rds_tlog_backup_copy_setup stored procedure.	Automated backups are not enabled for the DB instance.	For more information on enabling automated backups and configuring backup retention, see Backup retention period.
rds_tlog_backup_copy_to_S3	Error running the stored procedure rds_tlog_backup_copy_to_S3. Reconnect to the RDS endpoint and try again.	An internal error occurred.	Reconnect to the RDS endpoint and run the `rds_tlog_backup_copy_to_S3` stored procedure again.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	Only one of these three parameter sets can be provided. SET-1:(@backup_file_start_time, @backup_file_end_time) | SET-2:(@starting_lsn, @ending_lsn) | SET-3:(@rds_backup_starting_seq_id, @rds_backup_ending_seq_id)	Multiple parameter sets were provided.	You can specify either the time, lsn, or sequence ID parameters. One set from these three sets of parameters are required. For more information on required parameters, see Copying transaction log backups.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	Running the rds_tlog_backup_copy_to_S3 stored procedure inside a transaction is not supported. Verify that the session has no open transactions and try again.	The stored procedure was attempted within a transaction using `BEGIN` and `END`.	Avoid using `BEGIN` and `END` when running the `rds_tlog_backup_copy_to_S3` stored procedure.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	The provided parameters fall outside of the transaction backup log retention period. To list of available transaction log backup files, run the rds_fn_list_tlog_backup_metadata function.	There are no available transactional log backups for the provided input parameters that fit in the copy retention window.	Try again with a valid set of parameters. For more information on required parameters, see Copying transaction log backups.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	There was a permissions error in processing the request. Ensure the bucket is in the same Account and Region as the DB Instance, and confirm the S3 bucket policy permissions against the template in the public documentation.	There was an issue detected with the provided S3 bucket or its policy permissions.	Confirm your setup for access to transaction log backups is correct. For more information on setup requirements for your S3 bucket, see Requirements.

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	Running the `rds_tlog_backup_copy_to_S3` stored procedure on an RDS read replica instance isn't permitted.	The stored procedure was attempted on a RDS read replica instance.	Connect to the RDS primary DB instance to run the `rds_tlog_backup_copy_to_S3` stored procedure.
rds_tlog_backup_copy_to_S3	The LSN for the input parameter `@starting_lsn` must be less than `@ending_lsn .`	The value provided for input parameter `@starting_lsn` was greater than the value provided for input parameter `@ending_lsn .`	Ensure the value provided for input parameter `@starting_lsn` is less than the value provided for input parameter `@ending_lsn .`

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	The `rds_tlog_backup_copy_to_S3` stored procedure can only be performed by the members of `db_owner` role in the source database.	The `db_owner` role has not been granted for the account attempting to run the `rds_tlog_backup_copy_to_S3` stored procedure on the provided `db_name`.	Ensure the account running the stored procedure is permissioned with the `db_owner` role for the provided `db_name`.
rds_tlog_backup_copy_to_S3	The sequence ID for the input parameter `@rds_backup_starting_seq_id` must be less than or equal to `@rds_backup_ending_seq_id` .	The value provided for input parameter `@rds_backup_starting_seq_id` was greater than the value provided for input parameter `@rds_backup_ending_seq_id` .	Ensure the value provided for input parameter `@rds_backup_starting_seq_id` is less than the value provided for input parameter `@rds_backup_ending_seq_id` .

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	The SQLSERVER_BACKUP_RESTORE option isn't enabled or is in the process of being enabled. Enable the option or try again later.	The SQLSERVER_BACKUP_RESTORE option is not enabled on the DB instance or was just enabled and pending internal activation.	Enable the SQLSERVER_BACKUP_RESTORE option as specified in the Requirements section. Wait a few minutes and run the rds_tlog_backup_copy_to_S3 stored procedure again.
rds_tlog_backup_copy_to_S3	The start time for the input parameter @backup_file_start_time must be less than @backup_file_end_time .	The value provided for input parameter @backup_file_start_time was greater than the value provided for input parameter @backup_file_end_time .	Ensure the value provided for input parameter @backup_file_start_time is less than the value provided for input parameter @backup_file_end_time .

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	We were unable to process the request due to a lack of access. Please check your setup and permissions for the feature.	There may be an issue with the Amazon S3 bucket permissions, or the Amazon S3 bucket provided is in another account or Region.	Ensure the Amazon S3 bucket policy permissions are permissioned to allow RDS access. Ensure the Amazon S3 bucket is in the same account and Region as the DB instance.
rds_tlog_backup_copy_to_S3	You cannot provide a KMS Key ARN as input parameter to the stored procedure for instances that are not storage-encrypted.	When storage encryption is not enabled on the DB instance, the input parameter `@kms_key_arn` should not be provided.	Do not provide an input parameter for `@kms_key_arn` .

Stored Procedure	Error Message	Issue	Troubleshooting suggestions
rds_tlog_backup_copy_to_S3	You must provide a KMS Key ARN as input parameter to the stored procedure for storage encrypted instances.	When storage encryption is enabled on the DB instance, the input parameter `@kms_key_arn` must be provided.	Provide an input parameter for `@kms_key_arn` with a value that matches the ARN of the Amazon S3 bucket to use for transaction log backups.
rds_tlog_backup_copy_to_S3	You must run the `rds_tlog_copy_setup` stored procedure and set the `@target_s3_arn` , before running the `rds_tlog_backup_copy_to_S3` stored procedure.	The access to transaction log backups setup procedure was not completed before attempting to run the `rds_tlog_backup_copy_to_S3` stored procedure.	Run the `rds_tlog_copy_setup` stored procedure before running the `rds_tlog_backup_copy_to_S3` stored procedure . For more information on running the setup procedure for access to transaction log backups, see Setting up access to transaction log backups.

Options for the Microsoft SQL Server database engine

In this section, you can find descriptions for options that are available for Amazon RDS instances running the Microsoft SQL Server DB engine. To enable these options, you add them to an option group, and then associate the option group with your DB instance. For more information, see Working with option groups.

If you're looking for optional features that aren't added through RDS option groups (such as SSL, Microsoft Windows Authentication, and Amazon S3 integration), see Additional features for Microsoft SQL Server on Amazon RDS.

Amazon RDS supports the following options for Microsoft SQL Server DB instances.

Option	Option ID	Engine editions
Linked Servers with Oracle OLEDB	`OLEDB_ORACLE`	SQL Server Enterprise Edition SQL Server Standard Edition
Native backup and restore	`SQLSERVER_BACKUP_RESTORE`	SQL Server Enterprise Edition SQL Server Standard Edition SQL Server Web Edition SQL Server Express Edition
Transparent Data Encryption	`TRANSPARENT_DATA_ENCRYPTION` (RDS console)	SQL Server 2016–2022 Enterprise Edition SQL Server 2022 Standard Edition

Option	Option ID	Engine editions
	TDE (AWS CLI and RDS API)	
SQL Server Audit	SQLSERVER_AUDIT	In RDS, starting with SQL Server 2016, all editions of SQL Server support server-level audits, and Enterprise Edition also supports database-level audits. Starting with SQL Server SQL Server 2016 (13.x) SP1, all editions support both server-level and database-level audits. For more information, see SQL Server Audit (database engine) in the SQL Server documentation.
SQL Server Analysis Services	SSAS	SQL Server Enterprise Edition SQL Server Standard Edition

Option	Option ID	Engine editions
SQL Server Integration Services	SSIS	SQL Server Enterprise Edition SQL Server Standard Edition
SQL Server Reporting Services	SSRS	SQL Server Enterprise Edition SQL Server Standard Edition
Microsoft Distributed Transaction Coordinator	MSDTC	In RDS, starting with SQL Server 2016, all editions of SQL Server support distributed transactions.
SQL Server resource governor	RESOURCE_GOVERNOR	SQL Server Enterprise Edition SQL Server 2022 Developer Edition

Listing the available options for SQL Server versions and editions

You can use the `describe-option-group-options` AWS CLI command to list the available options for SQL Server versions and editions, and the settings for those options.

The following example shows the options and option settings for SQL Server 2019 Enterprise Edition. The `--engine-name` option is required.

```
aws rds describe-option-group-options --engine-name sqlserver-ee --major-engine-version
 15.00
```

The output resembles the following:

```
{
    "OptionGroupOptions": [
        {
            "Name": "MSDTC",
            "Description": "Microsoft Distributed Transaction Coordinator",
            "EngineName": "sqlserver-ee",
            "MajorEngineVersion": "15.00",
            "MinimumRequiredMinorEngineVersion": "4043.16.v1",
            "PortRequired": true,
            "DefaultPort": 5000,
            "OptionsDependedOn": [],
            "OptionsConflictsWith": [],
            "Persistent": false,
            "Permanent": false,
            "RequiresAutoMinorEngineVersionUpgrade": false,
            "VpcOnly": false,
            "OptionGroupOptionSettings": [
                {
                    "SettingName": "ENABLE_SNA_LU",
                    "SettingDescription": "Enable support for SNA LU protocol",
                    "DefaultValue": "true",
                    "ApplyType": "DYNAMIC",
                    "AllowedValues": "true,false",
                    "IsModifiable": true,
                    "IsRequired": false,
                    "MinimumEngineVersionPerAllowedValue": []
                },
        ...

        {
            "Name": "TDE",
            "Description": "SQL Server - Transparent Data Encryption",
            "EngineName": "sqlserver-ee",
            "MajorEngineVersion": "15.00",
            "MinimumRequiredMinorEngineVersion": "4043.16.v1",
            "PortRequired": false,
            "OptionsDependedOn": [],
            "OptionsConflictsWith": [],
            "Persistent": true,
            "Permanent": false,
            "RequiresAutoMinorEngineVersionUpgrade": false,
            "VpcOnly": false,
            "OptionGroupOptionSettings": []
```

```
        }
    ]
}
```

Support for Linked Servers with Oracle OLEDB in Amazon RDS for SQL Server

Linked servers with the Oracle Provider for OLEDB on RDS for SQL Server lets you access external data sources on an Oracle database. You can read data from remote Oracle data sources and run commands against remote Oracle database servers outside of your RDS for SQL Server DB instance. Using linked servers with Oracle OLEDB, you can:

- Directly access data sources other than SQL Server
- Query against diverse Oracle data sources with the same query without moving the data
- Issue distributed queries, updates, commands, and transactions on data sources across an enterprise ecosystem
- Integrate connections to an Oracle database from within the Microsoft Business Intelligence suite (SSIS, SSRS, SSAS)
- Migrate from an Oracle database to RDS for SQL Server

You can activate one or more linked servers for Oracle on either an existing or new RDS for SQL Server DB instance. Then you can integrate external Oracle data sources with your DB instance.

Contents

- Supported versions and Regions
- Limitations and recommendations
- Activating linked servers with Oracle
  - Creating the option group for OLEDB_ORACLE
  - Adding the OLEDB_ORACLE option to the option group
  - Modifying the OLEDB_ORACLE version option to another version
  - Associating the option group with your DB instance
- Modifying OLEDB provider properties
- Modifying OLEDB driver properties
- Deactivating linked servers with Oracle

Supported versions and Regions

RDS for SQL Server supports linked servers with Oracle OLEDB in all Regions for SQL Server Standard and Enterprise Editions on the following versions:

- SQL Server 2022, all versions
- SQL Server 2019, all versions
- SQL Server 2017, all versions

Linked servers with Oracle OLEDB is supported for the following Oracle Database versions:

- Oracle Database 21c, all versions
- Oracle Database 19c, all versions
- Oracle Database 18c, all versions

Linked servers with Oracle OLEDB is supported for the following OLEDB Oracle driver versions:

- 21.7
- 21.16

Limitations and recommendations

Keep in mind the following limitations and recommendations that apply to linked servers with Oracle OLEDB:

- Allow network traffic by adding the applicable TCP port in the security group for each RDS for SQL Server DB instance. For example, if you're configuring a linked server between an EC2 Oracle DB instance and an RDS for SQL Server DB instance, then you must allow traffic from the IP address of the EC2 Oracle DB instance. You also must allow traffic on the port that SQL Server is using to listen for database communication. For more information on security groups, see Controlling access with security groups.
- Perform a reboot of the RDS for SQL Server DB instance after turning on, turning off, or modifying the `OLEDB_ORACLE` option in your option group. The option group status displays `pending_reboot` for these events and is required. For RDS for SQL Server Multi-AZ instances with AlwaysOn or Mirroring option enabled, a failover is expected when instance is rebooted after the new instance creation or restore.

- Only simple authentication is supported with a user name and password for the Oracle data source.
- Open Database Connectivity (ODBC) drivers are not supported. Only the OLEDB driver versions listed above are supported.
- Distributed transactions (XA) are supported. To activate distributed transactions, turn on the `MSDTC` option in the Option Group for your DB instance and make sure XA transactions are turned on. For more information, see Support for Microsoft Distributed Transaction Coordinator in RDS for SQL Server.
- Creating data source names (DSNs) to use as a shortcut for a connection string is not supported.
- OLEDB driver tracing is not supported. You can use SQL Server Extended Events to trace OLEDB events. For more information, see Set up Extended Events in RDS for SQL Server.
- Access to the catalogs folder for an Oracle linked server is not supported using SQL Server Management Studio (SSMS).

Activating linked servers with Oracle

Activate linked servers with Oracle by adding the `OLEDB_ORACLE` option to your RDS for SQL Server DB instance. Use the following process:

1. Create a new option group, or choose an existing option group.
2. Add the `OLEDB_ORACLE` option to the option group.
3. Choose a version of the OLEDB driver to use.
4. Associate the option group with the DB instance.
5. Reboot the DB instance.

Creating the option group for OLEDB_ORACLE

To work with linked servers with Oracle, create an option group or modify an option group that corresponds to the SQL Server edition and version of the DB instance that you plan to use. To complete this procedure, use the AWS Management Console or the AWS CLI.

Console

The following procedure creates an option group for SQL Server Standard Edition 2019.

To create the option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** window, do the following:
   a. For **Name**, enter a name for the option group that is unique within your AWS account, such as **`oracle-oledb-se-2019`**. The name can contain only letters, digits, and hyphens.
   b. For **Description**, enter a brief description of the option group, such as **`OLEDB_ORACLE option group for SQL Server SE 2019`**. The description is used for display purposes.
   c. For **Engine**, choose **sqlserver-se**.
   d. For **Major engine version**, choose **15.00**.
5. Choose **Create**.

CLI

The following procedure creates an option group for SQL Server Standard Edition 2019.

To create the option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-option-group \
    --option-group-name oracle-oledb-se-2019 \
    --engine-name sqlserver-se \
    --major-engine-version 15.00 \
    --option-group-description "OLEDB_ORACLE option group for SQL Server SE 2019"
```

  For Windows:

```
aws rds create-option-group ^
```

```
    --option-group-name oracle-oledb-se-2019 ^
    --engine-name sqlserver-se ^
    --major-engine-version 15.00 ^
    --option-group-description "OLEDB_ORACLE option group for SQL Server SE 2019"
```

Adding the OLEDB_ORACLE option to the option group

Next, use the AWS Management Console or the AWS CLI to add the OLEDB_ORACLE option to your option group.

Console

To add the OLEDB_ORACLE option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group that you just created, which is **oracle-oledb-se-2019** in this example.
4. Choose **Add option**.
5. Under **Option details**, choose **OLEDB_ORACLE** for **Option name**.
6. Under **Version**, choose the version of the OLEDB Oracle driver you want to install.
7. Under **Scheduling**, choose whether to add the option immediately or at the next maintenance window.
8. Choose **Add option**.

CLI

To add the OLEDB_ORACLE option

- Add the OLEDB_ORACLE option to the option group.

  Example

  For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name oracle-oledb-se-2019 \
    --options OptionName=OLEDB_ORACLE, OptionVersion=21.16 \
```

```
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name oracle-oledb-se-2019 ^
    --options OptionName=OLEDB_ORACLE, OptionVersion=21.16 ^
    --apply-immediately
```

Modifying the OLEDB_ORACLE version option to another version

To modify the OLEDB_ORACLE option version to another version, use the AWS Management Console or the AWS CLI.

Console

To Modify the OLEDB_ORACLE option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group with the OLEDB_ORACLE option (**oracle-oledb-se-2019** in the previous example).
4. Choose **Modify option**.
5. Under **Option details**, choose **OLEDB_ORACLE** for **Option name**.
6. Under **Version**, choose the version of the OLEDB Oracle driver you want to use.
7. Under **Scheduling**, choose whether to modify the option immediately or at the next maintenance window.
8. Choose **Modify option**.

CLI

To modify the OLEDB_ORACLE option version, use the `rds add-option-to-option-group` AWS CLI command with the option group and option version that you want to use.

To modify the OLEDB_ORACLE option

- **Example**

  For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name oracle-oledb-se-2019 \
    --options OptionName=OLEDB_ORACLE, OptionVersion=21.7 \
    --apply-immediately
```

  For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name oracle-oledb-se-2019 ^
    --options OptionName=OLEDB_ORACLE, OptionVersion=21.7 ^
    --apply-immediately
```

Associating the option group with your DB instance

To associate the `OLEDB_ORACLE` option group and parameter group with your DB instance, use the AWS Management Console or the AWS CLI

Console

To finish activating linked servers for Oracle, associate your `OLEDB_ORACLE` option group with a new or existing DB instance:

- For a new DB instance, associate them when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, associate them by modifying the instance. For more information, see Modifying an Amazon RDS DB instance.

CLI

You can associate the `OLEDB_ORACLE` option group and parameter group with a new or existing DB instance.

To create an instance with the `OLEDB_ORACLE` option group and parameter group

- Specify the same DB engine type and major version that you used when creating the option group.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mytestsqlserveroracleoledbinstance \
    --db-instance-class db.m5.2xlarge \
    --engine sqlserver-se \
    --engine-version 15.0.4236.7.v1 \
    --allocated-storage 100 \
    --manage-master-user-password \
    --master-username admin \
    --storage-type gp2 \
    --license-model li \
    --domain-iam-role-name my-directory-iam-role \
    --domain my-domain-id \
    --option-group-name oracle-oledb-se-2019 \
    --db-parameter-group-name my-parameter-group-name
```

  For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mytestsqlserveroracleoledbinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-se ^
    --engine-version 15.0.4236.7.v1 ^
    --allocated-storage 100 ^
    --manage-master-user-password ^
    --master-username admin ^
    --storage-type gp2 ^
    --license-model li ^
    --domain-iam-role-name my-directory-iam-role ^
    --domain my-domain-id ^
    --option-group-name oracle-oledb-se-2019 ^
    --db-parameter-group-name my-parameter-group-name
```

To modify an instance and associate the `OLEDB_ORACLE` option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mytestsqlserveroracleoledbinstance \
    --option-group-name oracle-oledb-se-2019 \
    --db-parameter-group-name my-parameter-group-name \
    --apply-immediately
```

  For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mytestsqlserveroracleoledbinstance ^
    --option-group-name oracle-oledb-se-2019 ^
    --db-parameter-group-name my-parameter-group-name ^
    --apply-immediately
```

Modifying OLEDB provider properties

You can view and change the properties of the OLEDB provider. Only the `master` user can perform this task. All linked servers for Oracle that are created on the DB instance use the same properties of that OLEDB provider. Call the `sp_MSset_oledb_prop` stored procedure to change the properties of the OLEDB provider.

To change the OLEDB provider properties

```
USE [master]
GO
EXEC sp_MSset_oledb_prop N'OraOLEDB.Oracle', N'AllowInProcess', 1
EXEC sp_MSset_oledb_prop N'OraOLEDB.Oracle', N'DynamicParameters', 0
GO
```

The following properties can be modified:

Property name	Recommended Value (1 = On, 0 = Off)	Description
`Dynamic parameter`	1	Allows SQL placeholders (represented by '?') in parameterized queries.
`Nested queries`	1	Allows nested `SELECT` statements in the `FROM` clause, such as sub-queries.
`Level zero only`	0	Only base-level OLEDB interfaces are called against the provider.
`Allow inprocess`	1	If turned on, Microsoft SQL Server allows the provider to be instantiated as an in-process server. Set this property to 1 to use Oracle linked servers.
`Non transacted updates`	0	If non-zero, SQL Server allows updates.
`Index as access path`	False	If non-zero, SQL Server attempts to use indexes of the provider to fetch data.
`Disallow adhoc access`	False	If set, SQL Server does not allow running pass-through queries against the OLEDB provider. While this option can be checked, it is sometimes appropriate to run pass-through queries.
`Supports LIKE operator`	1	Indicates that the provider supports queries using the LIKE keyword.

Modifying OLEDB driver properties

You can view and change the properties of the OLEDB driver when creating a linked server for Oracle. Only the `master` user can perform this task. Driver properties define how the OLEDB driver handles data when working with a remote Oracle data source. Driver properties are specific to each

Oracle linked server created on the DB instance. Call the `master.dbo.sp_addlinkedserver` stored procedure to change the properties of the OLEDB driver.

Example: To create a linked server and change the OLEDB driver `FetchSize` property

```
EXEC master.dbo.sp_addlinkedserver
@server = N'Oracle_link2',
@srvproduct=N'Oracle',
@provider=N'OraOLEDB.Oracle',
@datasrc=N'my-oracle-test.cnetsipka.us-west-2.rds.amazonaws.com:1521/ORCL',
@provstr='FetchSize=200'
GO
```

```
EXEC master.dbo.sp_addlinkedsrvlogin
@rmtsrvname=N'Oracle_link2',
@useself=N'False',
@locallogin=NULL,
@rmtuser=N'master',
@rmtpassword='Test#1234'
GO
```

Note

Specify a password other than the prompt shown here as a security best practice.

Deactivating linked servers with Oracle

To deactivate linked servers with Oracle, remove the `OLEDB_ORACLE` option from its option group.

Important

Removing the option doesn't delete the existing linked server configurations on the DB instance. You must manually drop them to remove them from the DB instance.
You can reactivate the `OLEDB_ORACLE` option after removal to reuse the linked server configurations that were previously configured on the DB instance.

Console

The following procedure removes the OLEDB_ORACLE option.

To remove the OLEDB_ORACLE option from its option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group with the OLEDB_ORACLE option (`oracle-oledb-se-2019` in the previous examples).
4. Choose **Delete option**.
5. Under **Deletion options**, choose **OLEDB_ORACLE** for **Options to delete**.
6. Under **Apply immediately**, choose **Yes** to delete the option immediately, or **No** to delete it during the next maintenance window.
7. Choose **Delete**.

CLI

The following procedure removes the OLEDB_ORACLE option.

To remove the OLEDB_ORACLE option from its option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds remove-option-from-option-group \
    --option-group-name oracle-oledb-se-2019 \
    --options OLEDB_ORACLE \
    --apply-immediately
```

  For Windows:

```
aws rds remove-option-from-option-group ^
    --option-group-name oracle-oledb-se-2019 ^
    --options OLEDB_ORACLE ^
```

```
    --apply-immediately
```

Linked Servers with Teradata ODBC in RDS for SQL Server

Support for linked servers with the Teradata ODBC driver on RDS for SQL Server lets you access external data sources on a Teradata database. You can read data and run commands from remote Teradata database servers outside of your RDS for SQL Server instance. Use linked-servers with Teradata ODBC to enable the following capabilities:

- Directly access data sources other than SQL Server.
- Query against diverse Teradata data sources with the same query without moving the data.
- Issue distributed queries, updates, commands, and transactions on data sources across an enterprise ecosystem.
- Integrate connections to a Teradata database from within the Microsoft Business Intelligence Suite (SSIS, SSRS, SSAS).
- Migrate from a Teradata database to RDS for SQL Server.

You can choose to activate one or more linked servers for Teradata on either an existing or new RDS for SQL Server DB instance. You can then integrate external Teradata data sources with your DB instance.

Topics

- Supported versions and Regions
- Limitations and recommendations
- Considerations for Multi-AZ deployment
- Activating linked servers with Teradata
- Creating linked servers with Teradata
- Deactivating servers linked to Teradata

Supported versions and Regions

RDS for SQL Server supports linked servers with Teradata ODBC in all AWS Regions for SQL Server Standard and Enterprise Edition for the following versions:

- SQL Server 2022, all versions
- SQL Server 2019, all versions
- SQL Server 2017, all versions

The following Teradata database versions support linking with RDS for SQL Server

- Teradata 17.20, all versions

Limitations and recommendations

The following limitations apply to linked servers with Teradata ODBC:

- RDS for SQL Server support only simple authentication with a username and password for the Teradata source.
- RDS for SQL Server supports only Teradata ODBC driver version 17.20.0.33.
- RDS for SQL Server does not support creating data source names (DSNs) to use as shortcuts for a connection string.
- RDS for SQL Server does not support ODBC driver tracing. Use SQL Server Extended Events to trace ODBC events. For more information, see Set up Extended Events in RDS for SQL Server.
- RDS for SQL Server does not support access to the catalogs folder for a Teradata linked server when using SQL Server Management Studio (SSMS).

Consider the following recommendations when using linked servers with Teradata ODBC:

- Allow network traffic by adding the applicable TCP port in the security group for each RDS for SQL Server DB instance. If you're configuring a linked server between an EC2 Teradata DB instance and an RDS for SQL Server DB instance, then you must allow traffic from the IP address of the EC2 Teradata DB instance. You also must allow traffic on the port that the RDS for SQL Server DB instance is using to listen for database communication. For more information on security groups, see Controlling access with security groups.
- Distributed transactions (XA) are supported. To activate distributed transactions, turn on the `MSDTC` option in the option group for your DB instance and make sure XA transactions are turned on. For more information, see Support for Microsoft Distributed Transaction Coordinator in RDS for SQL Server.
- Linked Teradata ODBC support SSL/TLS as long as configured on the Teradata Server. For more information, see Enable TLS Connectivity on Teradata Vantage.

Considerations for Multi-AZ deployment

RDS for SQL Server currently doesn't replicate linked servers to the mirrored database server (or Always-On availability group secondary server) in a Multi-AZ deployment. If the linked servers are added before the configuration is changed to add mirroring or Always-On, then the linked servers are copied for the existing linked servers.

Alternatively, you can create the linked servers on the primary instance, fail over to the high availability server instance and then create the linked servers again so that they are on both RDS for SQL Server instances.

Activating linked servers with Teradata

Activate linked servers with Teradata by adding the `ODBC_TERADATA` option to your RDS for SQL Server DB instance. Use the following process:

Topics

- Creating the option group for ODBC_TERADATA
- Adding the ODBC_TERADATA option to the option group
- Associating the ODBC_TERADATA option with your DB instance

Creating the option group for `ODBC_TERADATA`

To work with linked servers with Teradata, create an option group or modify an option group that corresponds to the SQL Server eddition and version of the DB instance that you plan to use. To complete this procedure, use the AWS Management Console or the AWS CLI.

Console

Use the following procedure to create an option group for SQL Server Standard Edition 2019.

To create the option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** window, do the following:

a. For **Name**, enter a name for the option group that is unique within your AWS account, such as `teradata-odbc-se-2019`. The name can contain only letters, digits, and hyphens.

b. For **Description**, enter a brief description of the option group.

c. For **Engine**, choose **sqlserver-se**.

d. For **Major engine version**, choose **15.00**.

5. Choose **Create**.

AWS CLI

The following procedure creates an option group for SQL Server Standard Edition 2019.

Example

For Linux, macOS, or Unix:

```
aws rds create-option-group \
    --option-group-name teradata-odbc-se-2019 \
    --engine-name sqlserver-se \
    --major-engine-version 15.00 \
    --option-group-description "ODBC_TERADATA option group for SQL Server SE 2019"
```

Example

For Windows:

```
aws rds create-option-group ^
    --option-group-name teradata-odbc-se-2019 ^
    --engine-name sqlserver-se ^
    --major-engine-version 15.00 ^
    --option-group-description "ODBC_TERADATA option group for SQL Server SE 2019"
```

Adding the `ODBC_TERADATA` option to the option group

Next, use the AWS Management Console or the AWS CLI to add the `ODBC_Teradata` option to your option group.

Console

Use the following procedure creates an option group for SQL Server Standard Edition 2019.

To add the ODBC_TERADATA option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose your new option group.
4. Choose **Add option**.
5. Under **Option details**:
   a. Choose **ODBC_TERADATA** for **Option name**.
   b. For `17.20.33.00` for **Option version**.
6. Under scheduling, choose whether to add the option immediately or at the next maintenance window.
7. Choose **Add option**.

AWS CLI

The following procedure adds the ODBC_TERADATA option to your option group.

Example

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name teradata-odbc-se-2019 \
    --options "OptionName=ODBC_TERADATA,OptionVersion=17.20.33.00" \
    --apply-immediately
```

Example

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name teradata-odbc-se-2019 ^
    --options "OptionName=ODBC_TERADATA,OptionVersion=17.20.33.00" ^
    --apply-immediately
```

Associating the ODBC_TERADATA option with your DB instance

To associate the ODBC_TERADATA option group with your DB instance, use the AWS Management Console or AWS CLI.

Console

To finish activating linked servers for Teradata, associate your option group with a new or existing DB instance:

- For a new DB instance, associate it when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, associate it by modifying the instance. For more information, see Modifying an Amazon RDS DB instance.

AWS CLI

Specify the same DB engine type and major version that you used when creating the option group.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mytestsqlserverteradataodbcinstance \
    --db-instance-class db.m5.2xlarge \
    --engine sqlserver-se \
    --engine-version 15.00 \
    --license-model license-included \
    --allocated-storage 100 \
    --master-username admin \
    --master-user-password password \
    --storage-type gp2 \
    --option-group-name teradata-odbc-se-2019
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mytestsqlserverteradataodbcinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-se ^
    --engine-version 15.00 ^
```

```
    --license-model license-included ^
    --allocated-storage 100 ^
    --master-username admin ^
    --master-user-password password ^
    --storage-type gp2 ^
    --option-group-name teradata-odbc-se-2019
```

To modify an instance and associate the new option group:

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mytestsqlserverteradataodbcinstance \
    --option-group-name teradata-odbc-se-2019 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mytestsqlserverteradataodbcinstance ^
    --option-group-name teradata-odbc-se-2019 ^
    --apply-immediately
```

Creating linked servers with Teradata

To create a linked server with Teradata, run the following commands:

```
EXECUTE master.dbo.sp_addlinkedserver
    @server = N'LinkedServer_NAME',
    @srvproduct=N'',
    @provider=N'MSDASQL',
    @provstr=N'"PROVIDER=MSDASQL;DRIVER={Teradata Database ODBC Driver 17.20};
                DBCName=Server;UID=user_name;PWD=user_password;
                UseDataEncryption=YES/NO;SSLMODE=PREFER/ALLOW/DISABLE>;"',
    @catalog='database'
```

```
EXECUTE master.dbo.sp_addlinkedsrvlogin
    @rmtsrvname = N'LinkedServer_NAME',
    @locallogin = NULL ,
    @useself = N'False',
    @rmtuser = N'user_name',
```

```
    @rmtpassword = N'user_password'
```

An example of the the commands above are shown here:

```
EXECUTE master.dbo.sp_addlinkedserver
    @server = N'LinkedServerToTeradata',
    @srvproduct=N'',
    @provider=N'MSDASQL',
    @provstr=N'"PROVIDER=MSDASQL;DRIVER={Teradata Database ODBC Driver 17.20};
                DBCName=my-teradata-test.cnetsipka.us-west-2.rds.amazonaws.com;
                UID=master;
                PWD=Test#1234;
                UseDataEncryption=YES;
                SSLMODE=PREFER;"',
    @catalog='MyTestTeradataDB'

EXECUTE master.dbo.sp_addlinkedsrvlogin
    @rmtsrvname = N'LinkedServerToTeradata',
    @locallogin = NULL ,
    @useself = N'False',
    @rmtuser = N'master',
    @rmtpassword = N'Test#1234'
```

Note

Specify a password other than the prompt shown here as a security best practice.

Deactivating servers linked to Teradata

To deactivate linked servers to Teradata, remove the `ODBC_TERADATA` option from its option group.

Important

Deleting the option doesn't delete the linked server configurations on the DB instance. You must manually drop them to remove them from the DB instance.
You can reactivate the `ODBC_TERADATA` after removal to reuse the linked server configurations that were previously configured on the DB instance.

Console

To remove the ODBC_TERADATA option from the option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group with the ODBC_TERADATA option.
4. Choose **Delete**.
5. Under **Deletion options**, choose ODBC_TERADATA under **Options to delete**.
6. Under **Apply immediately**, choose **Yes** to delete the option immediately, or **No** to delete it during the next maintenance window.
7. Choose **Delete**.

AWS CLI

The following commands removes the ODBC_TERADATA option.

For Linux, macOS, or Unix:

```
aws rds remove-option-from-option-group \
    --option-group-name teradata-odbc-se-2019 \
    --options ODBC_TERADATA \
    --apply-immediately
```

For Windows:

```
aws rds remove-option-from-option-group ^
    --option-group-name teradata-odbc-se-2019 ^
    --options ODBC_TERADATA ^
    --apply-immediately
```

Support for native backup and restore in SQL Server

By using native backup and restore for SQL Server databases, you can create a differential or full backup of your on-premises database and store the backup files on Amazon S3. You can then restore to an existing Amazon RDS DB instance running SQL Server. You can also back up an RDS for SQL Server database, store it on Amazon S3, and restore it in other locations. In addition, you can restore the backup to an on-premises server, or a different Amazon RDS DB instance running SQL Server. For more information, see Importing and exporting SQL Server databases using native backup and restore.

Amazon RDS supports native backup and restore for Microsoft SQL Server databases by using differential and full backup files (.bak files).

Adding the native backup and restore option

The general process for adding the native backup and restore option to a DB instance is the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the `SQLSERVER_BACKUP_RESTORE` option to the option group.
3. Associate an AWS Identity and Access Management (IAM) role with the option. The IAM role must have access to an S3 bucket to store the database backups.

   That is, the option must have as its option setting a valid Amazon Resource Name (ARN) in the format `arn:aws:iam::`*`account-id`*`:role/`*`role-name`*. For more information, see Amazon Resource Names (ARNs) in the *AWS General Reference.*

   The IAM role must also have a trust relationship and a permissions policy attached. The trust relationship allows RDS to assume the role, and the permissions policy defines the actions that the role can perform. For more information, see Manually creating an IAM role for native backup and restore.
4. Associate the option group with the DB instance.

After you add the native backup and restore option, you don't need to restart your DB instance. As soon as the option group is active, you can begin backing up and restoring immediately.

Console

To add the native backup and restore option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Create a new option group or use an existing option group. For information on how to create a custom DB option group, see Creating an option group.

   To use an existing option group, skip to the next step.
4. Add the **SQLSERVER_BACKUP_RESTORE** option to the option group. For more information about adding options, see Adding an option to an option group.
5. Do one of the following:

   - To use an existing IAM role and Amazon S3 settings, choose an existing IAM role for **IAM Role**. If you use an existing IAM role, RDS uses the Amazon S3 settings configured for this role.
   - To create a new role and configure Amazon S3 settings, do the following:

     1. For **IAM role**, choose **Create a new role**.
     2. For **S3 bucket**, choose an S3 bucket from the list.
     3. For **S3 prefix (optional)**, specify a prefix to use for the files stored in your Amazon S3 bucket.

        This prefix can include a file path but doesn't have to. If you provide a prefix, RDS attaches that prefix to all backup files. RDS then uses the prefix during a restore to identify related files and ignore irrelevant files. For example, you might use the S3 bucket for purposes besides holding backup files. In this case, you can use the prefix to have RDS perform native backup and restore only on a particular folder and its subfolders.

        If you leave the prefix blank, then RDS doesn't use a prefix to identify backup files or files to restore. As a result, during a multiple-file restore, RDS attempts to restore every file in every folder of the S3 bucket.
     4. Choose the **Enable encryption** check box to encrypt the backup file. Leave the check box cleared (the default) to have the backup file unencrypted.

If you chose **Enable encryption**, choose an encryption key for **AWS KMS key**. For more information about encryption keys, see Getting started in the *AWS Key Management Service Developer Guide.*

6. Choose **Add option**.
7. Apply the option group to a new or existing DB instance:
   - For a new DB instance, apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

CLI

This procedure makes the following assumptions:

- You're adding the SQLSERVER_BACKUP_RESTORE option to an option group that already exists. For more information about adding options, see Adding an option to an option group.
- You're associating the option with an IAM role that already exists and has access to an S3 bucket to store the backups.
- You're applying the option group to a DB instance that already exists. For more information, see Modifying an Amazon RDS DB instance.

To add the native backup and restore option

1. Add the SQLSERVER_BACKUP_RESTORE option to the option group.

   Example

   For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
 --apply-immediately \
 --option-group-name mybackupgroup \
 --options "OptionName=SQLSERVER_BACKUP_RESTORE, \
   OptionSettings=[{Name=IAM_ROLE_ARN,Value=arn:aws:iam::account-id:role/role-
name}]"
```

For Windows:

```
aws rds add-option-to-option-group ^
 --option-group-name mybackupgroup ^
 --options "[{\"OptionName\": \"SQLSERVER_BACKUP_RESTORE\", ^
 \"OptionSettings\": [{\"Name\": \"IAM_ROLE_ARN\", ^
 \"Value\": \"arn:aws:iam::account-id:role/role-name"}]}]" ^
 --apply-immediately
```

> **Note**
>
> When using the Windows command prompt, you must escape double quotes (") in JSON code by prefixing them with a backslash (\).

2. Apply the option group to the DB instance.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
 --db-instance-identifier mydbinstance \
 --option-group-name mybackupgroup \
 --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
 --db-instance-identifier mydbinstance ^
 --option-group-name mybackupgroup ^
 --apply-immediately
```

Modifying native backup and restore option settings

After you enable the native backup and restore option, you can modify the settings for the option. For more information about how to modify option settings, see Modifying an option setting.

Removing the native backup and restore option

You can turn off native backup and restore by removing the option from your DB instance. After you remove the native backup and restore option, you don't need to restart your DB instance.

To remove the native backup and restore option from a DB instance, do one of the following:

- Remove the option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
- Modify the DB instance and specify a different option group that doesn't include the native backup and restore option. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Support for Transparent Data Encryption in SQL Server

Amazon RDS supports using Transparent Data Encryption (TDE) to encrypt stored data on your DB instances running Microsoft SQL Server. TDE automatically encrypts data before it is written to storage, and automatically decrypts data when the data is read from storage.

Amazon RDS supports TDE for the following SQL Server versions and editions:

- SQL Server 2022 Standard and Enterprise Editions
- SQL Server 2019 Standard and Enterprise Editions
- SQL Server 2017 Enterprise Edition
- SQL Server 2016 Enterprise Edition

> **Note**
>
> RDS for SQL Server does not support TDE for read-only databases.

Transparent Data Encryption for SQL Server provides encryption key management by using a two-tier key architecture. A certificate, which is generated from the database master key, is used to protect the data encryption keys. The database encryption key performs the actual encryption and decryption of data on the user database. Amazon RDS backs up and manages the database master key and the TDE certificate.

Transparent Data Encryption is used in scenarios where you need to encrypt sensitive data. For example, you might want to provide data files and backups to a third party, or address security-related regulatory compliance issues. You can't encrypt the system databases for SQL Server, such as the `model` or `master` databases.

A detailed discussion of Transparent Data Encryption is beyond the scope of this guide, but make sure that you understand the security strengths and weaknesses of each encryption algorithm and key. For information about Transparent Data Encryption for SQL Server, see Transparent Data Encryption (TDE) in the Microsoft documentation.

Topics

- Turning on TDE for RDS for SQL Server
- Encrypting data on RDS for SQL Server

- Backing up and restoring TDE certificates on RDS for SQL Server
- Backing up and restoring TDE certificates for on-premises databases
- Turning off TDE for RDS for SQL Server

Turning on TDE for RDS for SQL Server

To turn on Transparent Data Encryption for an RDS for SQL Server DB instance, specify the TDE option in an RDS option group that's associated with that DB instance:

1. Determine whether your DB instance is already associated with an option group that has the TDE option. To view the option group that a DB instance is associated with, use the RDS console, the describe-db-instance AWS CLI command, or the API operation DescribeDBInstances.
2. If the DB instance isn't associated with an option group that has TDE turned on, you have two choices. You can create an option group and add the TDE option, or you can modify the associated option group to add it.

   > **Note**
   >
   > In the RDS console, the option is named `TRANSPARENT_DATA_ENCRYPTION`. In the AWS CLI and RDS API, it's named `TDE`.

   For information about creating or modifying an option group, see Working with option groups. For information about adding an option to an option group, see Adding an option to an option group.
3. Associate the DB instance with the option group that has the TDE option. For information about associating a DB instance with an option group, see Modifying an Amazon RDS DB instance.

Option group considerations

The TDE option is a persistent option. You can't remove it from an option group unless all DB instances and backups are no longer associated with the option group. After you add the TDE option to an option group, the option group can be associated only with DB instances that use TDE. For more information about persistent options in an option group, see Option groups overview.

Because the TDE option is a persistent option, you can have a conflict between the option group and an associated DB instance. You can have a conflict in the following situations:

- The current option group has the TDE option, and you replace it with an option group that doesn't have the TDE option.
- You restore from a DB snapshot to a new DB instance that doesn't have an option group that contains the TDE option. For more information about this scenario, see Considerations for option groups.

SQL Server performance considerations

Using Transparent Data Encryption can affect the performance of a SQL Server DB instance.

Performance for unencrypted databases can also be degraded if the databases are on a DB instance that has at least one encrypted database. As a result, we recommend that you keep encrypted and unencrypted databases on separate DB instances.

Encrypting data on RDS for SQL Server

When the TDE option is added to an option group, Amazon RDS generates a certificate that's used in the encryption process. You can then use the certificate to run SQL statements that encrypt data in a database on the DB instance.

The following example uses the RDS-created certificate called `RDSTDECertificateName` to encrypt a database called `myDatabase`.

```
---------- Turning on TDE -------------

-- Find an RDS TDE certificate to use
USE [master]
GO
SELECT name FROM sys.certificates WHERE name LIKE 'RDSTDECertificate%'
GO

USE [myDatabase]
GO
-- Create a database encryption key (DEK) using one of the certificates from the
 previous step
CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE [RDSTDECertificateName]
GO

-- Turn on encryption for the database
ALTER DATABASE [myDatabase] SET ENCRYPTION ON
```

```
GO

-- Verify that the database is encrypted
USE [master]
GO
SELECT name FROM sys.databases WHERE is_encrypted = 1
GO
SELECT db_name(database_id) as DatabaseName, * FROM sys.dm_database_encryption_keys
GO
```

The time that it takes to encrypt a SQL Server database using TDE depends on several factors. These include the size of the DB instance, whether the instance uses Provisioned IOPS storage, the amount of data, and other factors.

Backing up and restoring TDE certificates on RDS for SQL Server

RDS for SQL Server provides stored procedures for backing up, restoring, and dropping TDE certificates. RDS for SQL Server also provides a function for viewing restored user TDE certificates.

User TDE certificates are used to restore databases to RDS for SQL Server that are on-premises and have TDE turned on. These certificates have the prefix `UserTDECertificate_`. After restoring databases, and before making them available to use, RDS modifies the databases that have TDE turned on to use RDS-generated TDE certificates. These certificates have the prefix `RDSTDECertificate`.

User TDE certificates remain on the RDS for SQL Server DB instance, unless you drop them using the `rds_drop_tde_certificate` stored procedure. For more information, see Dropping restored TDE certificates.

You can use a user TDE certificate to restore other databases from the source DB instance. The databases to restore must use the same TDE certificate and have TDE turned on. You don't have to import (restore) the same certificate again.

Topics

- Prerequisites
- Limitations
- Backing up a TDE certificate
- Restoring a TDE certificate
- Viewing restored TDE certificates

- Dropping restored TDE certificates

Prerequisites

Before you can back up or restore TDE certificates on RDS for SQL Server, make sure to perform the following tasks. The first three are described in Setting up for native backup and restore.

1. Create Amazon S3 general purpose buckets or directory buckets for storing files to back up and restore.

   We recommend that you use separate buckets for database backups and for TDE certificate backups.
2. Create an IAM role for backing up and restoring files.

   The IAM role must be both a user and an administrator for the AWS KMS key.

   When using directory buckets, no additional permissions are required other than the permissions required for Manually creating an IAM role for native backup and restore with directory buckets.

   When using S3 resources, the IAM role also requires the following permissions in addition to the permissions required for Manually creating an IAM role for native backup and restore:

   - `s3:GetBucketAcl`, `s3:GetBucketLocation`, and `s3:ListBucket` on the S3 bucket resource
3. Add the `SQLSERVER_BACKUP_RESTORE` option to an option group on your DB instance.

   This is in addition to the `TRANSPARENT_DATA_ENCRYPTION` (`TDE`) option.
4. Make sure that you have a symmetric encryption KMS key. You have the following options:
   - If you have an existing KMS key in your account, you can use it. No further action is necessary.
   - If you don't have an existing symmetric encryption KMS key in your account, create a KMS key by following the instructions in Creating keys in the *AWS Key Management Service Developer Guide*.
5. Enable Amazon S3 integration to transfer files between the DB instance and Amazon S3.

   For more information on enabling Amazon S3 integration, see Integrating an Amazon RDS for SQL Server DB instance with Amazon S3.

   Note that directory buckets are not supported for S3 integration. This step is only required for Backing up and restoring TDE certificates for on-premises databases.

Limitations

Using stored procedures to back up and restore TDE certificates has the following limitations:

- Both the SQLSERVER_BACKUP_RESTORE and TRANSPARENT_DATA_ENCRYPTION (TDE) options must be added to the option group that you associated with your DB instance.
- TDE certificate backup and restore aren't supported on Multi-AZ DB instances.
- Canceling TDE certificate backup and restore tasks isn't supported.
- You can't use a user TDE certificate for TDE encryption of any other database on your RDS for SQL Server DB instance. You can use it to restore only other databases from the source DB instance that have TDE turned on and that use the same TDE certificate.
- You can drop only user TDE certificates.
- The maximum number of user TDE certificates supported on RDS is 10. If the number exceeds 10, drop unused TDE certificates and try again.
- The certificate name can't be empty or null.
- When restoring a certificate, the certificate name can't include the keyword RDSTDECERTIFICATE, and must start with the UserTDECertificate_ prefix.
- The @certificate_name parameter can include only the following characters: a-z, 0-9, @, $, #, and underscore (_).
- The file extension for @certificate_file_s3_arn must be .cer (case-insensitive).
- The file extension for @private_key_file_s3_arn must be .pvk (case-insensitive).
- The S3 metadata for the private key file must include the x-amz-meta-rds-tde-pwd tag. For more information, see Backing up and restoring TDE certificates for on-premises databases.
- RDS for SQL Server does not support using cross-account keys for TDE.

Backing up a TDE certificate

To back up TDE certificates, use the rds_backup_tde_certificate stored procedure. It has the following syntax.

```
EXECUTE msdb.dbo.rds_backup_tde_certificate
    @certificate_name='UserTDECertificate_certificate_name |
 RDSTDECertificatetimestamp',
    @certificate_file_s3_arn='arn:aws:s3:::bucket_name/certificate_file_name.cer',
    @private_key_file_s3_arn='arn:aws:s3:::bucket_name/key_file_name.pvk',
    @kms_password_key_arn='arn:aws:kms:region:account-id:key/key-id',
```

```
    [@overwrite_s3_files=0|1];
```

The following parameters are required:

- `@certificate_name` – The name of the TDE certificate to back up.
- `@certificate_file_s3_arn` – The destination Amazon Resource Name (ARN) for the certificate backup file in Amazon S3.
- `@private_key_file_s3_arn` – The destination S3 ARN of the private key file that secures the TDE certificate.
- `@kms_password_key_arn` – The ARN of the symmetric KMS key used to encrypt the private key password.

The following parameter is optional:

- `@overwrite_s3_files` – Indicates whether to overwrite the existing certificate and private key files in S3:
  - `0` – Doesn't overwrite the existing files. This value is the default.

    Setting `@overwrite_s3_files` to 0 returns an error if a file already exists.
  - 1 – Overwrites an existing file that has the specified name, even if it isn't a backup file.

Example of backing up a TDE certificate

```
EXECUTE msdb.dbo.rds_backup_tde_certificate
    @certificate_name='RDSTDECertificate20211115T185333',
    @certificate_file_s3_arn='arn:aws:s3:::TDE_certs/mycertfile.cer',
    @private_key_file_s3_arn='arn:aws:s3:::TDE_certs/mykeyfile.pvk',
    @kms_password_key_arn='arn:aws:kms:us-
west-2:123456789012:key/AKIAIOSFODNN7EXAMPLE',
    @overwrite_s3_files=1;
```

Restoring a TDE certificate

You use the `rds_restore_tde_certificate` stored procedure to restore (import) user TDE certificates. It has the following syntax.

```
EXECUTE msdb.dbo.rds_restore_tde_certificate
    @certificate_name='UserTDECertificate_certificate_name',
```

```
    @certificate_file_s3_arn='arn:aws:s3:::bucket_name/certificate_file_name.cer',
    @private_key_file_s3_arn='arn:aws:s3:::bucket_name/key_file_name.pvk',
    @kms_password_key_arn='arn:aws:kms:region:account-id:key/key-id';
```

The following parameters are required:

- `@certificate_name` – The name of the TDE certificate to restore. The name must start with the `UserTDECertificate_` prefix.
- `@certificate_file_s3_arn` – The S3 ARN of the backup file used to restore the TDE certificate.
- `@private_key_file_s3_arn` – The S3 ARN of the private key backup file of the TDE certificate to be restored.
- `@kms_password_key_arn` – The ARN of the symmetric KMS key used to encrypt the private key password.

Example of restoring a TDE certificate

```
EXECUTE msdb.dbo.rds_restore_tde_certificate
    @certificate_name='UserTDECertificate_myTDEcertificate',
    @certificate_file_s3_arn='arn:aws:s3:::TDE_certs/mycertfile.cer',
    @private_key_file_s3_arn='arn:aws:s3:::TDE_certs/mykeyfile.pvk',
    @kms_password_key_arn='arn:aws:kms:us-
west-2:123456789012:key/AKIAIOSFODNN7EXAMPLE';
```

Viewing restored TDE certificates

You use the `rds_fn_list_user_tde_certificates` function to view restored (imported) user TDE certificates. It has the following syntax.

```
SELECT * FROM msdb.dbo.rds_fn_list_user_tde_certificates();
```

The output resembles the following. Not all columns are shown here.

```
name   certif princi pvt_ke issuer cert_s thumbp subjec start_ expiry pvt_key_l
       te_id  _id    ncrypt me     al_num t      t       e      te     ast_backu
                     _type_                                            p_date
                     c
```

UserTDrtific_tde_c	343	1	ENCRYP_BY_MAR_KEY	AnyCony Shippi	793e57a369fd1d9e472c32671d9cca af	0x6BB2341103 80B FE1BA2C69509 5B5	AnyCony Shippi	2022-05 19:49:000000	2023-05 19:49:000000	NULL

Dropping restored TDE certificates

To drop restored (imported) user TDE certificates that you aren't using, use the `rds_drop_tde_certificate` stored procedure. It has the following syntax.

```
EXECUTE msdb.dbo.rds_drop_tde_certificate
 @certificate_name='UserTDECertificate_certificate_name';
```

The following parameter is required:

- `@certificate_name` – The name of the TDE certificate to drop.

You can drop only restored (imported) TDE certificates. You can't drop RDS-created certificates.

Example of dropping a TDE certificate

```
EXECUTE msdb.dbo.rds_drop_tde_certificate
 @certificate_name='UserTDECertificate_myTDEcertificate';
```

Backing up and restoring TDE certificates for on-premises databases

You can back up TDE certificates for on-premises databases, then later restore them to RDS for SQL Server. You can also restore an RDS for SQL Server TDE certificate to an on-premises DB instance.

> **Note**
>
> RDS for SQL Server does not support using cross-account keys for TDE.

The following procedure backs up a TDE certificate and private key. The private key is encrypted using a data key generated from your symmetric encryption KMS key.

To back up an on-premises TDE certificate

1. Generate the data key using the AWS CLI generate-data-key command.

```
aws kms generate-data-key \
    --key-id my_KMS_key_ID \
    --key-spec AES_256
```

   The output resembles the following.

```
{
"CiphertextBlob": "AQIDAHimL2NEoAlOY6Bn7LJfnxi/OZe9kTQo/
XQXduug1rmerwGiL7g5ux4av9GfZLxYTDATAAAAfjB8BgkqhkiG9w0B
BwagbzBtAgEAMGgGCSqGSIb3DQEHATAeBglghkgBZQMEAS4wEQQMyCxLMi7GRZgKqD65AgEQgDtjvZLJo2cQ31Vetng
2RezQy3sAS6ZHrCjfnfn0c65bFdhsXxjSMnudIY7AKw==",
"Plaintext": "U/fpGtmzGCYBi8A2+0/9qcRQRK2zmG/aOn939ZnKi/0=",
"KeyId": "arn:aws:kms:us-west-2:123456789012:key/1234abcd-00ee-99ff-88dd-
aa11bb22cc33"
}
```

   You use the plain text output in the next step as the private key password.

2. Back up your TDE certificate as shown in the following example.

```
BACKUP CERTIFICATE myOnPremTDEcertificate TO FILE = 'D:\tde-cert-backup.cer'
WITH PRIVATE KEY (
```

```
FILE = 'C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\MSSQL\DATA\cert-backup-key.pvk',
ENCRYPTION BY PASSWORD = 'U/fpGtmzGCYBi8A2+0/9qcRQRK2zmG/aOn939ZnKi/0=');
```

3. Save the certificate backup file to your Amazon S3 certificate bucket.
4. Save the private key backup file to your S3 certificate bucket, with the following tag in the file's metadata:

   - Key – `x-amz-meta-rds-tde-pwd`
   - Value – The `CiphertextBlob` value from generating the data key, as in the following example.

   ```
   AQIDAHimL2NEoAlOY6Bn7LJfnxi/OZe9kTQo/
   XQXduug1rmerwGiL7g5ux4av9GfZLxYTDATAAAAfjB8BgkqhkiG9w0B
   BwagbzBtAgEAMGgGCSqGSIb3DQEHATAeBglghkgBZQMEAS4wEQQMyCxLMi7GRZgKqD65AgEQgDtjvZLJo2cQ31Vet
   2RezQy3sAS6ZHrCjfnfn0c65bFdhsXxjSMnudIY7AKw==
   ```

The following procedure restores an RDS for SQL Server TDE certificate to an on-premises DB instance. You copy and restore the TDE certificate on your destination DB instance using the certificate backup, corresponding private key file, and data key. The restored certificate is encrypted by the database master key of the new server.

To restore a TDE certificate

1. Copy the TDE certificate backup file and private key file from Amazon S3 to the destination instance. For more information on copying files from Amazon S3, see Transferring files between RDS for SQL Server and Amazon S3.
2. Use your KMS key to decrypt the output cipher text to retrieve the plain text of the data key. The cipher text is located in the S3 metadata of the private key backup file.

   ```
   aws kms decrypt \
       --key-id my_KMS_key_ID \
       --ciphertext-blob fileb://exampleCiphertextFile | base64 -d \
       --output text \
       --query Plaintext
   ```

   You use the plain text output in the next step as the private key password.
3. Use the following SQL command to restore your TDE certificate.

```
CREATE CERTIFICATE myOnPremTDEcertificate FROM FILE='D:\tde-cert-backup.cer'
WITH PRIVATE KEY (FILE = N'D:\tde-cert-key.pvk',
DECRYPTION BY PASSWORD = 'plain_text_output');
```

For more information on KMS decryption, see decrypt in the KMS section of the *AWS CLI Command Reference*.

After the TDE certificate is restored on the destination DB instance, you can restore encrypted databases with that certificate.

> **Note**
>
> You can use the same TDE certificate to encrypt multiple SQL Server databases on the source DB instance. To migrate multiple databases to a destination instance, copy the TDE certificate associated with them to the destination instance only once.

Turning off TDE for RDS for SQL Server

To turn off TDE for an RDS for SQL Server DB instance, first make sure that there are no encrypted objects left on the DB instance. To do so, either decrypt the objects or drop them. If any encrypted objects exist on the DB instance, you can't turn off TDE for the DB instance. If a user TDE certificate for encryption was restored (imported), then it should be dropped. When you use the console to remove the TDE option from an option group, the console indicates that it's processing. In addition, an error event is created if the option group is associated with an encrypted DB instance or DB snapshot.

The following example removes the TDE encryption from a database called `customerDatabase`.

```
------------ Removing TDE ---------------

USE [customerDatabase]
GO

-- Turn off encryption of the database
ALTER DATABASE [customerDatabase]
SET ENCRYPTION OFF
GO
```

```
-- Wait until the encryption state of the database becomes 1. The state is 5
 (Decryption in progress) for a while
SELECT db_name(database_id) as DatabaseName, * FROM sys.dm_database_encryption_keys
GO

-- Drop the DEK used for encryption
DROP DATABASE ENCRYPTION KEY
GO

-- Drop a user TDE certificate if it was restored (imported)
EXECUTE msdb.dbo.rds_drop_tde_certificate
 @certificate_name='UserTDECertificate_certificate_name';

-- Alter to SIMPLE Recovery mode so that your encrypted log gets truncated
USE [master]
GO
ALTER DATABASE [customerDatabase] SET RECOVERY SIMPLE
GO
```

When all objects are decrypted, you have two options:

1. You can modify the DB instance to be associated with an option group without the TDE option.
2. You can remove the TDE option from the option group.

SQL Server Audit

In Amazon RDS, you can audit Microsoft SQL Server databases by using the built-in SQL Server auditing mechanism. You can create audits and audit specifications in the same way that you create them for on-premises database servers.

RDS uploads the completed audit logs to your S3 bucket, using the IAM role that you provide. If you enable retention, RDS keeps your audit logs on your DB instance for the configured period of time.

For more information, see SQL Server Audit (database engine) in the Microsoft SQL Server documentation.

SQL Server Audit with Database Activity Streams

You can use Database Activity Streams for RDS to integrate SQL Server Audit events with database activity monitoring tools from Imperva, McAfee, and IBM. For more information about auditing with Database Activity Streams for RDS SQL Server, see Auditing in Microsoft SQL Server

Topics

- Support for SQL Server Audit
- Adding SQL Server Audit to the DB instance options
- Using SQL Server Audit
- Viewing audit logs
- Using SQL Server Audit with Multi-AZ instances
- Configuring an S3 bucket
- Manually creating an IAM role for SQL Server Audit

Support for SQL Server Audit

In Amazon RDS, starting with SQL Server 2016, all editions of SQL Server support server-level audits, and the Enterprise edition also supports database-level audits. Starting with SQL Server 2016 (13.x) SP1, all editions support both server-level and database-level audits. For more information, see SQL Server Audit (database engine) in the SQL Server documentation.

RDS supports configuring the following option settings for SQL Server Audit.

Option setting	Valid values	Description
IAM_ROLE_ARN	A valid Amazon Resource Name (ARN) in the format arn:aws:iam:: *account-id* :role/*role-name* .	The ARN of the IAM role that grants access to the S3 bucket where you want to store your audit logs. For more informati on, see Amazon Resource Names (ARNs) in the *AWS General Reference.*
S3_BUCKET_ARN	A valid ARN in the format arn:aws:s3::: *amzn-s3-demo-bucket* or arn:aws:s3::: *amzn-s3-demo-bucket* /key-pref ix	The ARN for the S3 bucket where you want to store your audit logs.
ENABLE_COMPRESSION	true or false	Controls audit log compressi on. By default, compression is enabled (set to true).
RETENTION_TIME	0 to 840	The retention time (in hours) that SQL Server audit records are kept on your RDS instance. By default, retention is disabled.

Adding SQL Server Audit to the DB instance options

Enabling SQL Server Audit requires two steps: enabling the option on the DB instance, and enabling the feature inside SQL Server. The process for adding the SQL Server Audit option to a DB instance is as follows:

1. Create a new option group, or copy or modify an existing option group.
2. Add and configure all required options.
3. Associate the option group with the DB instance.

After you add the SQL Server Audit option, you don't need to restart your DB instance. As soon as the option group is active, you can create audits and store audit logs in your S3 bucket.

To add and configure SQL Server Audit on a DB instance's option group

1. Choose one of the following:
   - Use an existing option group.
   - Create a custom DB option group and use that option group. For more information, see Creating an option group.
2. Add the **SQLSERVER_AUDIT** option to the option group, and configure the option settings. For more information about adding options, see Adding an option to an option group.
   - For **IAM role**, if you already have an IAM role with the required policies, you can choose that role. To create a new IAM role, choose **Create a New Role**. For information about the required policies, see Manually creating an IAM role for SQL Server Audit.
   - For **Select S3 destination**, if you already have an S3 bucket that you want to use, choose it. To create an S3 bucket, choose **Create a New S3 Bucket**.
   - For **Enable Compression**, leave this option chosen to compress audit files. Compression is enabled by default. To disable compression, clear **Enable Compression**.
   - For **Audit log retention**, to keep audit records on the DB instance, choose this option. Specify a retention time in hours. The maximum retention time is 35 days.
3. Apply the option group to a new or existing DB instance. Choose one of the following:
   - If you are creating a new DB instance, apply the option group when you launch the instance.
   - On an existing DB instance, apply the option group by modifying the instance and then attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

Modifying the SQL Server Audit option

After you enable the SQL Server Audit option, you can modify the settings. For information about how to modify option settings, see Modifying an option setting.

Removing SQL Server Audit from the DB instance options

You can turn off the SQL Server Audit feature by disabling audits and then deleting the option.

To remove auditing

1. Disable all of the audit settings inside SQL Server. To learn where audits are running, query the SQL Server security catalog views. For more information, see Security catalog views in the Microsoft SQL Server documentation.
2. Delete the SQL Server Audit option from the DB instance. Choose one of the following:
   - Delete the SQL Server Audit option from the option group that the DB instance uses. This change affects all DB instances that use the same option group. For more information, see Removing an option from an option group.
   - Modify the DB instance, and then choose an option group without the SQL Server Audit option. This change affects only the DB instance that you modify. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.
3. After you delete the SQL Server Audit option from the DB instance, you don't need to restart the instance. Remove unneeded audit files from your S3 bucket.

Using SQL Server Audit

You can control server audits, server audit specifications, and database audit specifications the same way that you control them for on-premises database servers.

Creating audits

You create server audits in the same way that you create them for on-premises database servers. For information about how to create server audits, see CREATE SERVER AUDIT in the Microsoft SQL Server documentation.

To avoid errors, adhere to the following limitations:

- Don't exceed the maximum number of supported server audits per instance of 50.
- Instruct SQL Server to write data to a binary file.
- Don't use `RDS_` as a prefix in the server audit name.
- For `FILEPATH`, specify `D:\rdsdbdata\SQLAudit`.
- For `MAXSIZE`, specify a size between 2 MB and 50 MB.
- Don't configure `MAX_ROLLOVER_FILES` or `MAX_FILES`.

- Don't configure SQL Server to shut down the DB instance if it fails to write the audit record.

Creating audit specifications

You create server audit specifications and database audit specifications the same way that you create them for on-premises database servers. For information about creating audit specifications, see CREATE SERVER AUDIT SPECIFICATION and CREATE DATABASE AUDIT SPECIFICATION in the Microsoft SQL Server documentation.

To avoid errors, don't use RDS_ as a prefix in the name of the database audit specification or server audit specification.

Viewing audit logs

Your audit logs are stored in `D:\rdsdbdata\SQLAudit`.

After SQL Server finishes writing to an audit log file—when the file reaches its size limit—Amazon RDS uploads the file to your S3 bucket. If retention is enabled, Amazon RDS moves the file into the retention folder: `D:\rdsdbdata\SQLAudit\transmitted`.

For information about configuring retention, see Adding SQL Server Audit to the DB instance options.

Audit records are kept on the DB instance until the audit log file is uploaded. You can view the audit records by running the following command.

```
SELECT   *
 FROM     msdb.dbo.rds_fn_get_audit_file
             ('D:\rdsdbdata\SQLAudit\*.sqlaudit'
             , default
             , default )
```

You can use the same command to view audit records in your retention folder by changing the filter to `D:\rdsdbdata\SQLAudit\transmitted\*.sqlaudit`.

```
SELECT   *
 FROM     msdb.dbo.rds_fn_get_audit_file
             ('D:\rdsdbdata\SQLAudit\transmitted\*.sqlaudit'
             , default
             , default )
```

Using SQL Server Audit with Multi-AZ instances

For Multi-AZ instances, the process for sending audit log files to Amazon S3 is similar to the process for Single-AZ instances. However, there are some important differences:

- Database audit specification objects are replicated to all nodes.
- Server audits and server audit specifications aren't replicated to secondary nodes. Instead, you have to create or modify them manually.

To capture server audits or a server audit specification from both nodes:

1. Create a server audit or a server audit specification on the primary node.
2. Fail over to the secondary node and create a server audit or a server audit specification with the same name and GUID on the secondary node. Use the `AUDIT_GUID` parameter to specify the GUID.

Configuring an S3 bucket

The audit log files are automatically uploaded from the DB instance to your S3 bucket. The following restrictions apply to the S3 bucket that you use as a target for audit files:

- It must be in the same AWS Region and AWS account as the DB instance.
- It must not be open to the public.
- The bucket owner must also be the IAM role owner.
- Your IAM role must have permissions for the customer-managed KMS key associated with the S3 bucket server-side encryption.

The target key that is used to store the data follows this naming schema: *`amzn-s3-demo-bucket`*`/key-prefix/instance-name/audit-name/node_file-name.ext`

> **Note**
>
> You set both the bucket name and the key prefix values with the (`S3_BUCKET_ARN`) option setting.

The schema is composed of the following elements:

- ***amzn-s3-demo-bucket*** – The name of your S3 bucket.
- **key-prefix** – The custom key prefix you want to use for audit logs.
- **instance-name** – The name of your Amazon RDS instance.
- **audit-name** – The name of the audit.
- **node** – The identifier of the node that is the source of the audit logs (node1 or node2). There is one node for a Single-AZ instance and two replication nodes for a Multi-AZ instance. These are not primary and secondary nodes, because the roles of primary and secondary change over time. Instead, the node identifier is a simple label.
  - **node1** – The first replication node (Single-AZ has one node only).
  - **node2** – The second replication node (Multi-AZ has two nodes).
- **file-name** – The target file name. The file name is taken as-is from SQL Server.
- **ext** – The extension of the file (`zip` or `sqlaudit`):
  - **zip** – If compression is enabled (default).
  - **sqlaudit** – If compression is disabled.

Manually creating an IAM role for SQL Server Audit

Typically, when you create a new option, the AWS Management Console creates the IAM role and the IAM trust policy for you. However, you can manually create a new IAM role to use with SQL Server Audits, so that you can customize it with any additional requirements you might have. To do this, you create an IAM role and delegate permissions so that the Amazon RDS service can use your Amazon S3 bucket. When you create this IAM role, you attach trust and permissions policies. The trust policy allows Amazon RDS to assume this role. The permission policy defines the actions that this role can do. For more information, see Creating a role to delegate permissions to an AWS service in the *AWS Identity and Access Management User Guide*.

You can use the examples in this section to create the trust relationships and permissions policies you need.

The following example shows a trust relationship for SQL Server Audit. It uses the *service principal* `rds.amazonaws.com` to allow RDS to write to the S3 bucket. A *service principal* is an identifier that is used to grant permissions to a service. Anytime you allow access to `rds.amazonaws.com` in this way, you are allowing RDS to perform an action on your behalf. For more information about service principals, see AWS JSON policy elements: Principal.

Example trust relationship for SQL Server Audit

JSON

```
{
     "Version":"2012-10-17",
     "Statement": [
         {
             "Effect": "Allow",
             "Principal": {
                 "Service": "rds.amazonaws.com"
             },
             "Action": "sts:AssumeRole"
         }
     ]
 }
```

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based trust relationships to limit the service's permissions to a specific resource. This is the most effective way to protect against the confused deputy problem.

You might use both global condition context keys and have the `aws:SourceArn` value contain the account ID. In this case, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same statement.

- Use `aws:SourceArn` if you want cross-service access for a single resource.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

In the trust relationship, make sure to use the `aws:SourceArn` global condition context key with the full Amazon Resource Name (ARN) of the resources accessing the role. For SQL Server Audit, make sure to include both the DB option group and the DB instances, as shown in the following example.

Example trust relationship with global condition context key for SQL Server Audit

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": [

 "arn:aws:rds:Region:my_account_ID:db:db_instance_identifier",
                        "arn:aws:rds:Region:my_account_ID:og:option_group_name"
                    ]
                }
            }
        }
    ]
}
```

In the following example of a permissions policy for SQL Server Audit, we specify an ARN for the Amazon S3 bucket. You can use ARNs to identify a specific account, user, or role that you want grant access to. For more information about using ARNs, see Amazon resource names (ARNs).

Example permissions policy for SQL Server Audit

JSON

```
{
     "Version":"2012-10-17",
     "Statement": [
         {
             "Effect": "Allow",
```

```
            "Action": "s3:ListAllMyBuckets",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketACL",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/key_prefix/*"
        }
    ]
}
```

Note

The `s3:ListAllMyBuckets` action is required for verifying that the same AWS account owns both the S3 bucket and the SQL Server DB instance. The action lists the names of the buckets in the account.

S3 bucket namespaces are global. If you accidentally delete your bucket, another user can create a bucket with the same name in a different account. Then the SQL Server Audit data is written to the new bucket.

Support for SQL Server Analysis Services in Amazon RDS for SQL Server

Microsoft SQL Server Analysis Services (SSAS) is part of the Microsoft Business Intelligence (MSBI) suite. SSAS is an online analytical processing (OLAP) and data mining tool that is installed within SQL Server. You use SSAS to analyze data to help make business decisions. SSAS differs from the SQL Server relational database because SSAS is optimized for queries and calculations common in a business intelligence environment.

You can turn on SSAS for existing or new DB instances. It's installed on the same DB instance as your database engine. For more information on SSAS, see the Microsoft Analysis services documentation.

Amazon RDS supports SSAS for SQL Server Standard and Enterprise Editions on the following versions:

- Tabular mode:
  - SQL Server 2019, version 15.00.4043.16.v1 and higher
  - SQL Server 2017, version 14.00.3223.3.v1 and higher
  - SQL Server 2016, version 13.00.5426.0.v1 and higher
- Multidimensional mode:
  - SQL Server 2019, version 15.00.4153.1.v1 and higher
  - SQL Server 2017, version 14.00.3381.3.v1 and higher
  - SQL Server 2016, version 13.00.5882.1.v1 and higher

Contents

- Limitations
- Turning on SSAS
  - Creating an option group for SSAS
  - Adding the SSAS option to the option group
  - Associating the option group with your DB instance
  - Allowing inbound access to your VPC security group
  - Enabling Amazon S3 integration
- Deploying SSAS projects on Amazon RDS

- Monitoring the status of a deployment task
- Using SSAS on Amazon RDS
  - Setting up a Windows-authenticated user for SSAS
  - Adding a domain user as a database administrator
  - Creating an SSAS proxy
  - Scheduling SSAS database processing using SQL Server Agent
  - Revoking SSAS access from the proxy
- Backing up an SSAS database
- Restoring an SSAS database
  - Restoring a DB instance to a specified time
- Changing the SSAS mode
- Turning off SSAS
- Troubleshooting SSAS issues

Limitations

The following limitations apply to using SSAS on RDS for SQL Server:

- RDS for SQL Server supports running SSAS in Tabular or Multidimensional mode. For more information, see Comparing tabular and multidimensional solutions in the Microsoft documentation.
- You can only use one SSAS mode at a time. Before changing modes, make sure to delete all of the SSAS databases.

  For more information, see Changing the SSAS mode.
- Multi-AZ instances aren't supported.
- Instances must use self-managed Active Directory or AWS Directory Service for Microsoft Active Directory for SSAS authentication. For more information, see Working with Active Directory with RDS for SQL Server.
- Users aren't given SSAS server administrator access, but they can be granted database-level administrator access.
- The only supported port for accessing SSAS is 2383.
- You can't deploy projects directly. We provide an RDS stored procedure to do this. For more information, see Deploying SSAS projects on Amazon RDS.

- Processing during deployment isn't supported.
- Using .xmla files for deployment isn't supported.
- SSAS project input files and database backup output files can only be in the `D:\S3` folder on the DB instance.

Turning on SSAS

Use the following process to turn on SSAS for your DB instance:

1. Create a new option group, or choose an existing option group.
2. Add the SSAS option to the option group.
3. Associate the option group with the DB instance.
4. Allow inbound access to the virtual private cloud (VPC) security group for the SSAS listener port.
5. Turn on Amazon S3 integration.

Creating an option group for SSAS

Use the AWS Management Console or the AWS CLI to create an option group that corresponds to the SQL Server engine and version of the DB instance that you plan to use.

> **Note**
>
> You can also use an existing option group if it's for the correct SQL Server engine and version.

Console

The following console procedure creates an option group for SQL Server Standard Edition 2017.

To create the option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** pane, do the following:

a. For **Name**, enter a name for the option group that is unique within your AWS account, such as **`ssas-se-2017`**. The name can contain only letters, digits, and hyphens.

b. For **Description**, enter a brief description of the option group, such as **`SSAS option group for SQL Server SE 2017`**. The description is used for display purposes.

c. For **Engine**, choose **sqlserver-se**.

d. For **Major engine version**, choose **14.00**.

5. Choose **Create**.

CLI

The following CLI example creates an option group for SQL Server Standard Edition 2017.

To create the option group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-option-group \
    --option-group-name ssas-se-2017 \
    --engine-name sqlserver-se \
    --major-engine-version 14.00 \
    --option-group-description "SSAS option group for SQL Server SE 2017"
```

  For Windows:

```
aws rds create-option-group ^
    --option-group-name ssas-se-2017 ^
    --engine-name sqlserver-se ^
    --major-engine-version 14.00 ^
    --option-group-description "SSAS option group for SQL Server SE 2017"
```

Adding the SSAS option to the option group

Next, use the AWS Management Console or the AWS CLI to add the SSAS option to the option group.

Console

To add the SSAS option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group that you just created.
4. Choose **Add option**.
5. Under **Option details**, choose **SSAS** for **Option name**.
6. Under **Option settings**, do the following:

   a. For **Max memory**, enter a value in the range 10–80.

      Max memory specifies the upper threshold above which SSAS begins releasing memory more aggressively to make room for requests that are running, and also new high-priority requests. The number is a percentage of the total memory of the DB instance. The allowed values are 10–80, and the default is 45.

   b. For **Mode**, choose the SSAS server mode, **Tabular** or **Multidimensional**.

      If you don't see the **Mode** option setting, it means that Multidimensional mode isn't supported in your AWS Region. For more information, see Limitations.

      Tabular is the default.

   c. For **Security groups**, choose the VPC security group to associate with the option.

   > **Note**
   >
   > The port for accessing SSAS, 2383, is prepopulated.

7. Under **Scheduling**, choose whether to add the option immediately or at the next maintenance window.
8. Choose **Add option**.

CLI

To add the SSAS option

1. Create a JSON file, for example `ssas-option.json`, with the following parameters:

   - `OptionGroupName` – The name of option group that you created or chose previously (`ssas-se-2017` in the following example).
   - `Port` – The port that you use to access SSAS. The only supported port is 2383.
   - `VpcSecurityGroupMemberships` – Memberships for VPC security groups for your RDS DB instance.
   - `MAX_MEMORY` – The upper threshold above which SSAS should begin releasing memory more aggressively to make room for requests that are running, and also new high-priority requests. The number is a percentage of the total memory of the DB instance. The allowed values are 10–80, and the default is 45.
   - `MODE` – The SSAS server mode, either `Tabular` or `Multidimensional`. `Tabular` is the default.

     If you receive an error that the `MODE` option setting isn't valid, it means that Multidimensional mode isn't supported in your AWS Region. For more information, see Limitations.

   The following is an example of a JSON file with SSAS option settings.

   ```
   {
   "OptionGroupName": "ssas-se-2017",
   "OptionsToInclude": [
    {
    "OptionName": "SSAS",
    "Port": 2383,
    "VpcSecurityGroupMemberships": ["sg-0abcdef123"],
    "OptionSettings": [{"Name":"MAX_MEMORY","Value":"60"},
   {"Name":"MODE","Value":"Multidimensional"}]
    }],
   "ApplyImmediately": true
   }
   ```

2. Add the SSAS option to the option group.

Example

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --cli-input-json file://ssas-option.json \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --cli-input-json file://ssas-option.json ^
    --apply-immediately
```

Associating the option group with your DB instance

You can use the console or the CLI to associate the option group with your DB instance.

Console

Associate your option group with a new or existing DB instance:

- For a new DB instance, associate the option group with the DB instance when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, modify the instance and associate the new option group with it. For more information, see Modifying an Amazon RDS DB instance.

> **Note**
>
> If you use an existing instance, it must already have an Active Directory domain and AWS Identity and Access Management (IAM) role associated with it. If you create a new instance, specify an existing Active Directory domain and IAM role. For more information, see Working with Active Directory with RDS for SQL Server.

CLI

You can associate your option group with a new or existing DB instance.

> **Note**
>
> If you use an existing instance, it must already have an Active Directory domain and IAM role associated with it. If you create a new instance, specify an existing Active Directory domain and IAM role. For more information, see Working with Active Directory with RDS for SQL Server.

To create a DB instance that uses the option group

- Specify the same DB engine type and major version that you used when creating the option group.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier myssasinstance \
    --db-instance-class db.m5.2xlarge \
    --engine sqlserver-se \
    --engine-version 14.00.3223.3.v1 \
    --allocated-storage 100 \
    --manage-master-user-password \
    --master-username admin \
    --storage-type gp2 \
    --license-model li \
    --domain-iam-role-name my-directory-iam-role \
    --domain my-domain-id \
    --option-group-name ssas-se-2017
```

  For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier myssasinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-se ^
    --engine-version 14.00.3223.3.v1 ^
    --allocated-storage 100 ^
    --manage-master-user-password ^
    --master-username admin ^
```

```
    --storage-type gp2 ^
    --license-model li ^
    --domain-iam-role-name my-directory-iam-role ^
    --domain my-domain-id ^
    --option-group-name ssas-se-2017
```

To modify a DB instance to associate the option group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier myssasinstance \
    --option-group-name ssas-se-2017 \
    --apply-immediately
```

  For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier myssasinstance ^
    --option-group-name ssas-se-2017 ^
    --apply-immediately
```

Allowing inbound access to your VPC security group

Create an inbound rule for the specified SSAS listener port in the VPC security group associated with your DB instance. For more information about setting up security groups, see Provide access to your DB instance in your VPC by creating a security group.

Enabling Amazon S3 integration

To download model configuration files to your host for deployment, use Amazon S3 integration. For more information, see Integrating an Amazon RDS for SQL Server DB instance with Amazon S3.

Deploying SSAS projects on Amazon RDS

On RDS, you can't deploy SSAS projects directly by using SQL Server Management Studio (SSMS). To deploy projects, use an RDS stored procedure.

> **Note**
>
> Using .xmla files for deployment isn't supported.

Before you deploy projects, make sure of the following:

- Amazon S3 integration is turned on. For more information, see Integrating an Amazon RDS for SQL Server DB instance with Amazon S3.
- The `Processing Option` configuration setting is set to `Do Not Process`. This setting means that no processing happens after deployment.
- You have both the *`myssasproject`*`.asdatabase` and *`myssasproject`*`.deploymentoptions` files. They're automatically generated when you build the SSAS project.

To deploy an SSAS project on RDS

1. Download the `.asdatabase` (SSAS model) file from your S3 bucket to your DB instance, as shown in the following example. For more information on the download parameters, see Downloading files from an Amazon S3 bucket to a SQL Server DB instance.

```
exec msdb.dbo.rds_download_from_s3
@s3_arn_of_file='arn:aws:s3:::bucket_name/myssasproject.asdatabase',
[@rds_file_path='D:\S3\myssasproject.asdatabase'],
[@overwrite_file=1];
```

2. Download the `.deploymentoptions` file from your S3 bucket to your DB instance.

```
exec msdb.dbo.rds_download_from_s3
@s3_arn_of_file='arn:aws:s3:::bucket_name/myssasproject.deploymentoptions',
[@rds_file_path='D:\S3\myssasproject.deploymentoptions'],
[@overwrite_file=1];
```

3. Deploy the project.

```
exec msdb.dbo.rds_msbi_task
@task_type='SSAS_DEPLOY_PROJECT',
@file_path='D:\S3\myssasproject.asdatabase';
```

Monitoring the status of a deployment task

To track the status of your deployment (or download) task, call the `rds_fn_task_status` function. It takes two parameters. The first parameter should always be `NULL` because it doesn't apply to SSAS. The second parameter accepts a task ID.

To see a list of all tasks, set the first parameter to `NULL` and the second parameter to `0`, as shown in the following example.

```
SELECT * FROM msdb.dbo.rds_fn_task_status(NULL,0);
```

To get a specific task, set the first parameter to `NULL` and the second parameter to the task ID, as shown in the following example.

```
SELECT * FROM msdb.dbo.rds_fn_task_status(NULL,42);
```

The `rds_fn_task_status` function returns the following information.

Output parameter	Description
`task_id`	The ID of the task.
`task_type`	For SSAS, tasks can have the following task types: • `SSAS_DEPLOY_PROJECT` • `SSAS_ADD_DB_ADMIN_MEMBER` • `SSAS_BACKUP_DB` • `SSAS_RESTORE_DB`
`database_name`	Not applicable to SSAS tasks.
`% complete`	The progress of the task as a percentage.

Output parameter	Description
duration (mins)	The amount of time spent on the task, in minutes.
lifecycle	The status of the task. Possible statuses are the following: • CREATED – After you call one of the SSAS stored procedures, a task is created and the status is set to CREATED. • IN_PROGRESS – After a task starts, the status is set to IN_PROGRESS. It can take up to five minutes for the status to change from CREATED to IN_PROGRESS. • SUCCESS – After a task completes, the status is set to SUCCESS. • ERROR – If a task fails, the status is set to ERROR. For more information about the error, see the task_info column. • CANCEL_REQUESTED – After you call rds_cancel_task, the status of the task is set to CANCEL_REQUESTED. • CANCELLED – After a task is successfully canceled, the status of the task is set to CANCELLED.

Output parameter	Description
task_info	Additional information about the task. If an error occurs during processing, this column contains information about the error. For more information, see Troubleshooting SSAS issues.
last_updated	The date and time that the task status was last updated.
created_at	The date and time that the task was created.
S3_object_arn	Not applicable to SSAS tasks.
overwrite_S3_backup_file	Not applicable to SSAS tasks.
KMS_master_key_arn	Not applicable to SSAS tasks.
filepath	Not applicable to SSAS tasks.
overwrite_file	Not applicable to SSAS tasks.
task_metadata	Metadata associated with the SSAS task.

Using SSAS on Amazon RDS

After deploying the SSAS project, you can directly process the OLAP database on SSMS.

To use SSAS on RDS

1. In SSMS, connect to SSAS using the user name and password for the Active Directory domain.
2. Expand **Databases**. The newly deployed SSAS database appears.
3. Locate the connection string, and update the user name and password to give access to the source SQL database. Doing this is required for processing SSAS objects.

a. For Tabular mode, do the following:

   1. Expand the **Connections** tab.
   2. Open the context (right-click) menu for the connection object, and then choose **Properties**.
   3. Update the user name and password in the connection string.

b. For Multidimensional mode, do the following:

   1. Expand the **Data Sources** tab.
   2. Open the context (right-click) menu for the data source object, and then choose **Properties**.
   3. Update the user name and password in the connection string.

4. Open the context (right-click) menu for the SSAS database that you created and choose **Process Database**.

   Depending on the size of the input data, the processing operation might take several minutes to complete.

Topics

- Setting up a Windows-authenticated user for SSAS
- Adding a domain user as a database administrator
- Creating an SSAS proxy
- Scheduling SSAS database processing using SQL Server Agent
- Revoking SSAS access from the proxy

Setting up a Windows-authenticated user for SSAS

The main administrator user (sometimes called the master user) can use the following code example to set up a Windows-authenticated login and grant the required procedure permissions. Doing this grants permissions to the domain user to run SSAS customer tasks, use S3 file transfer procedures, create credentials, and work with the SQL Server Agent proxy. For more information, see Credentials (database engine) and Create a SQL Server Agent proxy in the Microsoft documentation.

You can grant some or all of the following permissions as needed to Windows-authenticated users.

Example

```
-- Create a server-level domain user login, if it doesn't already exist
USE [master]
GO
CREATE LOGIN [mydomain\user_name] FROM WINDOWS
GO

-- Create domain user, if it doesn't already exist
USE [msdb]
GO
CREATE USER [mydomain\user_name] FOR LOGIN [mydomain\user_name]
GO

-- Grant necessary privileges to the domain user
USE [master]
GO
GRANT ALTER ANY CREDENTIAL TO [mydomain\user_name]
GO

USE [msdb]
GO
GRANT EXEC ON msdb.dbo.rds_msbi_task TO [mydomain\user_name] with grant option
GRANT SELECT ON msdb.dbo.rds_fn_task_status TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_task_status TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_cancel_task TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_download_from_s3 TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_upload_to_s3 TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_delete_from_filesystem TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.rds_gather_file_details TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.sp_add_proxy TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.sp_update_proxy TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.sp_grant_login_to_proxy TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.sp_revoke_login_from_proxy TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.sp_delete_proxy TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.sp_enum_login_for_proxy to [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.sp_enum_proxy_for_subsystem TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.rds_sqlagent_proxy TO [mydomain\user_name] with grant option
```

```
ALTER ROLE [SQLAgentUserRole] ADD MEMBER [mydomain\user_name]
GO
```

Adding a domain user as a database administrator

You can add a domain user as an SSAS database administrator in the following ways:

- A database administrator can use SSMS to create a role with `admin` privileges, then add users to that role.
- You can use the following stored procedure.

```
exec msdb.dbo.rds_msbi_task
@task_type='SSAS_ADD_DB_ADMIN_MEMBER',
@database_name='myssasdb',
@ssas_role_name='exampleRole',
@ssas_role_member='domain_name\domain_user_name';
```

  The following parameters are required:

  - `@task_type` – The type of the MSBI task, in this case `SSAS_ADD_DB_ADMIN_MEMBER`.
  - `@database_name` – The name of the SSAS database to which you're granting administrator privileges.
  - `@ssas_role_name` – The SSAS database administrator role name. If the role doesn't already exist, it's created.
  - `@ssas_role_member` – The SSAS database user that you're adding to the administrator role.

Creating an SSAS proxy

To be able to schedule SSAS database processing using SQL Server Agent, create an SSAS credential and an SSAS proxy. Run these procedures as a Windows-authenticated user.

To create the SSAS credential

- Create the credential for the proxy. To do this, you can use SSMS or the following SQL statement.

```
USE [master]
GO
CREATE CREDENTIAL [SSAS_Credential] WITH IDENTITY = N'mydomain\user_name', SECRET =
 N'mysecret'
```

```
GO
```

> **Note**
>
> IDENTITY must be a domain-authenticated login. Replace *mysecret* with the password for the domain-authenticated login.

To create the SSAS proxy

1. Use the following SQL statement to create the proxy.

```
USE [msdb]
GO
EXEC msdb.dbo.sp_add_proxy
 @proxy_name=N'SSAS_Proxy',@credential_name=N'SSAS_Credential',@description=N''
GO
```

2. Use the following SQL statement to grant access to the proxy to other users.

```
USE [msdb]
GO
EXEC msdb.dbo.sp_grant_login_to_proxy
 @proxy_name=N'SSAS_Proxy',@login_name=N'mydomain\user_name'
GO
```

3. Use the following SQL statement to give the SSAS subsystem access to the proxy.

```
USE [msdb]
GO
EXEC msdb.dbo.rds_sqlagent_proxy
 @task_type='GRANT_SUBSYSTEM_ACCESS',@proxy_name='SSAS_Proxy',@proxy_subsystem='SSAS'
GO
```

To view the proxy and grants on the proxy

1. Use the following SQL statement to view the grantees of the proxy.

```
USE [msdb]
GO
```

```
EXEC sp_help_proxy
GO
```

2. Use the following SQL statement to view the subsystem grants.

```
USE [msdb]
GO
EXEC msdb.dbo.sp_enum_proxy_for_subsystem
GO
```

Scheduling SSAS database processing using SQL Server Agent

After you create the credential and proxy and grant SSAS access to the proxy, you can create a SQL Server Agent job to schedule SSAS database processing.

To schedule SSAS database processing

- Use SSMS or T-SQL for creating the SQL Server Agent job. The following example uses T-SQL. You can further configure its job schedule through SSMS or T-SQL.
  - The `@command` parameter outlines the XML for Analysis (XMLA) command to be run by the SQL Server Agent job. This example configures SSAS Multidimensional database processing.
  - The `@server` parameter outlines the target SSAS server name of the SQL Server Agent job.

    To call the SSAS service within the same RDS DB instance where the SQL Server Agent job resides, use `localhost:2383`.

    To call the SSAS service from outside the RDS DB instance, use the RDS endpoint. You can also use the Kerberos Active Directory (AD) endpoint (*`your-DB-instance-name`*`.`*`your-AD-domain-name`*) if the RDS DB instances are joined by the same domain. For external DB instances, make sure to properly configure the VPC security group associated with the RDS DB instance for a secure connection.

  You can further edit the query to support various XMLA operations. Make edits either by directly modifying the T-SQL query or by using the SSMS UI following SQL Server Agent job creation.

```
USE [msdb]
GO
```

```
DECLARE @jobId BINARY(16)
EXEC msdb.dbo.sp_add_job @job_name=N'SSAS_Job',
    @enabled=1,
    @notify_level_eventlog=0,
    @notify_level_email=0,
    @notify_level_netsend=0,
    @notify_level_page=0,
    @delete_level=0,
    @category_name=N'[Uncategorized (Local)]',
    @job_id = @jobId OUTPUT
GO
EXEC msdb.dbo.sp_add_jobserver
    @job_name=N'SSAS_Job',
    @server_name = N'(local)'
GO
EXEC msdb.dbo.sp_add_jobstep @job_name=N'SSAS_Job',
 @step_name=N'Process_SSAS_Object',
    @step_id=1,
    @cmdexec_success_code=0,
    @on_success_action=1,
    @on_success_step_id=0,
    @on_fail_action=2,
    @on_fail_step_id=0,
    @retry_attempts=0,
    @retry_interval=0,
    @os_run_priority=0, @subsystem=N'ANALYSISCOMMAND',
    @command=N'<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/
engine">
        <Parallel>
            <Process xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:ddl2="http://schemas.microsoft.com/analysisservices/2003/
engine/2" xmlns:ddl2_2="http://schemas.microsoft.com/analysisservices/2003/
engine/2/2"
                xmlns:ddl100_100="http://schemas.microsoft.com/
analysisservices/2008/engine/100/100" xmlns:ddl200="http://schemas.microsoft.com/
analysisservices/2010/engine/200"
                xmlns:ddl200_200="http://schemas.microsoft.com/
analysisservices/2010/engine/200/200" xmlns:ddl300="http://schemas.microsoft.com/
analysisservices/2011/engine/300"
                xmlns:ddl300_300="http://schemas.microsoft.com/
analysisservices/2011/engine/300/300" xmlns:ddl400="http://schemas.microsoft.com/
analysisservices/2012/engine/400"
```

```
                xmlns:ddl400_400="http://schemas.microsoft.com/
analysisservices/2012/engine/400/400" xmlns:ddl500="http://schemas.microsoft.com/
analysisservices/2013/engine/500"
                xmlns:ddl500_500="http://schemas.microsoft.com/
analysisservices/2013/engine/500/500">
                <Object>
                    <DatabaseID>Your_SSAS_Database_ID</DatabaseID>
                </Object>
                <Type>ProcessFull</Type>
                <WriteBackTableCreation>UseExisting</WriteBackTableCreation>
            </Process>
        </Parallel>
    </Batch>',
    @server=N'localhost:2383',
    @database_name=N'master',
    @flags=0,
    @proxy_name=N'SSAS_Proxy'
GO
```

Revoking SSAS access from the proxy

You can revoke access to the SSAS subsystem and delete the SSAS proxy using the following stored procedures.

To revoke access and delete the proxy

1. Revoke subsystem access.

```
USE [msdb]
GO
EXEC msdb.dbo.rds_sqlagent_proxy
 @task_type='REVOKE_SUBSYSTEM_ACCESS',@proxy_name='SSAS_Proxy',@proxy_subsystem='SSAS'
GO
```

2. Revoke the grants on the proxy.

```
USE [msdb]
GO
EXEC msdb.dbo.sp_revoke_login_from_proxy
 @proxy_name=N'SSAS_Proxy',@name=N'mydomain\user_name'
GO
```

3. Delete the proxy.

```
USE [msdb]
GO
EXEC dbo.sp_delete_proxy @proxy_name = N'SSAS_Proxy'
GO
```

Backing up an SSAS database

You can create SSAS database backup files only in the `D:\S3` folder on the DB instance. To move the backup files to your S3 bucket, use Amazon S3.

You can back up an SSAS database as follows:

- A domain user with the `admin` role for a particular database can use SSMS to back up the database to the `D:\S3` folder.

  For more information, see Adding a domain user as a database administrator.
- You can use the following stored procedure. This stored procedure doesn't support encryption.

```
exec msdb.dbo.rds_msbi_task
@task_type='SSAS_BACKUP_DB',
@database_name='myssasdb',
@file_path='D:\S3\ssas_db_backup.abf',
[@ssas_apply_compression=1],
[@ssas_overwrite_file=1];
```

  The following parameters are required:

  - `@task_type` – The type of the MSBI task, in this case SSAS_BACKUP_DB.
  - `@database_name` – The name of the SSAS database that you're backing up.
  - `@file_path` – The path for the SSAS backup file. The `.abf` extension is required.

  The following parameters are optional:

  - `@ssas_apply_compression` – Whether to apply SSAS backup compression. Valid values are 1 (Yes) and 0 (No).
  - `@ssas_overwrite_file` – Whether to overwrite the SSAS backup file. Valid values are 1 (Yes) and 0 (No).

Restoring an SSAS database

Use the following stored procedure to restore an SSAS database from a backup.

You can't restore a database if there is an existing SSAS database with the same name. The stored procedure for restoring doesn't support encrypted backup files.

```
exec msdb.dbo.rds_msbi_task
@task_type='SSAS_RESTORE_DB',
@database_name='mynewssasdb',
@file_path='D:\S3\ssas_db_backup.abf';
```

The following parameters are required:

- `@task_type` – The type of the MSBI task, in this case `SSAS_RESTORE_DB`.
- `@database_name` – The name of the new SSAS database that you're restoring to.
- `@file_path` – The path to the SSAS backup file.

Restoring a DB instance to a specified time

Point-in-time recovery (PITR) doesn't apply to SSAS databases. If you do PITR, only the SSAS data in the last snapshot before the requested time is available on the restored instance.

To have up-to-date SSAS databases on a restored DB instance

1. Back up your SSAS databases to the `D:\S3` folder on the source instance.
2. Transfer the backup files to the S3 bucket.
3. Transfer the backup files from the S3 bucket to the `D:\S3` folder on the restored instance.
4. Run the stored procedure to restore the SSAS databases onto the restored instance.

   You can also reprocess the SSAS project to restore the databases.

Changing the SSAS mode

You can change the mode in which SSAS runs, either Tabular or Multidimensional. To change the mode, use the AWS Management Console or the AWS CLI to modify the options settings in the SSAS option.

> **Important**
>
> You can only use one SSAS mode at a time. Make sure to delete all of the SSAS databases before changing the mode, or you receive an error.

Console

The following Amazon RDS console procedure changes the SSAS mode to Tabular and sets the `MAX_MEMORY` parameter to 70 percent.

To modify the SSAS option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group with the SSAS option that you want to modify (`ssas-se-2017` in the previous examples).
4. Choose **Modify option**.
5. Change the option settings:
   a. For **Max memory**, enter **70**.
   b. For **Mode**, choose **Tabular**.
6. Choose **Modify option**.

AWS CLI

The following AWS CLI example changes the SSAS mode to Tabular and sets the `MAX_MEMORY` parameter to 70 percent.

For the CLI command to work, make sure to include all of the required parameters, even if you're not modifying them.

To modify the SSAS option

- Use one of the following commands.

Example

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name ssas-se-2017 \
    --options
 "OptionName=SSAS,VpcSecurityGroupMemberships=sg-12345e67,OptionSettings=[{Name=MAX_MEMORY,
{Name=MODE,Value=Tabular}]" \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name ssas-se-2017 ^
    --options
 OptionName=SSAS,VpcSecurityGroupMemberships=sg-12345e67,OptionSettings=[{Name=MAX_MEMORY,V
{Name=MODE,Value=Tabular}] ^
    --apply-immediately
```

Turning off SSAS

To turn off SSAS, remove the SSAS option from its option group.

> **Important**
>
> Before you remove the SSAS option, delete your SSAS databases.
> We highly recommend that you back up your SSAS databases before deleting them and removing the SSAS option.

Console

To remove the SSAS option from its option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.

3. Choose the option group with the SSAS option that you want to remove (`ssas-se-2017` in the previous examples).
4. Choose **Delete option**.
5. Under **Deletion options**, choose **SSAS** for **Options to delete**.
6. Under **Apply immediately**, choose **Yes** to delete the option immediately, or **No** to delete it at the next maintenance window.
7. Choose **Delete**.

AWS CLI

To remove the SSAS option from its option group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds remove-option-from-option-group \
    --option-group-name ssas-se-2017 \
    --options SSAS \
    --apply-immediately
```

  For Windows:

```
aws rds remove-option-from-option-group ^
    --option-group-name ssas-se-2017 ^
    --options SSAS ^
    --apply-immediately
```

Troubleshooting SSAS issues

You might encounter the following issues when using SSAS.

Issue	Type	Troubleshooting suggestions
Unable to configure the SSAS option. The requested SSAS mode	RDS event	You can't change the SSAS mode if you still have SSAS databases that use the current

Issue	Type	Troubleshooting suggestions
is *new_mode*, but the current DB instance has *number current_mode* databases. Delete the existing databases before switching to *new_mode* mode. To regain access to *current_mode* mode for database deletion, either update the current DB option group, or attach a new option group with %s as the MODE option setting value for the SSAS option.		mode. Delete the SSAS databases, then try again.
Unable to remove the SSAS option because there are *number* existing *mode* databases. The SSAS option can't be removed until all SSAS databases are deleted. Add the SSAS option again, delete all SSAS databases, and try again.	RDS event	You can't turn off SSAS if you still have SSAS databases. Delete the SSAS databases , then try again.
The SSAS option isn't enabled or is in the process of being enabled. Try again later.	RDS stored procedure	You can't run SSAS stored procedures when the option is turned off, or when it's being turned on.

Issue	Type	Troubleshooting suggestions
The SSAS option is configured incorrectly. Make sure that the option group membership status is "in-sync" , and review the RDS event logs for relevant SSAS configuration error messages. Following these investiga tions, try again. If errors continue to occur, contact AWS Support.	RDS stored procedure	You can't run SSAS stored procedures when your option group membership isn't in the `in-sync` status. This puts the SSAS option in an incorrect configuration state. If your option group membership status changes to `failed` due to SSAS option modification, there are two possible reasons: 1. The SSAS option was removed without the SSAS databases being deleted. 2. The SSAS mode was updated from Tabular to Multidimensional, or from Multidimenisonal to Tabular, without the existing SSAS databases being deleted. Reconfigure the SSAS option, because RDS allows only one SSAS mode at a time, and doesn't support SSAS option removal with SSAS databases present. Check the RDS event logs for configuration errors for your SSAS instance, and resolve the issues accordingly.
Deployment failed. The change can only be deployed on a server running in *`deployment_file_mode`* mode. The current server mode is *`current_mode`* .	RDS stored procedure	You can't deploy a Tabular database to a Multidimensional server, or a Multidime nsional database to a Tabular server. Make sure that you're using files with the correct mode, and verify that the `MODE` option setting is set to the appropriate value.

Issue	Type	Troubleshooting suggestions
The restore failed. The backup file can only be restored on a server running in *restore_file_mode* mode. The current server mode is *current_mode* .	RDS stored procedure	You can't restore a Tabular database to a Multidimensional server, or a Multidimensional database to a Tabular server. Make sure that you're using files with the correct mode, and verify that the MODE option setting is set to the appropriate value.
The restore failed. The backup file and the RDS DB instance versions are incompatible.	RDS stored procedure	You can't restore an SSAS database with a version incompatible to the SQL Server instance version. For more information, see Compatibility levels for tabular models and Compatibility level of a multidimensional database in the Microsoft documentation.
The restore failed. The backup file specified in the restore operation is damaged or is not an SSAS backup file. Make sure that @rds_file_path is correctly formatted.	RDS stored procedure	You can't restore an SSAS database with a damaged file. Make sure that the file isn't damaged or corrupted. This error can also be raised when `@rds_file_path` isn't correctly formatted (for example, it has double backslashes as in `D:\S3\\incorrect_format.abf`).

Issue	Type	Troubleshooting suggestions
The restore failed. The restored database name can't contain any reserved words or invalid character s: . , ; ' ` : / \\ * | ? \" & % $! + = ( ) [ ] { } < >, or be longer than 100 characters.	RDS stored procedure	The restored database name can't contain any reserved words or characters that aren't valid, or be longer than 100 charac ters. For SSAS object naming conventions, see Object naming rules in the Microsoft documentation.
An invalid role name was provided. The role name can't contain any reserved strings.	RDS stored procedure	The role name can't contain any reserved strings. For SSAS object naming conventions, see Object naming rules in the Microsoft documentation.
An invalid role name was provided. The role name can't contain any of the following reserved characters: . , ; ' ` : / \\ * | ? \" & % $! + = ( ) [ ] { } < >	RDS stored procedure	The role name can't contain any reserved characters. For SSAS object naming conventions, see Object naming rules in the Microsoft documentation.

Support for SQL Server Integration Services in Amazon RDS for SQL Server

Microsoft SQL Server Integration Services (SSIS) is a component that you can use to perform a broad range of data migration tasks. SSIS is a platform for data integration and workflow applications. It features a data warehousing tool used for data extraction, transformation, and loading (ETL). You can also use this tool to automate maintenance of SQL Server databases and updates to multidimensional cube data.

SSIS projects are organized into packages saved as XML-based .dtsx files. Packages can contain control flows and data flows. You use data flows to represent ETL operations. After deployment, packages are stored in SQL Server in the SSISDB database. SSISDB is an online transaction processing (OLTP) database in the full recovery mode.

Amazon RDS for SQL Server supports running SSIS directly on an RDS DB instance. You can enable SSIS on an existing or new DB instance. SSIS is installed on the same DB instance as your database engine.

RDS supports SSIS for SQL Server Standard and Enterprise Editions on the following versions:

- SQL Server 2022, all versions
- SQL Server 2019, version 15.00.4043.16.v1 and higher
- SQL Server 2017, version 14.00.3223.3.v1 and higher
- SQL Server 2016, version 13.00.5426.0.v1 and higher

Contents

- Limitations and recommendations
- Enabling SSIS
  - Creating the option group for SSIS
  - Adding the SSIS option to the option group
  - Creating the parameter group for SSIS
  - Modifying the parameter for SSIS
  - Associating the option group and parameter group with your DB instance
  - Enabling S3 integration
- Administrative permissions on SSISDB

- Setting up a Windows-authenticated user for SSIS
- Deploying an SSIS project
- Monitoring the status of a deployment task
- Using SSIS
  - Setting database connection managers for SSIS projects
  - Creating an SSIS proxy
  - Scheduling an SSIS package using SQL Server Agent
  - Revoking SSIS access from the proxy
- Disable and drop SSIS database
  - Disabling SSIS
  - Dropping the SSISDB database

Limitations and recommendations

The following limitations and recommendations apply to running SSIS on RDS for SQL Server:

- The DB instance must have an associated parameter group with the `clr enabled` parameter set to 1. For more information, see Modifying the parameter for SSIS.

  > **Note**
  >
  > If you enable the `clr enabled` parameter on SQL Server 2017 or 2019, you can't use the common language runtime (CLR) on your DB instance. For more information, see Features not supported and features with limited support.

- The following control flow tasks are supported:
  - Analysis Services Execute DDL Task
  - Analysis Services Processing Task
  - Bulk Insert Task
  - Check Database Integrity Task
  - Data Flow Task
  - Data Mining Query Task
  - Data Profiling Task

  - Execute Package Task
  - Execute SQL Server Agent Job Task
  - Execute SQL Task
  - Execute T-SQL Statement Task
  - Notify Operator Task
  - Rebuild Index Task
  - Reorganize Index Task
  - Shrink Database Task
  - Transfer Database Task
  - Transfer Jobs Task
  - Transfer Logins Task
  - Transfer SQL Server Objects Task
  - Update Statistics Task
- Only project deployment is supported.
- Running SSIS packages by using SQL Server Agent is supported.
- SSIS log records can be inserted only into user-created databases.
- Use only the `D:\S3` folder for working with files. Files placed in any other directory are deleted. Be aware of a few other file location details:
  - Place SSIS project input and output files in the `D:\S3` folder.
  - For the Data Flow Task, change the location for `BLOBTempStoragePath` and `BufferTempStoragePath` to a file inside the `D:\S3` folder. The file path must start with `D:\S3\`.
  - Ensure that all parameters, variables, and expressions used for file connections point to the `D:\S3` folder.
  - On Multi-AZ instances, files created by SSIS in the `D:\S3` folder are deleted after a failover. For more information, see Multi-AZ limitations for S3 integration.
  - Upload the files created by SSIS in the `D:\S3` folder to your Amazon S3 bucket to make them durable.
- Import Column and Export Column transformations and the Script component on the Data Flow Task aren't supported.
- You can't enable dump on running SSIS packages, and you can't add data taps on SSIS packages.

- The SSIS Scale Out feature isn't supported.
- You can't deploy projects directly. We provide RDS stored procedures to do this. For more information, see Deploying an SSIS project.
- Build SSIS project (.ispac) files with the `DoNotSavePasswords` protection mode for deploying on RDS.
- SSIS isn't supported on Always On instances with read replicas.
- You can't back up the SSISDB database that is associated with the `SSIS` option.
- Importing and restoring the SSISDB database from other instances of SSIS isn't supported.
- You can connect to other SQL Server DB instances or to an Oracle data source. Connecting to other database engines, such as MySQL or PostgreSQL, isn't supported for SSIS on RDS for SQL Server. For more information on connecting to an Oracle data source, see Linked Servers with Oracle OLEDB.
- SSIS does not support a domain joined instance with an outgoing trust to an on-premises domain. When using an outgoing trust, run the SSIS job from an account in the local AWS domain.
- Executing file system based packages is not supported.

Enabling SSIS

You enable SSIS by adding the SSIS option to your DB instance. Use the following process:

1. Create a new option group, or choose an existing option group.
2. Add the `SSIS` option to the option group.
3. Create a new parameter group, or choose an existing parameter group.
4. Modify the parameter group to set the `clr enabled` parameter to 1.
5. Associate the option group and parameter group with the DB instance.
6. Enable Amazon S3 integration.

> **Note**
>
> If a database with the name SSISDB or a reserved SSIS login already exists on the DB instance, you can't enable SSIS on the instance.

Creating the option group for SSIS

To work with SSIS, create an option group or modify an option group that corresponds to the SQL Server edition and version of the DB instance that you plan to use. To do this, use the AWS Management Console or the AWS CLI.

Console

The following procedure creates an option group for SQL Server Standard Edition 2016.

To create the option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** window, do the following:
   a. For **Name**, enter a name for the option group that is unique within your AWS account, such as **`ssis-se-2016`**. The name can contain only letters, digits, and hyphens.
   b. For **Description**, enter a brief description of the option group, such as **`SSIS option group for SQL Server SE 2016`**. The description is used for display purposes.
   c. For **Engine**, choose **sqlserver-se**.
   d. For **Major engine version**, choose **13.00**.
5. Choose **Create**.

CLI

The following procedure creates an option group for SQL Server Standard Edition 2016.

To create the option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-option-group \
```

```
    --option-group-name ssis-se-2016 \
    --engine-name sqlserver-se \
    --major-engine-version 13.00 \
    --option-group-description "SSIS option group for SQL Server SE 2016"
```

For Windows:

```
aws rds create-option-group ^
    --option-group-name ssis-se-2016 ^
    --engine-name sqlserver-se ^
    --major-engine-version 13.00 ^
    --option-group-description "SSIS option group for SQL Server SE 2016"
```

Adding the SSIS option to the option group

Next, use the AWS Management Console or the AWS CLI to add the SSIS option to your option group.

Console

To add the SSIS option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group that you just created, **ssis-se-2016** in this example.
4. Choose **Add option**.
5. Under **Option details**, choose **SSIS** for **Option name**.
6. Under **Scheduling**, choose whether to add the option immediately or at the next maintenance window.
7. Choose **Add option**.

CLI

To add the SSIS option

- Add the SSIS option to the option group.

Example

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name ssis-se-2016 \
    --options OptionName=SSIS \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name ssis-se-2016 ^
    --options OptionName=SSIS ^
    --apply-immediately
```

Creating the parameter group for SSIS

Create or modify a parameter group for the `clr enabled` parameter that corresponds to the SQL Server edition and version of the DB instance that you plan to use for SSIS.

Console

The following procedure creates a parameter group for SQL Server Standard Edition 2016.

To create the parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
4. In the **Create parameter group** pane, do the following:
   a. For **Parameter group family**, choose **sqlserver-se-13.0**.
   b. For **Group name**, enter an identifier for the parameter group, such as **`ssis-sqlserver-se-13`**.
   c. For **Description**, enter **`clr enabled parameter group`**.
5. Choose **Create**.

CLI

The following procedure creates a parameter group for SQL Server Standard Edition 2016.

To create the parameter group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name ssis-sqlserver-se-13 \
    --db-parameter-group-family "sqlserver-se-13.0" \
    --description "clr enabled parameter group"
```

  For Windows:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name ssis-sqlserver-se-13 ^
    --db-parameter-group-family "sqlserver-se-13.0" ^
    --description "clr enabled parameter group"
```

Modifying the parameter for SSIS

Modify the `clr enabled` parameter in the parameter group that corresponds to the SQL Server edition and version of your DB instance. For SSIS, set the `clr enabled` parameter to 1.

Console

The following procedure modifies the parameter group that you created for SQL Server Standard Edition 2016.

To modify the parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose the parameter group, such as **ssis-sqlserver-se-13**.

4. Under **Parameters**, filter the parameter list for **`clr`**.
5. Choose **clr enabled**.
6. Choose **Edit parameters**.
7. From **Values**, choose **1**.
8. Choose **Save changes**.

CLI

The following procedure modifies the parameter group that you created for SQL Server Standard Edition 2016.

To modify the parameter group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name ssis-sqlserver-se-13 \
    --parameters "ParameterName='clr
 enabled',ParameterValue=1,ApplyMethod=immediate"
```

  For Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name ssis-sqlserver-se-13 ^
    --parameters "ParameterName='clr
 enabled',ParameterValue=1,ApplyMethod=immediate"
```

Associating the option group and parameter group with your DB instance

To associate the SSIS option group and parameter group with your DB instance, use the AWS Management Console or the AWS CLI

> **Note**
>
> If you use an existing instance, it must already have an Active Directory domain and AWS Identity and Access Management (IAM) role associated with it. If you create a new instance, specify an existing Active Directory domain and IAM role. For more information, see Working with Active Directory with RDS for SQL Server.

Console

To finish enabling SSIS, associate your SSIS option group and parameter group with a new or existing DB instance:

- For a new DB instance, associate them when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, associate them by modifying the instance. For more information, see Modifying an Amazon RDS DB instance.

CLI

You can associate the SSIS option group and parameter group with a new or existing DB instance.

To create an instance with the SSIS option group and parameter group

- Specify the same DB engine type and major version as you used when creating the option group.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier myssisinstance \
    --db-instance-class db.m5.2xlarge \
    --engine sqlserver-se \
    --engine-version 13.00.5426.0.v1 \
    --allocated-storage 100 \
    --manage-master-user-password \
    --master-username admin \
    --storage-type gp2 \
```

```
    --license-model li \
    --domain-iam-role-name my-directory-iam-role \
    --domain my-domain-id \
    --option-group-name ssis-se-2016 \
    --db-parameter-group-name ssis-sqlserver-se-13
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier myssisinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-se ^
    --engine-version 13.00.5426.0.v1 ^
    --allocated-storage 100 ^
    --manage-master-user-password ^
    --master-username admin ^
    --storage-type gp2 ^
    --license-model li ^
    --domain-iam-role-name my-directory-iam-role ^
    --domain my-domain-id ^
    --option-group-name ssis-se-2016 ^
    --db-parameter-group-name ssis-sqlserver-se-13
```

To modify an instance and associate the SSIS option group and parameter group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier myssisinstance \
    --option-group-name ssis-se-2016 \
    --db-parameter-group-name ssis-sqlserver-se-13 \
    --apply-immediately
```

  For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier myssisinstance ^
```

```
    --option-group-name ssis-se-2016 ^
    --db-parameter-group-name ssis-sqlserver-se-13 ^
    --apply-immediately
```

Enabling S3 integration

To download SSIS project (.ispac) files to your host for deployment, use S3 file integration. For more information, see Integrating an Amazon RDS for SQL Server DB instance with Amazon S3.

Administrative permissions on SSISDB

When the instance is created or modified with the SSIS option, the result is an SSISDB database with the ssis_admin and ssis_logreader roles granted to the master user. The master user has the following privileges in SSISDB:

- alter on ssis_admin role
- alter on ssis_logreader role
- alter any user

Because the master user is a SQL-authenticated user, you can't use the master user for executing SSIS packages. The master user can use these privileges to create new SSISDB users and add them to the ssis_admin and ssis_logreader roles. Doing this is useful for giving access to your domain users for using SSIS.

Setting up a Windows-authenticated user for SSIS

The master user can use the following code example to set up a Windows-authenticated login in SSISDB and grant the required procedure permissions. Doing this grants permissions to the domain user to deploy and run SSIS packages, use S3 file transfer procedures, create credentials, and work with the SQL Server Agent proxy. For more information, see Credentials (database engine) and Create a SQL Server Agent proxy in the Microsoft documentation.

> **Note**
>
> You can grant some or all of the following permissions as needed to Windows-authenticated users.

Example

```
-- Create a server-level SQL login for the domain user, if it doesn't already exist
USE [master]
GO
CREATE LOGIN [mydomain\user_name] FROM WINDOWS
GO

-- Create a database-level account for the domain user, if it doesn't already exist

USE [SSISDB]
GO
CREATE USER [mydomain\user_name] FOR LOGIN [mydomain\user_name]

-- Add SSIS role membership to the domain user
ALTER ROLE [ssis_admin] ADD MEMBER [mydomain\user_name]
ALTER ROLE [ssis_logreader] ADD MEMBER [mydomain\user_name]
GO

-- Add MSDB role membership to the domain user
USE [msdb]
GO
CREATE USER [mydomain\user_name] FOR LOGIN [mydomain\user_name]

-- Grant MSDB stored procedure privileges to the domain user
GRANT EXEC ON msdb.dbo.rds_msbi_task TO [mydomain\user_name] with grant option
GRANT SELECT ON msdb.dbo.rds_fn_task_status TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_task_status TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_cancel_task TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_download_from_s3 TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_upload_to_s3 TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.rds_delete_from_filesystem TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.rds_gather_file_details TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.sp_add_proxy TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.sp_update_proxy TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.sp_grant_login_to_proxy TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.sp_revoke_login_from_proxy TO [mydomain\user_name] with grant
 option
GRANT EXEC ON msdb.dbo.sp_delete_proxy TO [mydomain\user_name] with grant option
GRANT EXEC ON msdb.dbo.sp_enum_login_for_proxy to [mydomain\user_name] with grant
 option
```

```
GRANT EXEC ON msdb.dbo.sp_enum_proxy_for_subsystem TO [mydomain\user_name]  with grant
 option
GRANT EXEC ON msdb.dbo.rds_sqlagent_proxy TO [mydomain\user_name] WITH GRANT OPTION


-- Add the SQLAgentUserRole privilege to the domain user
USE [msdb]
GO
ALTER ROLE [SQLAgentUserRole] ADD MEMBER [mydomain\user_name]
GO

-- Grant the ALTER ANY CREDENTIAL privilege to the domain user
USE [master]
GO
GRANT ALTER ANY CREDENTIAL TO [mydomain\user_name]
GO
```

Deploying an SSIS project

On RDS, you can't deploy SSIS projects directly by using SQL Server Management Studio (SSMS) or SSIS procedures. To download project files from Amazon S3 and then deploy them, use RDS stored procedures.

To run the stored procedures, log in as any user that you granted permissions for running the stored procedures. For more information, see Setting up a Windows-authenticated user for SSIS.

To deploy the SSIS project

1. Download the project (.ispac) file.

   ```
   exec msdb.dbo.rds_download_from_s3
   @s3_arn_of_file='arn:aws:s3:::bucket_name/ssisproject.ispac',
   @rds_file_path='D:\S3\ssisproject.ispac',
   @overwrite_file=1;
   ```

2. Submit the deployment task, making sure of the following:

   - The folder is present in the SSIS catalog.
   - The project name matches the project name that you used while developing the SSIS project.

```
exec msdb.dbo.rds_msbi_task
@task_type='SSIS_DEPLOY_PROJECT',
@folder_name='DEMO',
@project_name='ssisproject',
@file_path='D:\S3\ssisproject.ispac';
```

Monitoring the status of a deployment task

To track the status of your deployment task, call the `rds_fn_task_status` function. It takes two parameters. The first parameter should always be `NULL` because it doesn't apply to SSIS. The second parameter accepts a task ID.

To see a list of all tasks, set the first parameter to `NULL` and the second parameter to `0`, as shown in the following example.

```
SELECT * FROM msdb.dbo.rds_fn_task_status(NULL,0);
```

To get a specific task, set the first parameter to `NULL` and the second parameter to the task ID, as shown in the following example.

```
SELECT * FROM msdb.dbo.rds_fn_task_status(NULL,42);
```

The `rds_fn_task_status` function returns the following information.

Output parameter	Description
`task_id`	The ID of the task.
`task_type`	`SSIS_DEPLOY_PROJECT`
`database_name`	Not applicable to SSIS tasks.
`% complete`	The progress of the task as a percentage.
`duration (mins)`	The amount of time spent on the task, in minutes.

Output parameter	Description
lifecycle	The status of the task. Possible statuses are the following: • CREATED – After you call the msdb.dbo.rds_msbi_task stored procedure, a task is created and the status is set to CREATED. • IN_PROGRESS – After a task starts, the status is set to IN_PROGRESS. It can take up to five minutes for the status to change from CREATED to IN_PROGRESS. • SUCCESS – After a task completes, the status is set to SUCCESS. • ERROR – If a task fails, the status is set to ERROR. For more information about the error, see the task_info column. • CANCEL_REQUESTED – After you call rds_cancel_task, the status of the task is set to CANCEL_REQUESTED. • CANCELLED – After a task is successfully canceled, the status of the task is set to CANCELLED.
task_info	Additional information about the task. If an error occurs during processing, this column contains information about the error.
last_updated	The date and time that the task status was last updated.

Output parameter	Description
created_at	The date and time that the task was created.
S3_object_arn	Not applicable to SSIS tasks.
overwrite_S3_backup_file	Not applicable to SSIS tasks.
KMS_master_key_arn	Not applicable to SSIS tasks.
filepath	Not applicable to SSIS tasks.
overwrite_file	Not applicable to SSIS tasks.
task_metadata	Metadata associated with the SSIS task.

Using SSIS

After deploying the SSIS project into the SSIS catalog, you can run packages directly from SSMS or schedule them by using SQL Server Agent. You must use a Windows-authenticated login for executing SSIS packages. For more information, see Setting up a Windows-authenticated user for SSIS.

Topics

- Setting database connection managers for SSIS projects
- Creating an SSIS proxy
- Scheduling an SSIS package using SQL Server Agent
- Revoking SSIS access from the proxy

Setting database connection managers for SSIS projects

When you use a connection manager, you can use these types of authentication:

- For local database connections using AWS Managed Active Directory, you can use SQL authentication or Windows authentication. For Windows authentication, use

DB_instance_name.*fully_qualified_domain_name* as the server name of the connection string.

An example is `myssisinstance.corp-ad.example.com`, where `myssisinstance` is the DB instance name and `corp-ad.example.com` is the fully qualified domain name.

- For remote connections, always use SQL authentication.
- For local database connections using self-managed Active Directory, you can use SQL authentication or Windows authentication. For Windows authentication, use `.` or *`LocalHost`* as the server name of the connection string.

Creating an SSIS proxy

To be able to schedule SSIS packages using SQL Server Agent, create an SSIS credential and an SSIS proxy. Run these procedures as a Windows-authenticated user.

To create the SSIS credential

- Create the credential for the proxy. To do this, you can use SSMS or the following SQL statement.

```
USE [master]
GO
CREATE CREDENTIAL [SSIS_Credential] WITH IDENTITY = N'mydomain\user_name', SECRET =
 N'mysecret'
GO
```

> **Note**
>
> `IDENTITY` must be a domain-authenticated login. Replace *`mysecret`* with the password for the domain-authenticated login.
> Whenever the SSISDB primary host is changed, alter the SSIS proxy credentials to allow the new host to access them.

To create the SSIS proxy

1. Use the following SQL statement to create the proxy.

```
USE [msdb]
GO
EXEC msdb.dbo.sp_add_proxy
 @proxy_name=N'SSIS_Proxy',@credential_name=N'SSIS_Credential',@description=N''
GO
```

2. Use the following SQL statement to grant access to the proxy to other users.

```
USE [msdb]
GO
EXEC msdb.dbo.sp_grant_login_to_proxy
 @proxy_name=N'SSIS_Proxy',@login_name=N'mydomain\user_name'
GO
```

3. Use the following SQL statement to give the SSIS subsystem access to the proxy.

```
USE [msdb]
GO
EXEC msdb.dbo.rds_sqlagent_proxy
 @task_type='GRANT_SUBSYSTEM_ACCESS',@proxy_name='SSIS_Proxy',@proxy_subsystem='SSIS'
GO
```

To view the proxy and grants on the proxy

1. Use the following SQL statement to view the grantees of the proxy.

```
USE [msdb]
GO
EXEC sp_help_proxy
GO
```

2. Use the following SQL statement to view the subsystem grants.

```
USE [msdb]
GO
EXEC msdb.dbo.sp_enum_proxy_for_subsystem
GO
```

Scheduling an SSIS package using SQL Server Agent

After you create the credential and proxy and grant SSIS access to the proxy, you can create a SQL Server Agent job to schedule the SSIS package.

To schedule the SSIS package

- You can use SSMS or T-SQL for creating the SQL Server Agent job. The following example uses T-SQL.

```
USE [msdb]
GO
DECLARE @jobId BINARY(16)
EXEC msdb.dbo.sp_add_job @job_name=N'MYSSISJob',
@enabled=1,
@notify_level_eventlog=0,
@notify_level_email=2,
@notify_level_page=2,
@delete_level=0,
@category_name=N'[Uncategorized (Local)]',
@job_id = @jobId OUTPUT
GO
EXEC msdb.dbo.sp_add_jobserver @job_name=N'MYSSISJob',@server_name=N'(local)'
GO
EXEC msdb.dbo.sp_add_jobstep
 @job_name=N'MYSSISJob',@step_name=N'ExecuteSSISPackage',
@step_id=1,
@cmdexec_success_code=0,
@on_success_action=1,
@on_fail_action=2,
@retry_attempts=0,
@retry_interval=0,
@os_run_priority=0,
@subsystem=N'SSIS',
@command=N'/ISSERVER "\"\SSISDB\MySSISFolder\MySSISProject\MySSISPackage.dtsx\"" /
SERVER "\"my-rds-ssis-instance.corp-ad.company.com/\""
/Par "\"$ServerOption::LOGGING_LEVEL(Int16)\"";1 /Par
 "\"$ServerOption::SYNCHRONIZED(Boolean)\"";True /CALLERINFO SQLAGENT /REPORTING
 E',
@database_name=N'master',
@flags=0,
@proxy_name=N'SSIS_Proxy'
GO
```

Revoking SSIS access from the proxy

You can revoke access to the SSIS subsystem and delete the SSIS proxy using the following stored procedures.

To revoke access and delete the proxy

1. Revoke subsystem access.

```
USE [msdb]
GO
EXEC msdb.dbo.rds_sqlagent_proxy
 @task_type='REVOKE_SUBSYSTEM_ACCESS',@proxy_name='SSIS_Proxy',@proxy_subsystem='SSIS'
GO
```

2. Revoke the grants on the proxy.

```
USE [msdb]
GO
EXEC msdb.dbo.sp_revoke_login_from_proxy
 @proxy_name=N'SSIS_Proxy',@name=N'mydomain\user_name'
GO
```

3. Delete the proxy.

```
USE [msdb]
GO
EXEC dbo.sp_delete_proxy @proxy_name = N'SSIS_Proxy'
GO
```

Disable and drop SSIS database

Use the following steps to disable or drop SSIS databases:

Topics

- Disabling SSIS
- Dropping the SSISDB database

Disabling SSIS

To disable SSIS, remove the `SSIS` option from its option group.

> **Important**
>
> Removing the option doesn't delete the SSISDB database, so you can safely remove the option without losing the SSIS projects.
> You can re-enable the `SSIS` option after removal to reuse the SSIS projects that were previously deployed to the SSIS catalog.

Console

The following procedure removes the `SSIS` option.

To remove the SSIS option from its option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group with the `SSIS` option (`ssis-se-2016` in the previous examples).
4. Choose **Delete option**.
5. Under **Deletion options**, choose **SSIS** for **Options to delete**.
6. Under **Apply immediately**, choose **Yes** to delete the option immediately, or **No** to delete it at the next maintenance window.
7. Choose **Delete**.

CLI

The following procedure removes the `SSIS` option.

To remove the SSIS option from its option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds remove-option-from-option-group \
    --option-group-name ssis-se-2016 \
    --options SSIS \
    --apply-immediately
```

For Windows:

```
aws rds remove-option-from-option-group ^
    --option-group-name ssis-se-2016 ^
    --options SSIS ^
    --apply-immediately
```

Dropping the SSISDB database

After removing the SSIS option, the SSISDB database isn't deleted. To drop the SSISDB database, use the `rds_drop_ssis_database` stored procedure after removing the SSIS option.

To drop the SSIS database

- Use the following stored procedure.

```
USE [msdb]
GO
EXEC dbo.rds_drop_ssis_database
GO
```

After dropping the SSISDB database, if you re-enable the SSIS option you get a fresh SSISDB catalog.

Support for SQL Server Reporting Services in Amazon RDS for SQL Server

Microsoft SQL Server Reporting Services (SSRS) is a server-based application used for report generation and distribution. It's part of a suite of SQL Server services that also includes SQL Server Analysis Services (SSAS) and SQL Server Integration Services (SSIS). SSRS is a service built on top of SQL Server. You can use it to collect data from various data sources and present it in a way that's easily understandable and ready for analysis.

Amazon RDS for SQL Server supports running SSRS directly on RDS DB instances. You can use SSRS with existing or new DB instances.

RDS supports SSRS for SQL Server Standard and Enterprise Editions on the following versions:

- SQL Server 2022, all versions
- SQL Server 2019, version 15.00.4043.16.v1 and higher
- SQL Server 2017, version 14.00.3223.3.v1 and higher
- SQL Server 2016, version 13.00.5820.21.v1 and higher

Contents

- Limitations and recommendations
- Turning on SSRS
  - Creating an option group for SSRS
  - Adding the SSRS option to your option group
  - Associating your option group with your DB instance
  - Allowing inbound access to your VPC security group
- Report server databases
- SSRS log files
- Accessing the SSRS web portal
  - Using SSL on RDS
  - Granting access to domain users
  - Accessing the web portal
- Deploying reports and configuring report data sources

- Deploying reports to SSRS
- Configuring the report data source
- Using SSRS Email to send reports
- Revoking system-level permissions
- Monitoring the status of a task
- Disabling and deleting SSRS databases
  - Turning off SSRS
  - Deleting the SSRS databases

Limitations and recommendations

The following limitations and recommendations apply to running SSRS on RDS for SQL Server:

- You can't use SSRS on DB instances that have read replicas.
- Instances must use self-managed Active Directory or AWS Directory Service for Microsoft Active Directory for SSRS web portal and web server authentication. For more information, see Working with Active Directory with RDS for SQL Server.
- You can't back up the reporting server databases that are created with the SSRS option.
- Importing and restoring report server databases from other instances of SSRS isn't supported. For more information, see Report server databases.
- You can't configure SSRS to listen on the default SSL port (443). The allowed values are 1150–49511, except 1234, 1434, 3260, 3343, 3389, and 47001.
- Subscriptions through a Microsoft Windows file share aren't supported.
- Using Reporting Services Configuration Manager isn't supported.
- Creating and modifying roles isn't supported.
- Modifying report server properties isn't supported.
- System administrator and system user roles aren't granted.
- You can't edit system-level role assignments through the web portal.

Turning on SSRS

Use the following process to turn on SSRS for your DB instance:

1. Create a new option group, or choose an existing option group.
2. Add the SSRS option to the option group.
3. Associate the option group with the DB instance.
4. Allow inbound access to the virtual private cloud (VPC) security group for the SSRS listener port.

Creating an option group for SSRS

To work with SSRS, create an option group that corresponds to the SQL Server engine and version of the DB instance that you plan to use. To do this, use the AWS Management Console or the AWS CLI.

> **Note**
>
> You can also use an existing option group if it's for the correct SQL Server engine and version.

Console

The following procedure creates an option group for SQL Server Standard Edition 2017.

To create the option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** pane, do the following:
   a. For **Name**, enter a name for the option group that is unique within your AWS account, such as **`ssrs-se-2017`**. The name can contain only letters, digits, and hyphens.
   b. For **Description**, enter a brief description of the option group, such as **`SSRS option group for SQL Server SE 2017`**. The description is used for display purposes.
   c. For **Engine**, choose **sqlserver-se**.
   d. For **Major engine version**, choose **14.00**.
5. Choose **Create**.

CLI

The following procedure creates an option group for SQL Server Standard Edition 2017.

To create the option group

- Run one of the following commands.

Example

For Linux, macOS, or Unix:

```
aws rds create-option-group \
    --option-group-name ssrs-se-2017 \
    --engine-name sqlserver-se \
    --major-engine-version 14.00 \
    --option-group-description "SSRS option group for SQL Server SE 2017"
```

For Windows:

```
aws rds create-option-group ^
    --option-group-name ssrs-se-2017 ^
    --engine-name sqlserver-se ^
    --major-engine-version 14.00 ^
    --option-group-description "SSRS option group for SQL Server SE 2017"
```

Adding the SSRS option to your option group

Next, use the AWS Management Console or the AWS CLI to add the SSRS option to your option group.

Console

To add the SSRS option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group that you just created, then choose **Add option**.
4. Under **Option details**, choose **SSRS** for **Option name**.

5. Under **Option settings**, do the following:

   a. Enter the port for the SSRS service to listen on. The default is 8443. For a list of allowed values, see Limitations and recommendations.

   b. Enter a value for **Max memory**.

      Max memory specifies the upper threshold above which no new memory allocation requests are granted to report server applications. The number is a percentage of the total memory of the DB instance. The allowed values are 10–80.

   c. For **Security groups**, choose the VPC security group to associate with the option. Use the same security group that is associated with your DB instance.

6. To use SSRS Email to send reports, choose the **Configure email delivery options** check box under **Email delivery in reporting services**, and then do the following:

   a. For **Sender email address**, enter the email address to use in the **From** field of messages sent by SSRS Email.

      Specify a user account that has permission to send mail from the SMTP server.

   b. For **SMTP server**, specify the SMTP server or gateway to use.

      It can be an IP address, the NetBIOS name of a computer on your corporate intranet, or a fully qualified domain name.

   c. For **SMTP port**, enter the port to use to connect to the mail server. The default is 25.

   d. To use authentication:

      i. Select the **Use authentication** check box.

      ii. For **Secret Amazon Resource Name (ARN)** enter the AWS Secrets Manager ARN for the user credentials.

      Use the following format:

      ```
      arn:aws:secretsmanager:Region:AccountId:secret:SecretName-6RandomChara
      ```

      For example:

      ```
      arn:aws:secretsmanager:us-west-2:123456789012:secret:MySecret-
      a1b2c3
      ```

      For more information on creating the secret, see Using SSRS Email to send reports.

e. Select the **Use Secure Sockets Layer (SSL)** check box to encrypt email messages using SSL.

7. Under **Scheduling**, choose whether to add the option immediately or at the next maintenance window.
8. Choose **Add option**.

CLI

To add the SSRS option

1. Create a JSON file, for example `ssrs-option.json`.

   a. Set the following required parameters:

   - `OptionGroupName` – The name of option group that you created or chose previously (`ssrs-se-2017` in the following example).
   - `Port` – The port for the SSRS service to listen on. The default is 8443. For a list of allowed values, see Limitations and recommendations.
   - `VpcSecurityGroupMemberships` – VPC security group memberships for your RDS DB instance.
   - `MAX_MEMORY` – The upper threshold above which no new memory allocation requests are granted to report server applications. The number is a percentage of the total memory of the DB instance. The allowed values are 10–80.

   b. (Optional) Set the following parameters to use SSRS Email:

   - `SMTP_ENABLE_EMAIL` – Set to `true` to use SSRS Email. The default is `false`.
   - `SMTP_SENDER_EMAIL_ADDRESS` – The email address to use in the **From** field of messages sent by SSRS Email. Specify a user account that has permission to send mail from the SMTP server.
   - `SMTP_SERVER` – The SMTP server or gateway to use. It can be an IP address, the NetBIOS name of a computer on your corporate intranet, or a fully qualified domain name.
   - `SMTP_PORT` – The port to use to connect to the mail server. The default is 25.
   - `SMTP_USE_SSL` – Set to `true` to encrypt email messages using SSL. The default is `true`.

- SMTP_EMAIL_CREDENTIALS_SECRET_ARN – The Secrets Manager ARN that holds the user credentials. Use the following format:

  arn:aws:secretsmanager:*Region*:*AccountId*:secret:*SecretName*-*6RandomCharact*

  For more information on creating the secret, see Using SSRS Email to send reports.
- SMTP_USE_ANONYMOUS_AUTHENTICATION – Set to `true` and don't include SMTP_EMAIL_CREDENTIALS_SECRET_ARN if you don't want to use authentication.

  The default is `false` when SMTP_ENABLE_EMAIL is `true`.

The following example includes the SSRS Email parameters, using the secret ARN.

```
{
"OptionGroupName": "ssrs-se-2017",
"OptionsToInclude": [
 {
 "OptionName": "SSRS",
 "Port": 8443,
 "VpcSecurityGroupMemberships": ["sg-0abcdef123"],
 "OptionSettings": [
            {"Name": "MAX_MEMORY","Value": "60"},
            {"Name": "SMTP_ENABLE_EMAIL","Value": "true"}
            {"Name": "SMTP_SENDER_EMAIL_ADDRESS","Value": "nobody@example.com"},
            {"Name": "SMTP_SERVER","Value": "email-smtp.us-west-2.amazonaws.com"},
            {"Name": "SMTP_PORT","Value": "25"},
            {"Name": "SMTP_USE_SSL","Value": "true"},
            {"Name": "SMTP_EMAIL_CREDENTIALS_SECRET_ARN","Value":
 "arn:aws:secretsmanager:us-west-2:123456789012:secret:MySecret-a1b2c3"}
            ]
 }],
"ApplyImmediately": true
}
```

2. Add the SSRS option to the option group.

   Example

   For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
```

```
    --cli-input-json file://ssrs-option.json \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --cli-input-json file://ssrs-option.json ^
    --apply-immediately
```

Associating your option group with your DB instance

Use the AWS Management Console or the AWS CLI to associate your option group with your DB instance.

If you use an existing DB instance, it must already have an Active Directory domain and AWS Identity and Access Management (IAM) role associated with it. If you create a new instance, specify an existing Active Directory domain and IAM role. For more information, see Working with Active Directory with RDS for SQL Server.

Console

You can associate your option group with a new or existing DB instance:

- For a new DB instance, associate the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, modify the instance and associate the new option group. For more information, see Modifying an Amazon RDS DB instance.

CLI

You can associate your option group with a new or existing DB instance.

To create a DB instance that uses your option group

- Specify the same DB engine type and major version as you used when creating the option group.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier myssrsinstance \
    --db-instance-class db.m5.2xlarge \
    --engine sqlserver-se \
    --engine-version 14.00.3223.3.v1 \
    --allocated-storage 100 \
    --manage-master-user-password  \
    --master-username admin \
    --storage-type gp2 \
    --license-model li \
    --domain-iam-role-name my-directory-iam-role \
    --domain my-domain-id \
    --option-group-name ssrs-se-2017
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier myssrsinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-se ^
    --engine-version 14.00.3223.3.v1 ^
    --allocated-storage 100 ^
    --manage-master-user-password ^
    --master-username admin ^
    --storage-type gp2 ^
    --license-model li ^
    --domain-iam-role-name my-directory-iam-role ^
    --domain my-domain-id ^
    --option-group-name ssrs-se-2017
```

To modify a DB instance to use your option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier myssrsinstance \
```

```
    --option-group-name ssrs-se-2017 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier myssrsinstance ^
    --option-group-name ssrs-se-2017 ^
    --apply-immediately
```

Allowing inbound access to your VPC security group

To allow inbound access to the VPC security group associated with your DB instance, create an inbound rule for the specified SSRS listener port. For more information about setting up security groups, see Provide access to your DB instance in your VPC by creating a security group.

Report server databases

When your DB instance is associated with the SSRS option, two new databases are created on your DB instance:

- `rdsadmin_ReportServer`
- `rdsadmin_ReportServerTempDB`

These databases act as the ReportServer and ReportServerTempDB databases. SSRS stores its data in the ReportServer database and caches its data in the ReportServerTempDB database. For more information, see Report Server Database in the Microsoft documentation.

RDS owns and manages these databases, so database operations on them such as ALTER and DROP aren't permitted. Access isn't permitted on the `rdsadmin_ReportServerTempDB` database. However, you can perform read operations on the `rdsadmin_ReportServer`database.

SSRS log files

You can list, view, and download SSRS log files. SSRS log files follow a naming convention of ReportServerService_*timestamp*.log. These report server logs are located in the `D:\rdsdbdata\Log\SSRS` directory. (The `D:\rdsdbdata\Log` directory is also the parent directory for error logs and SQL Server Agent logs.). For more information, see Viewing and listing database log files.

For existing SSRS instances, restarting the SSRS service might be necessary to access report server logs. You can restart the service by updating the SSRS option.

For more information, see Working with Amazon RDS for Microsoft SQL Server logs.

Accessing the SSRS web portal

Use the following process to access the SSRS web portal:

1. Turn on Secure Sockets Layer (SSL).
2. Grant access to domain users.
3. Access the web portal using a browser and the domain user credentials.

Using SSL on RDS

SSRS uses the HTTPS SSL protocol for its connections. To work with this protocol, import an SSL certificate into the Microsoft Windows operating system on your client computer.

For more information on SSL certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . For more information about using SSL with SQL Server, see Using SSL with a Microsoft SQL Server DB instance.

Granting access to domain users

In a new SSRS activation, there are no role assignments in SSRS. To give a domain user or user group access to the web portal, RDS provides a stored procedure.

To grant access to a domain user on the web portal

- Use the following stored procedure.

```
exec msdb.dbo.rds_msbi_task
@task_type='SSRS_GRANT_PORTAL_PERMISSION',
@ssrs_group_or_username=N'AD_domain\user';
```

The domain user or user group is granted the RDS_SSRS_ROLE system role. This role has the following system-level tasks granted to it:

- Run reports

- Manage jobs
- Manage shared schedules
- View shared schedules

The item-level role of `Content Manager` on the root folder is also granted.

Accessing the web portal

After the `SSRS_GRANT_PORTAL_PERMISSION` task finishes successfully, you have access to the portal using a web browser. The web portal URL has the following format.

```
https://rds_endpoint:port/Reports
```

In this format, the following applies:

- *`rds_endpoint`* – The endpoint for the RDS DB instance that you're using with SSRS.

  You can find the endpoint on the **Connectivity & security** tab for your DB instance. For more information, see Connecting to your Microsoft SQL Server DB instance.
- *`port`* – The listener port for SSRS that you set in the SSRS option.

To access the web portal

1. Enter the web portal URL in your browser.

   ```
   https://myssrsinstance.cg034itsfake.us-east-1.rds.amazonaws.com:8443/Reports
   ```
2. Log in with the credentials for a domain user that you granted access with the `SSRS_GRANT_PORTAL_PERMISSION` task.

Deploying reports and configuring report data sources

Use the following procedures to deploy reports to SSRS and configure the reporting data sources:

Topics

- Deploying reports to SSRS
- Configuring the report data source

Deploying reports to SSRS

After you have access to the web portal, you can deploy reports to it. You can use the Upload tool in the web portal to upload reports, or deploy directly from SQL Server data tools (SSDT). When deploying from SSDT, ensure the following:

- The user who launched SSDT has access to the SSRS web portal.
- The `TargetServerURL` value in the SSRS project properties is set to the HTTPS endpoint of the RDS DB instance suffixed with `ReportServer`, for example:

```
https://myssrsinstance.cg034itsfake.us-east-1.rds.amazonaws.com:8443/ReportServer
```

Configuring the report data source

After you deploy a report to SSRS, you should configure the report data source. When configuring the report data source, ensure the following:

- For RDS for SQL Server DB instances joined to AWS Directory Service for Microsoft Active Directory, use the fully qualified domain name (FQDN) as the data source name of the connection string. An example is *`myssrsinstance.corp-ad.example.com`*, where *`myssrsinstance`* is the DB instance name and *`corp-ad.example.com`* is the fully qualified domain name.
- For RDS for SQL Server DB instances joined to self-managed Active Directory, use `.`, or *`LocalHost`* as the data source name of the connection string.

Using SSRS Email to send reports

SSRS includes the SSRS Email extension, which you can use to send reports to users.

To configure SSRS Email, use the SSRS option settings. For more information, see Adding the SSRS option to your option group.

After configuring SSRS Email, you can subscribe to reports on the report server. For more information, see Email delivery in Reporting Services in the Microsoft documentation.

Integration with AWS Secrets Manager is required for SSRS Email to function on RDS. To integrate with Secrets Manager, you create a secret.

> **Note**
>
> If you change the secret later, you also have to update the SSRS option in the option group.

To create a secret for SSRS Email

1. Follow the steps in Create a secret in the *AWS Secrets Manager User Guide*.

   a. For **Select secret type**, choose **Other type of secrets**.

   b. For **Key/value pairs**, enter the following:

   - **SMTP_USERNAME** – Enter a user with permission to send mail from the SMTP server.
   - **SMTP_PASSWORD** – Enter a password for the SMTP user.

   c. For **Encryption key**, don't use the default AWS KMS key. Use your own existing key, or create a new one.

   The KMS key policy must allow the `kms:Decrypt` action, for example:

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "rds.amazonaws.com"
        ]
    },
    "Action": [
        "kms:Decrypt"
    ],
    "Resource": "*"
}
```

2. Follow the steps in Attach a permissions policy to a secret in the *AWS Secrets Manager User Guide*. The permissions policy gives the `secretsmanager:GetSecretValue` action to the `rds.amazonaws.com` service principal.

   We recommend that you use the `aws:sourceAccount` and `aws:sourceArn` conditions in the policy to avoid the *confused deputy* problem. Use your AWS account for

`aws:sourceAccount` and the option group ARN for `aws:sourceArn`. For more information, see Preventing cross-service confused deputy problems.

The following example shows a permissions policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [ {
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "rds.amazonaws.com"
    },
    "Action" : "secretsmanager:GetSecretValue",
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "aws:sourceAccount" : "123456789012"
      },
      "ArnLike" : {
        "aws:sourceArn" : "arn:aws:rds:us-west-2:123456789012:og:ssrs-
se-2017"
      }
    }
  } ]
}
```

For more examples, see Permissions policy examples for AWS Secrets Manager in the *AWS Secrets Manager User Guide*.

Revoking system-level permissions

The `RDS_SSRS_ROLE` system role doesn't have sufficient permissions to delete system-level role assignments. To remove a user or user group from `RDS_SSRS_ROLE`, use the same stored procedure that you used to grant the role but use the `SSRS_REVOKE_PORTAL_PERMISSION` task type.

To revoke access from a domain user for the web portal

- Use the following stored procedure.

```
exec msdb.dbo.rds_msbi_task
@task_type='SSRS_REVOKE_PORTAL_PERMISSION',
@ssrs_group_or_username=N'AD_domain\user';
```

Doing this deletes the user from the RDS_SSRS_ROLE system role. It also deletes the user from the `Content Manager` item-level role if the user has it.

Monitoring the status of a task

To track the status of your granting or revoking task, call the `rds_fn_task_status` function. It takes two parameters. The first parameter should always be NULL because it doesn't apply to SSRS. The second parameter accepts a task ID.

To see a list of all tasks, set the first parameter to NULL and the second parameter to `0`, as shown in the following example.

```
SELECT * FROM msdb.dbo.rds_fn_task_status(NULL,0);
```

To get a specific task, set the first parameter to NULL and the second parameter to the task ID, as shown in the following example.

```
SELECT * FROM msdb.dbo.rds_fn_task_status(NULL,42);
```

The `rds_fn_task_status` function returns the following information.

Output parameter	Description
`task_id`	The ID of the task.
`task_type`	For SSRS, tasks can have the following task types: • `SSRS_GRANT_PORTAL_PERMISSION` • `SSRS_REVOKE_PORTAL_PERMISSION`

Output parameter	Description
database_name	Not applicable to SSRS tasks.
% complete	The progress of the task as a percentage.
duration (mins)	The amount of time spent on the task, in minutes.
lifecycle	The status of the task. Possible statuses are the following: • CREATED – After you call one of the SSRS stored procedures, a task is created and the status is set to CREATED. • IN_PROGRESS – After a task starts, the status is set to IN_PROGRESS . It can take up to five minutes for the status to change from CREATED to IN_PROGRESS . • SUCCESS – After a task completes, the status is set to SUCCESS. • ERROR – If a task fails, the status is set to ERROR. For more information about the error, see the task_info column. • CANCEL_REQUESTED – After you call the rds_cancel_task stored procedure, the status of the task is set to CANCEL_REQUESTED . • CANCELLED – After a task is successfully canceled, the status of the task is set to CANCELLED .

Output parameter	Description
task_info	Additional information about the task. If an error occurs during processing, this column contains information about the error.
last_updated	The date and time that the task status was last updated.
created_at	The date and time that the task was created.
S3_object_arn	Not applicable to SSRS tasks.
overwrite_S3_backup_file	Not applicable to SSRS tasks.
KMS_master_key_arn	Not applicable to SSRS tasks.
filepath	Not applicable to SSRS tasks.
overwrite_file	Not applicable to SSRS tasks.
task_metadata	Metadata associated with the SSRS task.

Disabling and deleting SSRS databases

Use the following procedures to disable SSRS and delete SSRS databases:

Topics

- Turning off SSRS
- Deleting the SSRS databases

Turning off SSRS

To turn off SSRS, remove the SSRS option from its option group. Removing the option doesn't delete the SSRS databases. For more information, see Deleting the SSRS databases.

You can turn SSRS on again by adding back the SSRS option. If you have also deleted the SSRS databases, readding the option on the same DB instance creates new report server databases.

Console

To remove the SSRS option from its option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group with the SSRS option (`ssrs-se-2017` in the previous examples).
4. Choose **Delete option**.
5. Under **Deletion options**, choose **SSRS** for **Options to delete**.
6. Under **Apply immediately**, choose **Yes** to delete the option immediately, or **No** to delete it at the next maintenance window.
7. Choose **Delete**.

CLI

To remove the SSRS option from its option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds remove-option-from-option-group \
    --option-group-name ssrs-se-2017 \
    --options SSRS \
    --apply-immediately
```

  For Windows:

```
aws rds remove-option-from-option-group ^
    --option-group-name ssrs-se-2017 ^
    --options SSRS ^
    --apply-immediately
```

Deleting the SSRS databases

Removing the SSRS option doesn't delete the report server databases. To delete them, use the following stored procedure.

To delete the report server databases, be sure to remove the SSRS option first.

To delete the SSRS databases

- Use the following stored procedure.

```
exec msdb.dbo.rds_drop_ssrs_databases
```

Support for Microsoft Distributed Transaction Coordinator in RDS for SQL Server

A *distributed transaction* is a database transaction in which two or more network hosts are involved. RDS for SQL Server supports distributed transactions among hosts, where a single host can be one of the following:

- RDS for SQL Server DB instance
- On-premises SQL Server host
- Amazon EC2 host with SQL Server installed
- Any other EC2 host or RDS DB instance with a database engine that supports distributed transactions

In RDS, starting with SQL Server 2012 (version 11.00.5058.0.v1 and later), all editions of RDS for SQL Server support distributed transactions. The support is provided using Microsoft Distributed Transaction Coordinator (MSDTC). For in-depth information about MSDTC, see Distributed Transaction Coordinator in the Microsoft documentation.

Contents

- Limitations
- Enabling MSDTC
  - Creating the option group for MSDTC
  - Adding the MSDTC option to the option group
  - Creating the parameter group for MSDTC
  - Modifying the parameter for MSDTC
  - Associating the option group and parameter group with the DB instance
  - Modifying the MSDTC option
- Using transactions
  - Using distributed transactions
  - Using XA transactions
  - Using transaction tracing
- Disabling MSDTC
- Troubleshooting MSDTC for RDS for SQL Server

Limitations

The following limitations apply to using MSDTC on RDS for SQL Server:

- MSDTC isn't supported on instances using SQL Server Database Mirroring. For more information, see Transactions - availability groups and database mirroring.
- The `in-doubt xact resolution` parameter must be set to 1 or 2. For more information, see Modifying the parameter for MSDTC.
- MSDTC requires all hosts participating in distributed transactions to be resolvable using their host names. RDS automatically maintains this functionality for domain-joined instances. However, for standalone instances make sure to configure the DNS server manually.
- Java Database Connectivity (JDBC) XA transactions are supported for SQL Server 2017 version 14.00.3223.3 and higher, and SQL Server 2019.
- Distributed transactions that depend on client dynamic link libraries (DLLs) on RDS instances aren't supported.
- Using custom XA dynamic link libraries isn't supported.

Enabling MSDTC

Use the following process to enable MSDTC for your DB instance:

1. Create a new option group, or choose an existing option group.
2. Add the `MSDTC` option to the option group.
3. Create a new parameter group, or choose an existing parameter group.
4. Modify the parameter group to set the `in-doubt xact resolution` parameter to 1 or 2.
5. Associate the option group and parameter group with the DB instance.

Creating the option group for MSDTC

Use the AWS Management Console or the AWS CLI to create an option group that corresponds to the SQL Server engine and version of your DB instance.

> **Note**
>
> You can also use an existing option group if it's for the correct SQL Server engine and version.

Console

The following procedure creates an option group for SQL Server Standard Edition 2016.

To create the option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** pane, do the following:
   a. For **Name**, enter a name for the option group that is unique within your AWS account, such as **`msdtc-se-2016`**. The name can contain only letters, digits, and hyphens.
   b. For **Description**, enter a brief description of the option group, such as **`MSDTC option group for SQL Server SE 2016`**. The description is used for display purposes.
   c. For **Engine**, choose **sqlserver-se**.
   d. For **Major engine version**, choose **13.00**.
5. Choose **Create**.

CLI

The following example creates an option group for SQL Server Standard Edition 2016.

To create the option group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-option-group \
    --option-group-name msdtc-se-2016 \
    --engine-name sqlserver-se \
    --major-engine-version 13.00 \
    --option-group-description "MSDTC option group for SQL Server SE 2016"
```

For Windows:

```
aws rds create-option-group ^
    --option-group-name msdtc-se-2016 ^
    --engine-name sqlserver-se ^
    --major-engine-version 13.00 ^
    --option-group-description "MSDTC option group for SQL Server SE 2016"
```

Adding the MSDTC option to the option group

Next, use the AWS Management Console or the AWS CLI to add the MSDTC option to the option group.

The following option settings are required:

- **Port** – The port that you use to access MSDTC. Allowed values are 1150–49151 except for 1234, 1434, 3260, 3343, 3389, and 47001. The default value is 5000.

  Make sure that the port you want to use is enabled in your firewall rules. Also, make sure as needed that this port is enabled in the inbound and outbound rules for the security group associated with your DB instance. For more information, see Can't connect to Amazon RDS DB instance.
- **Security groups** – The VPC security group memberships for your RDS DB instance.
- **Authentication type** – The authentication mode between hosts. The following authentication types are supported:
  - Mutual – The RDS instances are mutually authenticated to each other using integrated authentication. If this option is selected, all instances associated with this option group must be domain-joined.
  - None – No authentication is performed between hosts. We don't recommend using this mode in production environments.

- **Transaction log size** – The size of the MSDTC transaction log. Allowed values are 4–1024 MB. The default size is 4 MB.

The following option settings are optional:

- **Enable inbound connections** – Whether to allow inbound MSDTC connections to instances associated with this option group.
- **Enable outbound connections** – Whether to allow outbound MSDTC connections from instances associated with this option group.
- **Enable XA** – Whether to allow XA transactions. For more information on the XA protocol, see XA specification.
- **Enable SNA LU** – Whether to allow the SNA LU protocol to be used for distributed transactions. For more information on SNA LU protocol support, see Managing IBM CICS LU 6.2 transactions in the Microsoft documentation.

Console

To add the MSDTC option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group that you just created.
4. Choose **Add option**.
5. Under **Option details**, choose **MSDTC** for **Option name**.
6. Under **Option settings**:
    a. For **Port**, enter the port number for accessing MSDTC. The default is **5000**.
    b. For **Security groups**, choose the VPC security group to associate with the option.
    c. For **Authentication type**, choose **Mutual** or **None**.
    d. For **Transaction log size**, enter a value from 4–1024. The default is **4**.
7. Under **Additional configuration**, do the following:
    a. For **Connections**, as needed choose **Enable inbound connections** and **Enable outbound connections**.

b. For **Allowed protocols**, as needed choose **Enable XA** and **Enable SNA LU**.

8. Under **Scheduling**, choose whether to add the option immediately or at the next maintenance window.
9. Choose **Add option**.

   To add this option, no reboot is required.

CLI

To add the MSDTC option

1. Create a JSON file, for example `msdtc-option.json`, with the following required parameters.

```
{
"OptionGroupName":"msdtc-se-2016",
"OptionsToInclude": [
 {
 "OptionName":"MSDTC",
 "Port":5000,
 "VpcSecurityGroupMemberships":["sg-0abcdef123"],
 "OptionSettings":[{"Name":"AUTHENTICATION","Value":"MUTUAL"},
{"Name":"TRANSACTION_LOG_SIZE","Value":"4"}]
 }],
"ApplyImmediately": true
}
```

2. Add the MSDTC option to the option group.

   Example

   For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --cli-input-json file://msdtc-option.json \
    --apply-immediately
```

   For Windows:

```
aws rds add-option-to-option-group ^
```

```
    --cli-input-json file://msdtc-option.json ^
    --apply-immediately
```

No reboot is required.

Creating the parameter group for MSDTC

Create or modify a parameter group for the `in-doubt xact resolution` parameter that corresponds to the SQL Server edition and version of your DB instance.

Console

The following example creates a parameter group for SQL Server Standard Edition 2016.

To create the parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
4. In the **Create parameter group** pane, do the following:
   a. For **Parameter group family**, choose **sqlserver-se-13.0**.
   b. For **Group name**, enter an identifier for the parameter group, such as **`msdtc-sqlserver-se-13`**.
   c. For **Description**, enter **`in-doubt xact resolution`**.
5. Choose **Create**.

CLI

The following example creates a parameter group for SQL Server Standard Edition 2016.

To create the parameter group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name msdtc-sqlserver-se-13 \
    --db-parameter-group-family "sqlserver-se-13.0" \
    --description "in-doubt xact resolution"
```

For Windows:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name msdtc-sqlserver-se-13 ^
    --db-parameter-group-family "sqlserver-se-13.0" ^
    --description "in-doubt xact resolution"
```

Modifying the parameter for MSDTC

Modify the `in-doubt xact resolution` parameter in the parameter group that corresponds to the SQL Server edition and version of your DB instance.

For MSDTC, set the `in-doubt xact resolution` parameter to one of the following:

- 1 – `Presume commit`. Any MSDTC in-doubt transactions are presumed to have committed.
- 2 – `Presume abort`. Any MSDTC in-doubt transactions are presumed to have stopped.

For more information, see in-doubt xact resolution server configuration option in the Microsoft documentation.

Console

The following example modifies the parameter group that you created for SQL Server Standard Edition 2016.

To modify the parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**.
3. Choose the parameter group, such as **msdtc-sqlserver-se-13**.
4. Under **Parameters**, filter the parameter list for **`xact`**.

5. Choose **in-doubt xact resolution**.
6. Choose **Edit parameters**.
7. Enter **1** or **2**.
8. Choose **Save changes**.

CLI

The following example modifies the parameter group that you created for SQL Server Standard Edition 2016.

To modify the parameter group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name msdtc-sqlserver-se-13 \
    --parameters "ParameterName='in-doubt xact
 resolution',ParameterValue=1,ApplyMethod=immediate"
```

  For Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name msdtc-sqlserver-se-13 ^
    --parameters "ParameterName='in-doubt xact
 resolution',ParameterValue=1,ApplyMethod=immediate"
```

Associating the option group and parameter group with the DB instance

You can use the AWS Management Console or the AWS CLI to associate the MSDTC option group and parameter group with the DB instance.

Console

You can associate the MSDTC option group and parameter group with a new or existing DB instance.

- For a new DB instance, associate them when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, associate them by modifying the instance. For more information, see Modifying an Amazon RDS DB instance.

> **Note**
>
> If you use an domain-joined existing DB instance, it must already have an Active Directory domain and AWS Identity and Access Management (IAM) role associated with it. If you create a new domain-joined instance, specify an existing Active Directory domain and IAM role. For more information, see Working with AWS Managed Active Directory with RDS for SQL Server.

CLI

You can associate the MSDTC option group and parameter group with a new or existing DB instance.

> **Note**
>
> If you use an existing domain-joined DB instance, it must already have an Active Directory domain and IAM role associated with it. If you create a new domain-joined instance, specify an existing Active Directory domain and IAM role. For more information, see Working with AWS Managed Active Directory with RDS for SQL Server.

To create a DB instance with the MSDTC option group and parameter group

- Specify the same DB engine type and major version as you used when creating the option group.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.m5.2xlarge \
```

```
    --engine sqlserver-se \
    --engine-version 13.00.5426.0.v1 \
    --allocated-storage 100 \
    --manage-master-user-password \
    --master-username admin \
    --storage-type gp2 \
    --license-model li \
    --domain-iam-role-name my-directory-iam-role \
    --domain my-domain-id \
    --option-group-name msdtc-se-2016 \
    --db-parameter-group-name msdtc-sqlserver-se-13
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-se ^
    --engine-version 13.00.5426.0.v1 ^
    --allocated-storage 100 ^
    --manage-master-user-password ^
    --master-username admin ^
    --storage-type gp2 ^
    --license-model li ^
    --domain-iam-role-name my-directory-iam-role ^
    --domain my-domain-id ^
    --option-group-name msdtc-se-2016 ^
    --db-parameter-group-name msdtc-sqlserver-se-13
```

To modify a DB instance and associate the MSDTC option group and parameter group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --option-group-name msdtc-se-2016 \
    --db-parameter-group-name msdtc-sqlserver-se-13 \
```

```
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --option-group-name msdtc-se-2016 ^
    --db-parameter-group-name msdtc-sqlserver-se-13 ^
    --apply-immediately
```

Modifying the MSDTC option

After you enable the MSDTC option, you can modify its settings. For information about how to modify option settings, see Modifying an option setting.

> **Note**
>
> Some changes to MSDTC option settings require the MSDTC service to be restarted. This requirement can affect running distributed transactions.

Using transactions

Using distributed transactions

In Amazon RDS for SQL Server, you run distributed transactions in the same way as distributed transactions running on-premises:

- Using .NET framework `System.Transactions` promotable transactions, which optimizes distributed transactions by deferring their creation until they're needed.

  In this case, promotion is automatic and doesn't require you to make any intervention. If there's only one resource manager within the transaction, no promotion is performed. For more information about implicit transaction scopes, see Implementing an implicit transaction using transaction scope in the Microsoft documentation.

  Promotable transactions are supported with these .NET implementations:

- Starting with ADO.NET 2.0, `System.Data.SqlClient` supports promotable transactions with SQL Server. For more information, see System.Transactions integration with SQL Server in the Microsoft documentation.
- ODP.NET supports `System.Transactions`. A local transaction is created for the first connection opened in the `TransactionsScope` scope to Oracle Database 11g release 1 (version 11.1) and later. When a second connection is opened, this transaction is automatically promoted to a distributed transaction. For more information about distributed transaction support in ODP.NET, see Microsoft Distributed Transaction Coordinator integration in the Microsoft documentation.
- Using the `BEGIN DISTRIBUTED TRANSACTION` statement. For more information, see BEGIN DISTRIBUTED TRANSACTION (Transact-SQL) in the Microsoft documentation.

Using XA transactions

Starting from RDS for SQL Server 2017 version14.00.3223.3, you can control distributed transactions using JDBC. When you set the `Enable XA` option setting to `true` in the `MSDTC` option, RDS automatically enables JDBC transactions and grants the `SqlJDBCXAUser` role to the `guest` user. This allows executing distributed transactions through JDBC. For more information, including a code example, see Understanding XA transactions in the Microsoft documentation.

Using transaction tracing

RDS supports controlling MSDTC transaction traces and downloading them from the RDS DB instance for troubleshooting. You can control transaction tracing sessions by running the following RDS stored procedure.

```
exec msdb.dbo.rds_msdtc_transaction_tracing 'trace_action',
[@traceall='0|1'],
[@traceaborted='0|1'],
[@tracelong='0|1'];
```

The following parameter is required:

- `trace_action` – The tracing action. It can be `START`, `STOP`, or `STATUS`.

The following parameters are optional:

- `@traceall` – Set to 1 to trace all distributed transactions. The default is 0.

- `@traceaborted` – Set to 1 to trace canceled distributed transactions. The default is 0.
- `@tracelong` – Set to 1 to trace long-running distributed transactions. The default is 0.

Example of START tracing action

To start a new transaction tracing session, run the following example statement.

```
exec msdb.dbo.rds_msdtc_transaction_tracing 'START',
@traceall='0',
@traceaborted='1',
@tracelong='1';
```

> **Note**
>
> Only one transaction tracing session can be active at one time. If a new tracing session START command is issued while a tracing session is active, an error is returned and the active tracing session remains unchanged.

Example of STOP tracing action

To stop a transaction tracing session, run the following statement.

```
exec msdb.dbo.rds_msdtc_transaction_tracing 'STOP'
```

This statement stops the active transaction tracing session and saves the transaction trace data into the log directory on the RDS DB instance. The first row of the output contains the overall result, and the following lines indicate details of the operation.

The following is an example of a successful tracing session stop.

```
OK: Trace session has been successfully stopped.
Setting log file to: D:\rdsdbdata\MSDTC\Trace\dtctrace.log
Examining D:\rdsdbdata\MSDTC\Trace\msdtctr.mof for message formats,  8 found.
Searching for TMF files on path: (null)
Logfile D:\rdsdbdata\MSDTC\Trace\dtctrace.log:
 OS version    10.0.14393  (Currently running on 6.2.9200)
 Start Time    <timestamp>
 End Time      <timestamp>
```

```
 Timezone is   @tzres.dll,-932 (Bias is 0mins)
 BufferSize            16384 B
 Maximum File Size     10 MB
 Buffers  Written      Not set (Logger may not have been stopped).
 Logger Mode Settings (11000002) ( circular paged
 ProcessorCount         1
Processing completed   Buffers: 1, Events: 3, EventsLost: 0 :: Format Errors: 0,
 Unknowns: 3
Event traces dumped to d:\rdsdbdata\Log\msdtc_<timestamp>.log
```

You can use the detailed information to query the name of the generated log file. For more information about downloading log files from the RDS DB instance, see Monitoring Amazon RDS log files.

The trace session logs remain on the instance for 35 days. Any older trace session logs are automatically deleted.

Example of STATUS tracing action

To trace the status of a transaction tracing session, run the following statement.

```
exec msdb.dbo.rds_msdtc_transaction_tracing 'STATUS'
```

This statement outputs the following as separate rows of the result set.

```
OK
SessionStatus: <Started|Stopped>
TraceAll: <True|False>
TraceAborted: <True|False>
TraceLongLived: <True|False>
```

The first line indicates the overall result of the operation: `OK` or `ERROR` with details, if applicable. The subsequent lines indicate details about the tracing session status:

- `SessionStatus` can be one of the following:
  - `Started` if a tracing session is running.
  - `Stopped` if no tracing session is running.
- The tracing session flags can be `True` or `False` depending on how they were set in the `START` command.

Disabling MSDTC

To disable MSDTC, remove the `MSDTC` option from its option group.

Console

To remove the MSDTC option from its option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group with the `MSDTC` option (`msdtc-se-2016` in the previous examples).
4. Choose **Delete option**.
5. Under **Deletion options**, choose **MSDTC** for **Options to delete**.
6. Under **Apply immediately**, choose **Yes** to delete the option immediately, or **No** to delete it at the next maintenance window.
7. Choose **Delete**.

CLI

To remove the MSDTC option from its option group

- Use one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds remove-option-from-option-group \
    --option-group-name msdtc-se-2016 \
    --options MSDTC \
    --apply-immediately
```

  For Windows:

```
aws rds remove-option-from-option-group ^
    --option-group-name msdtc-se-2016 ^
    --options MSDTC ^
```

```
    --apply-immediately
```

Troubleshooting MSDTC for RDS for SQL Server

In some cases, you might have trouble establishing a connection between MSDTC running on a client computer and the MSDTC service running on an RDS for SQL Server DB instance. If so, make sure of the following:

- The inbound rules for the security group associated with the DB instance are configured correctly. For more information, see Can't connect to Amazon RDS DB instance.
- Your client computer is configured correctly.
- The MSDTC firewall rules on your client computer are enabled.

To configure the client computer

1. Open **Component Services**.

   Or, in **Server Manager**, choose **Tools**, and then choose **Component Services**.
2. Expand **Component Services**, expand **Computers**, expand **My Computer**, and then expand **Distributed Transaction Coordinator**.
3. Open the context (right-click) menu for **Local DTC** and choose **Properties**.
4. Choose the **Security** tab.
5. Choose all of the following:

   - **Network DTC Access**
   - **Allow Inbound**
   - **Allow Outbound**
6. Make sure that the correct authentication mode is chosen:

   - **Mutual Authentication Required** – The client machine is joined to the same domain as other nodes participating in distributed transaction, or there is a trust relationship configured between domains.
   - **No Authentication Required** – All other cases.
7. Choose **OK** to save your changes.
8. If prompted to restart the service, choose **Yes**.

To enable MSDTC firewall rules

1. Open Windows Firewall, then choose **Advanced settings**.

   Or, in **Server Manager**, choose **Tools**, and then choose **Windows Firewall with Advanced Security**.

   > **Note**
   >
   > Depending on your operating system, Windows Firewall might be called Windows Defender Firewall.

2. Choose **Inbound Rules** in the left pane.
3. Enable the following firewall rules, if they are not already enabled:

   - **Distributed Transaction Coordinator (RPC)**
   - **Distributed Transaction Coordinator (RPC)-EPMAP**
   - **Distributed Transaction Coordinator (TCP-In)**
4. Close Windows Firewall.

Microsoft SQL Server resource governor with RDS for SQL Server

Resource governor is a SQL Server Enterprise Edition feature that gives you precise control over your instance resources. It enables you to set specific limits on how workloads use CPU, memory, and physical I/O resources. With resource governor, you can:

- Prevent resource monopolization in multi-tenant environments by managing how different workloads share instance resources
- Deliver predictable performance by setting specific resource limits and priorities for different users and applications

You can enable resource governor on either an existing or new RDS for SQL Server DB instance.

Resource governor uses three fundamental concepts:

- **Resource pool** - A container that manages your instance physical resources (CPU, memory, and I/O). You get two built-in pools (internal and default) and you can create additional custom pools.
- **Workload group** - A container for database sessions with similar characteristics. Every workload group belongs to a resource pool. You get two built-in workload groups (internal and default) and you can create additional custom workload groups.
- **Classification** - The process that determines which workload group handles incoming sessions based on user name, application name, database name or host name.

For additional details about resource governor functionality in SQL Server, see Resource Governor in the Microsoft documentation.

Contents

- Supported versions and Regions
- Limitations and recommendations
- Enabling Microsoft SQL Server resource governor for your RDS for SQL Server instance
  - Creating the option group for RESOURCE_GOVERNOR
  - Adding the RESOURCE_GOVERNOR option to the option group
  - Associating the option group with your DB instance
- Using Microsoft SQL Server resource governor for your RDS for SQL Server instance
  - Manage resource pool

    - Create resource Pool
    - Alter resource pool
    - Drop resource pool
  - Manage workload groups
    - Create workload group
    - Alter workload group
    - Drop workload group
  - Create and register classifier function
  - Drop classifier function
  - De-register classifier function
  - Reset statistics
  - Resource governor configuration changes
  - Bind TempDB to a resource pool
  - Unbind TempDB from a resource pool
  - Cleanup resource governor
- Considerations for Multi-AZ deployment
- Considerations for read replicas
- Monitor Microsoft SQL Server resource governor using system views for your RDS for SQL Server instance
  - Resource pool runtime statistics
- Disabling Microsoft SQL Server resource governor for your RDS for SQL Server instance
- Best practices for configuring resource governor on RDS for SQL Server

Supported versions and Regions

Amazon RDS supports resource governor for the following SQL Server versions and editions in all AWS Regions where RDS for SQL Server is available:

- SQL Server 2022 Developer and Enterprise Editions
- SQL Server 2019 Enterprise Edition
- SQL Server 2017 Enterprise Edition
- SQL Server 2016 Enterprise Edition

Limitations and recommendations

The following limitations and recommendations apply to resource governor:

- Edition and service restrictions:
  - Available only in SQL Server Enterprise Edition.
  - Resource management is limited to the SQL Server Database Engine. Resource governor for Analysis Services, Integration Services, and Reporting Services are not supported.
- Configuration restrictions:
  - Must use Amazon RDS stored procedures for all configurations.
  - Native DDL statements and SQL Server Management Studio GUI configurations aren't supported.
- Resource pool parameters:
  - Pool names starting with `rds_` aren't supported.
  - Internal and default resource pool modifications aren't permitted.
  - For the user-defined resource pools the following resource pool parameters aren't supported:
    - `MIN_MEMORY_PERCENT`
    - `MIN_CPU_PERCENT`
    - `MIN_IOPS_PER_VOLUME`
    - `AFFINITY`
- Workload group parameters:
  - Workload group names starting with `rds_` aren't supported.
  - Internal workload group modification isn't permitted.
  - For the default workload group:
    - Only the `REQUEST_MAX_MEMORY_GRANT_PERCENT` parameter can be modified.
    - For the default workload group, `REQUEST_MAX_MEMORY_GRANT_PERCENT` must be between 1 and 70.
    - All other parameters are locked and can't be changed.
  - User-defined workload groups allow modification of all parameters.
- Classifier function limitations:
  - Classifier function routes connections to custom workload groups based on specified criteria (user name, database, host, or application name).

- Supports up to two user-defined workload groups with their respective routing conditions.
- Combines criterion with AND conditions within each group.
- Requires at least one routing criterion per workload group.
- Only the classification methods listed above are supported.
- Function name must start with `rg_classifier_`.
- Default group assignment if no conditions match.

Enabling Microsoft SQL Server resource governor for your RDS for SQL Server instance

Enable resource governor by adding the RESOURCE_GOVERNOR option to your RDS for SQL Server DB instance. Use the following process:

1. Create a new option group, or choose an existing option group.
2. Add the RESOURCE_GOVERNOR option to the option group.
3. Associate the option group with the DB instance.

> **Note**
>
> Enabling resource governor through an option group doesn't require a reboot.

Creating the option group for RESOURCE_GOVERNOR

To enable resource governor, create an option group or modify an option group that corresponds to the SQL Server edition and version of the DB instance that you plan to use. To complete this procedure, use the AWS Management Console or the AWS CLI.

Console

Use the following procedure to create an option group for SQL Server Enterprise Edition 2022.

To create the option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Option groups**.
3. Choose **Create group**.
4. In the **Create option group** window, do the following:

   a. For **Name**, enter a name for the option group that is unique within your AWS account, such as **`resource-governor-ee-2022`**. The name can contain only letters, digits, and hyphens.

   b. For **Description**, enter a brief description of the option group, such as **`RESOURCE_GOVERNOR option group for SQL Server EE 2022`**. The description is used for display purposes.

   c. For **Engine**, choose **sqlserver-ee**.

   d. For **Major engine version**, choose **16.00**.
5. Choose **Create**.

CLI

The following procedure creates an option group for SQL Server Enterprise Edition 2022.

To create the option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-option-group \
    --option-group-name resource-governor-ee-2022 \
    --engine-name sqlserver-ee \
    --major-engine-version 16.00 \
    --option-group-description "RESOURCE_GOVERNOR option group for SQL Server EE
 2022"
```

  For Windows:

```
aws rds create-option-group ^
    --option-group-name resource-governor-ee-2022 ^
    --engine-name sqlserver-ee ^
    --major-engine-version 16.00 ^
```

```
    --option-group-description "RESOURCE_GOVERNOR option group for SQL Server EE
 2022"
```

Adding the RESOURCE_GOVERNOR option to the option group

Next, use the AWS Management Console or the AWS CLI to add the RESOURCE_GOVERNOR option to your option group.

Console

To add the RESOURCE_GOVERNOR option

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group that you just created, **resource-governor-ee-2022** in this example.
4. Choose **Add option**.
5. Under **Option details**, choose **RESOURCE_GOVERNOR** for **Option name**.
6. Under **Scheduling**, choose whether to add the option immediately or at the next maintenance window.
7. Choose **Add option**.

CLI

To add the RESOURCE_GOVERNOR option

- Add the RESOURCE_GOVERNOR option to the option group.

  Example

  For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name resource-governor-ee-2022 \
    --options "OptionName=RESOURCE_GOVERNOR" \
    --apply-immediately
```

  For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name resource-governor-ee-2022 ^
    --options "OptionName=RESOURCE_GOVERNOR" ^
    --apply-immediately
```

Associating the option group with your DB instance

To associate the RESOURCE_GOVERNOR option group with your DB instance, use the AWS Management Console or the AWS CLI.

Console

To finish activating resource governor, associate your RESOURCE_GOVERNOR option group with a new or existing DB instance:

- For a new DB instance, associate them when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, associate them by modifying the instance. For more information, see Modifying an Amazon RDS DB instance.

CLI

You can associate the RESOURCE_GOVERNOR option group with a new or existing DB instance.

To create an instance with the RESOURCE_GOVERNOR option group

- Specify the same DB engine type and major version that you used when creating the option group.

  Example

  For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mytestsqlserverresourcegovernorinstance \
    --db-instance-class db.m5.2xlarge \
    --engine sqlserver-ee \
    --engine-version 16.00 \
    --license-model license-included \
```

```
    --allocated-storage 100 \
    --master-username admin \
    --master-user-password password \
    --storage-type gp2 \
    --option-group-name resource-governor-ee-2022
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mytestsqlserverresourcegovernorinstance ^
    --db-instance-class db.m5.2xlarge ^
    --engine sqlserver-ee ^
    --engine-version 16.00 ^
    --license-model license-included ^
    --allocated-storage 100 ^
    --master-username admin ^
    --master-user-password password ^
    --storage-type gp2 ^
    --option-group-name resource-governor-ee-2022
```

To modify an instance and associate the `RESOURCE_GOVERNOR` option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mytestinstance \
    --option-group-name resource-governor-ee-2022 \
    --apply-immediately
```

  For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mytestinstance ^
    --option-group-name resource-governor-ee-2022 ^
    --apply-immediately
```

Using Microsoft SQL Server resource governor for your RDS for SQL Server instance

After adding the resource governor option to your option group, resource governor is not yet active at the database engine level. To fully enable resource governor, you must use RDS for SQL Server stored procedures to enable it and create the necessary resource governor objects. For more information, see Connecting to your Microsoft SQL Server DB instance.

First, connect to your SQL Server database, then call the appropriate RDS for SQL Server stored procedures to complete the configuration. For instructions on connecting to your database, see Connecting to your Microsoft SQL Server DB instance.

For instructions on how to call each stored procedure, see the following topics:

Topics

- Manage resource pool
- Manage workload groups
- Create and register classifier function
- Drop classifier function
- De-register classifier function
- Reset statistics
- Resource governor configuration changes
- Bind TempDB to a resource pool
- Unbind TempDB from a resource pool
- Cleanup resource governor

Manage resource pool

Create resource Pool

Once resource governor is enabled on the option group, you can create custom resource pools using `rds_create_resource_pool`. These pools let you allocate specific percentages of CPU, memory, and IOPS to different workloads.

Usage

```
USE [msdb]
```

```
EXEC dbo.rds_create_resource_pool
    @pool_name=value,
    @MAX_CPU_PERCENT=value,
    @CAP_CPU_PERCENT=value,
    @MAX_MEMORY_PERCENT=value,
    @MAX_IOPS_PER_VOLUME=value
```

The following parameters are required:

- `@group_name` - Is the name of an existing user-defined workload group.
- `@pool_name` - Is the user-defined name for the resource pool. *`pool_name`* is alphanumeric, can be up to 128 characters, must be unique within a Database Engine instance, and must comply with the rules for database identifiers.

The following parameters are optional:

- `@MAX_CPU_PERCENT` - Specifies the maximum average CPU bandwidth that all requests in resource pool receive when there's CPU contention. *`value`* is an integer with a default setting of 100. The allowed range for *`value`* is from 1 through 100.
- `@CAP_CPU_PERCENT` - Specifies a hard cap on the CPU bandwidth that all requests in the resource pool receive. Limits the maximum CPU bandwidth level to be the same as the specified value. *`value`* is an integer with a default setting of 100. The allowed range for *`value`* is from 1 through 100.
- `@MAX_MEMORY_PERCENT` - Specifies the maximum amount of query workspace memory that requests in this resource pool can use. *`value`* is an integer with a default setting of 100. The allowed range for *`value`* is from 1 through 100.
- `@MAX_IOPS_PER_VOLUME` - Specifies the maximum I/O operations per second (IOPS) per disk volume to allow for the resource pool. The allowed range for *`value`* is from 0 through 2^31-1 (2,147,483,647). Specify 0 to remove an IOPS limit for the pool. The default is 0.

Examples

Example of creating resource pool with all default values:

```
--This creates resource pool 'SalesPool' with all default values
USE [msdb]
EXEC rds_create_resource_pool @pool_name = 'SalesPool';
```

```
--Apply changes
USE [msdb]
EXEC msdb.dbo.rds_alter_resource_governor_configuration;

--Validate configuration
select * from sys.resource_governor_resource_pools
```

Example of creating resource pool with different parameters specified:

```
--creates resource pool
USE [msdb]
EXEC dbo.rds_create_resource_pool
@pool_name='analytics',
@MAX_CPU_PERCENT = 30,
@CAP_CPU_PERCENT = 40,
@MAX_MEMORY_PERCENT = 20;

--Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;

--Validate configuration
select * from sys.resource_governor_resource_pools
```

Alter resource pool

Usage

```
USE [msdb]
EXEC dbo.rds_alter_resource_pool
    @pool_name=value,
    @MAX_CPU_PERCENT=value,
    @CAP_CPU_PERCENT=value,
    @MAX_MEMORY_PERCENT=value,
    @MAX_IOPS_PER_VOLUME=value;
```

The following parameters are required:

- `@pool_name` - Is the name of an existing user-defined resource pool. Altering default resource pool isn't allowed in Amazon RDS SQL Server.

At least one of the optional parameter must be specified:

- @MAX_CPU_PERCENT - Specifies the maximum average CPU bandwidth that all requests in resource pool receive when there's CPU contention. *value* is an integer with a default setting of 100. The allowed range for *value* is from 1 through 100.
- @CAP_CPU_PERCENT - Specifies a hard cap on the CPU bandwidth that all requests in the resource pool receive. Limits the maximum CPU bandwidth level to be the same as the specified value. *value* is an integer with a default setting of 100. The allowed range for *value* is from 1 through 100.
- @MAX_MEMORY_PERCENT - Specifies the maximum amount of query workspace memory that requests in this resource pool can use. *value* is an integer with a default setting of 100. The allowed range for *value* is from 1 through 100.
- @MAX_IOPS_PER_VOLUME - Specifies the maximum I/O operations per second (IOPS) per disk volume to allow for the resource pool. The allowed range for *value* is from 0 through 2^31-1 (2,147,483,647). Specify 0 to remove an IOPS limit for the pool. The default is 0.

Examples

```
--This alters resource pool
USE [msdb]
EXEC dbo.rds_alter_resource_pool
    @pool_name='analytics',
    @MAX_CPU_PERCENT = 10,
    @CAP_CPU_PERCENT = 20,
    @MAX_MEMORY_PERCENT = 50;

--Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;

--Validate configuration.
select * from sys.resource_governor_resource_pools
```

Drop resource pool

Usage

```
USE [msdb]
EXEC dbo.rds_drop_resource_pool
@pool_name=value;
```

The following parameter is required:

- `@pool_name` - Is the name of an existing user-defined resource pool.

> **Note**
>
> Dropping Internal or default resource pool isn't allowed in SQL Server.

Examples

```
--This drops resource pool
USE [msdb]
EXEC dbo.rds_drop_resource_pool
@pool_name='analytics'

--Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;

--Validate configuration
select * from sys.resource_governor_resource_pools
```

Manage workload groups

Workload groups, created and managed with `rds_create_workload_group` and `rds_alter_workload_group`, allow you to set importance levels, memory grants, and other parameters for groups of queries.

Create workload group

Usage

```
USE [msdb]
EXEC dbo.rds_create_workload_group
@group_name = value,
@IMPORTANCE ={ LOW | MEDIUM | HIGH },
@REQUEST_MAX_MEMORY_GRANT_PERCENT =value,
@REQUEST_MAX_CPU_TIME_SEC = value ,
@REQUEST_MEMORY_GRANT_TIMEOUT_SEC = value,
@MAX_DOP = value,
@GROUP_MAX_REQUESTS = value,
@pool_name = value
```

The following parameters are required:

- `@pool_name` - Is the name of an existing user-defined resource pool.
- `@group_name` - Is the name of an existing user-defined workload group.

The following parameters are optional:

- `@IMPORTANCE` - Specifies the relative importance of a request in the workload group. The default value is `MEDIUM`.
- `@REQUEST_MAX_MEMORY_GRANT_PERCENT` - Specifies the maximum amount of query workspace memory that a single request can take from the pool. *`value`* is a percentage of the resource pool size defined by `MAX_MEMORY_PERCENT`. Default value is 25.
- `@REQUEST_MAX_CPU_TIME_SEC` - Specifies the maximum amount of CPU time, in seconds, that a batch request can use. *`value`* must be 0 or a positive integer. The default setting for *`value`* is 0, which means unlimited.
- `@REQUEST_MEMORY_GRANT_TIMEOUT_SEC` - Specifies the maximum time, in seconds, that a query can wait for a memory grant from the query workspace memory to become available. *`value`* must be 0 or a positive integer. The default setting for *`value`*, 0, uses an internal calculation based on query cost to determine the maximum time.
- `@MAX_DOP` - Specifies the maximum degree of parallelism (`MAXDOP`) for parallel query execution. The allowed range for *`value`* is from 0 through 64. The default setting for *`value`*, 0, uses the global setting.
- `@GROUP_MAX_REQUESTS` = Specifies the maximum number of simultaneous requests that are allowed to execute in the workload group. *`value`* must be 0 or a positive integer. The default setting for *`value`* is 0, and allows unlimited requests.
- `@pool_name` = Associates the workload group with the user-defined resource pool identified by *`pool_name`*, or with the `default` resource pool. If *`pool_name`* isn't provided, the workload group is associated with the built-in `default` pool.

Examples

```
--This creates workload group named 'analytics'
USE msdb;
EXEC dbo.rds_create_workload_group
    @group_name = 'analytics',
    @IMPORTANCE = 'HIGH',
```

```
    @REQUEST_MAX_MEMORY_GRANT_PERCENT = 25,
    @REQUEST_MAX_CPU_TIME_SEC = 0,
    @REQUEST_MEMORY_GRANT_TIMEOUT_SEC = 0,
    @MAX_DOP = 0,
    @GROUP_MAX_REQUESTS = 0,
    @pool_name = 'analytics';

--Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;

--Validate configuration
select * from sys.resource_governor_workload_groups
```

Alter workload group

Usage

```
EXEC msdb.dbo.rds_alter_workload_group
    @group_name = value,
    @IMPORTANCE = 'LOW|MEDIUM|HIGH',
    @REQUEST_MAX_MEMORY_GRANT_PERCENT = value,
    @REQUEST_MAX_CPU_TIME_SEC = value,
    @REQUEST_MEMORY_GRANT_TIMEOUT_SEC = value,
    @MAX_DOP = value,
    @GROUP_MAX_REQUESTS = value,
    @pool_name = value
```

The following parameters are required:

- `@group_name` - Is the name of default or an existing user-defined workload group.

> **Note**
>
> Changing only `REQUEST_MAX_MEMORY_GRANT_PERCENT` parameter on the default workload group is supported. For default workload group the `REQUEST_MAX_MEMORY_GRANT_PERCENT` must be between 1 and 70. No other parameters can be modified in default workload group. All parameters can be modified in the user-defined workload group.

The following parameters are optional:

- @IMPORTANCE - Specifies the relative importance of a request in the workload group. The default value is MEDIUM.
- @REQUEST_MAX_MEMORY_GRANT_PERCENT - Specifies the maximum amount of query workspace memory that a single request can take from the pool. *value* is a percentage of the resource pool size defined by MAX_MEMORY_PERCENT. Default value is 25. On Amazon RDS, REQUEST_MAX_MEMORY_GRANT_PERCENT must be between 1 and 70.
- @REQUEST_MAX_CPU_TIME_SEC - Specifies the maximum amount of CPU time, in seconds, that a batch request can use. *value* must be 0 or a positive integer. The default setting for *value* is 0, which means unlimited.
- @REQUEST_MEMORY_GRANT_TIMEOUT_SEC - Specifies the maximum time, in seconds, that a query can wait for a memory grant from the query workspace memory to become available. *value* must be 0 or a positive integer. The default setting for *value*, 0, uses an internal calculation based on query cost to determine the maximum time.
- @MAX_DOP - Specifies the maximum degree of parallelism (MAXDOP) for parallel query execution. The allowed range for *value* is from 0 through 64. The default setting for *value*, 0, uses the global setting.
- @GROUP_MAX_REQUESTS - Specifies the maximum number of simultaneous requests that are allowed to execute in the workload group. *value* must be 0 or a positive integer. The default setting for *value* is 0, and allows unlimited requests.
- @pool_name - Associates the workload group with the user-defined resource pool identified by *pool_name*.

Examples

Example to Modify default workload group change REQUEST_MAX_MEMORY_GRANT_PERCENT:

```
--Modify default workload group (set memory grant cap to 10%)
USE msdb
EXEC dbo.rds_alter_workload_group
    @group_name = 'default',
    @REQUEST_MAX_MEMORY_GRANT_PERCENT=10;

--Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;

--Validate configuration
SELECT * FROM sys.resource_governor_workload_groups WHERE name='default';
```

Example to modify non-default workload group:

```
EXEC msdb.dbo.rds_alter_workload_group
    @group_name = 'analytics',
    @IMPORTANCE = 'HIGH',
    @REQUEST_MAX_MEMORY_GRANT_PERCENT = 30,
    @REQUEST_MAX_CPU_TIME_SEC = 3600,
    @REQUEST_MEMORY_GRANT_TIMEOUT_SEC = 60,
    @MAX_DOP = 4,
    @GROUP_MAX_REQUESTS = 100;

--Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;
```

Example to move a Non-Default Workload Group to another resource pool:

```
EXEC msdb.dbo.rds_alter_workload_group
@group_name = 'analytics',
@pool_name='abc'

--Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;

--Validate configuration
select * from sys.resource_governor_workload_groups
```

Drop workload group

Usage

```
EXEC msdb.dbo.rds_drop_workload_group
@group_name = value
```

The following parameters are required:

- `@group_name` - Is the name of an existing user-defined workload group.

Examples

```
--Drops a Workload Group:
EXEC msdb.dbo.rds_drop_workload_group
```

```
@group_name = 'analytics';

--Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;

--Validate configuration
select * from sys.resource_governor_workload_groups
```

Create and register classifier function

This procedure creates a resource governor classifier function in master database that routes connections to custom workload groups based on specified criteria (user name, database, host, or application name). If resource governor is enabled and a classifier function is specified in the resource governor configuration, then the function output determines the workload group used for new sessions. In the absence of a classifier function, all sessions are classified into the `default` group.

Features:

- Supports up to two workload groups with their respective routing conditions.
- Combines criterion with `AND` conditions within each group.
- Requires at least one routing criterion per workload group.
- Function name must start with `rg_classifier_`.
- Default group assignment if no conditions match.

The classifier function has the following characteristics and behaviors:

- The function is defined in the server scope (in the master database).
- The function is defined with schema binding.
- The function is evaluated for every new session, even when connection pooling is enabled.
- The function returns the workload group context for the session. The session is assigned to the workload group returned by the classifier for the lifetime of the session.
- If the function returns NULL, default, or the name of a nonexistent workload group, the session is given the default workload group context. The session is also given the default context if the function fails for any reason.
- You can create multiple classifier functions. However, SQL Server allows only one classifier function to be registered at a time.

- The classifier function can't be dropped unless its classifier status is removed using the de-register procedure (`EXEC dbo.msdb.rds_alter_resource_governor_configuration @deregister_function = 1;`) that sets the function name to NULL or another classifier function is registered using (`EXEC dbo.msdb.rds_alter_resource_governor_configuration @classifier_function = <function_name>;`)
- In the absence of a classifier function, all sessions are classified into the default group.
- You can't modify a classifier function while it is referenced in the resource governor configuration. However, you can modify the configuration to use a different classifier function. If you want to make changes to the classifier, consider creating a pair of classifier functions. For example, you might create `rg_classifier_a` and `rg_classifier_b`.

Usage

```
EXEC msdb.dbo.rds_create_classifier_function
@function_name = value,
@workload_group1 = value,
@user_name1 = value,
@db_name1 = value,
@host_name1 = value,
@app_name1 = value,
@workload_group2 = value,
@user_name2 = value,
@db_name2 = value,
@host_name2 = value,
@app_name2 = value
```

The following parameters are required:

- `@function_name` - Name of the classifier function. Must start with `rg_classifier_`
- `@workload_group1` - Name of the first workload group

The following parameters are optional:

(At least one of these criteria must be specified for group 1)

- `@user_name1` - Login name for group 1
- `@db_name1` - Database name for group 1

- @host_name1 - Host name for group 1
- @app_name1 - Application name for group 1

(If group 2 is specified, at least one criterion must be provided)

- @workload_group2 - Name of the second workload group
- @user_name2 - Login name for group 2
- @db_name2 - Database name for group 2
- @host_name2 - Host name for group 2
- @app_name2 - Application name for group 2

> **Note**
>
> System accounts, databases, applications and host are restricted.

Examples

Basic Example with One Workload Group:

```
/*Create a classifier to route all requests from 'PowerBI' app to workload group
'reporting_group'*/

EXEC msdb.dbo.rds_create_classifier_function
@function_name = 'rg_classifier_a',
@workload_group1 = 'reporting_group',
@app_name1 = 'PowerBI';

--Register the classifier
EXEC msdb.dbo.rds_alter_resource_governor_configuration
@classifier_function = 'rg_classifier_a';

-- Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration

/*Query sys.resource_governor_configuration to validate that resource governor is
 enabled and is using the classifier function we created and registered*/

use master
```

```
go
SELECT OBJECT_SCHEMA_NAME(classifier_function_id) AS classifier_schema_name,
       OBJECT_NAME(classifier_function_id) AS classifier_object_name,
       is_enabled
FROM sys.resource_governor_configuration;
```

Drop classifier function

Usage

```
USE [msdb]
EXEC dbo.rds_drop_classifier_function
@function_name = value;
```

The following parameter is required:

- `@function_name` - Is the name of an existing user-defined classifier function

Example

```
EXEC msdb.dbo.rds_drop_classifier_function
@function_name = 'rg_classifier_b';
```

De-register classifier function

Use this procedure to de-register classifier function. After the function is de-registered, new sessions are automatically assigned to the default workload group.

Usage

```
USE [msdb]
EXEC dbo.rds_alter_resource_governor_configuration
@deregister_function = 1;
```

For de-registration the following parameter is required:

- `@deregister_function` must be 1

Example

```
EXEC msdb.dbo.rds_alter_resource_governor_configuration
    @deregister_function = 1;
GO

-- Apply changes
EXEC msdb.dbo.rds_alter_resource_governor_configuration;
```

Reset statistics

Resource governor statistics are cumulative since the last server restart. If you need to collect statistics starting from a certain time, you can reset statistics using the following Amazon RDS stored procedure.

Usage

```
USE [msdb]
EXEC dbo.rds_alter_resource_governor_configuration
@reset_statistics = 1;
```

For reset stats the following parameter is required:

- `@reset_statistics` must be 1

Resource governor configuration changes

When resource governor isn't enabled, `rds_alter_resource_governor_configuration` enables resource governor. Enabling resource governor has the following results:

- The classifier function, if any, is executed for new sessions, assigning them to workload groups.
- The resource limits that are specified in resource governor configuration are honored and enforced.
- The resource limits that are specified in resource governor configuration are honored and enforced.
- Requests that existed before enabling resource governor might be affected by any configuration changes made when resource governor is enabled.
- Existing requests, before enabling resource governor, might be affected by any configuration changes made when resource governor is enabled.

- On RDS for SQL Server, `EXEC msdb.dbo.rds_alter_resource_governor_configuration` must be executed for any resource governor configuration changes to take effect.

Usage

```
USE [msdb]
EXEC dbo.rds_alter_resource_governor_configuration
```

Bind TempDB to a resource pool

You can bind tempdb memory optimized metadata to a specific resource pool using `rds_bind_tempdb_metadata_to_resource_pool` in Amazon RDS SQL Server version 2019 and above.

> **Note**
>
> Memory-optimized tempdb metadata feature must be enabled before binding tempdb metadata to resource pool. To enable this feature on Amazon RDS its a static parameter `tempdb metadata memory-optimized`.

Enable the static parameter on Amazon RDS and perform a reboot without failover for the parameter to take effect:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name test-sqlserver-ee-2022 \
    --parameters "ParameterName='tempdb metadata memory-
optimized',ParameterValue=True,ApplyMethod=pending-reboot"
```

Usage

```
USE [msdb]
EXEC dbo.rds_bind_tempdb_metadata_to_resource_pool
@pool_name=value;
```

The following parameter is required:

- `@pool_name` - Is the name of an existing user-defined resource pool.

> **Note**
>
> This change also requires sql service reboot without failover to take effect, even if Memory-optimized TempDB metadata feature is already enabled.

Unbind TempDB from a resource pool

Unbind tempdb memory optimized metadata from a resource pool.

> **Note**
>
> This change also requires sql service reboot without failover to take effect

Usage

```
USE [msdb]
EXEC dbo.rds_unbind_tempdb_metadata_from_resource_pool
```

Cleanup resource governor

This procedure is to clean up all associated objects after you have removed the resource governor option from the option group. This disables resource governor, reverts default workload group to default settings, remove custom workload groups, resource pools, and classifier functions.

Key features

- Reverts default workload group to default settings
- Disables resource governor
- Removes custom workload groups
- Removes custom resource pools
- Drops classifier functions
- Removes tempdb resource pool binding if enabled

Important

This cleanup can error out if there are active sessions on the workload group. Either wait for the active sessions to finish or terminate the active sessions as per your business requirement. It's recommended to run this during the maintenance window.
This cleanup can error out if a resource pool was bound to tempdb and reboot without failover hasn't been taken place yet. If you bound a resource pool to tempdb or unbound a resource pool from tempdb earlier, perform a reboot without failover to make the change effective. It's recommended to run this during the maintenance window.

Usage

```
USE [msdb]
EXEC dbo.rds_cleanup_resource_governor
```

Considerations for Multi-AZ deployment

RDS for SQL Server replicates resource governor to secondary instance in a Multi-AZ deployment. You can verify when modified and new resource governor last synchronized with the secondary instance.

Use the following query to check the `last_sync_time` of the replication:

```
SELECT * from msdb.dbo.rds_fn_server_object_last_sync_time();
```

In the query results, if the sync time is past the resource governor updated or creation time, then the resource governor syncs with the secondary.

To perform a manual DB failover to confirm that the resource governor replicate, wait for the `last_sync_time` to update first. Then, proceed with the Multi-AZ failover.

Considerations for read replicas

- For SQL Server replicas in the same Region as the source DB instance, use the same option group as the source. Changes to the option group propagate to replicas immediately, regardless of their maintenance windows.
- When you create a SQL Server cross-Region replica, RDS creates a dedicated option group for it.

- You can't remove an SQL Server cross-Region replica from its dedicated option group. No other DB instances can use the dedicated option group for a SQL Server cross-Region replica.
- Resource governor option is non-replicated options. You can add or remove non-replicated options from a dedicated option group.
- When you promote a SQL Server cross-Region read replica, the promoted replica behaves the same as other SQL Server DB instances, including the management of its options.

> **Note**
>
> When using Resource governor on a read replica, you must manually ensure that resource governor has been configured on your read replica using Amazon RDS stored procedures after the option is added to the option group. Resource governor configurations do not automatically replicate to the read replica. Also, the workload on read replica is typically different than the primary instance. Hence, it's recommended to apply the resource configuration on the replica based on your workload and instance type. You can run these Amazon RDS stored procedures on read replica independently to configure resource governor on read replica.

Monitor Microsoft SQL Server resource governor using system views for your RDS for SQL Server instance

Resource Governor statistics are cumulative since the last server restart. If you need to collect statistics starting from a certain time, you can reset statistics using the following Amazon RDS stored procedure:

```
EXEC msdb.dbo.rds_alter_resource_governor_configuration
@reset_statistics = 1;
```

Resource pool runtime statistics

For each resource pool, resource governor tracks CPU and memory utilization, out-of-memory events, memory grants, I/O, and other statistics. For more information, see sys.dm_resource_governor_resource_pools.

The following query returns a subset of available statistics for all resource pools:

```
SELECT rp.pool_id,
```

```
        rp.name AS resource_pool_name,
        wg.workload_group_count,
        rp.statistics_start_time,
        rp.total_cpu_usage_ms,
        rp.target_memory_kb,
        rp.used_memory_kb,
        rp.out_of_memory_count,
        rp.active_memgrant_count,
        rp.total_memgrant_count,
        rp.total_memgrant_timeout_count,
        rp.read_io_completed_total,
        rp.write_io_completed_total,
        rp.read_bytes_total,
        rp.write_bytes_total,
        rp.read_io_stall_total_ms,
        rp.write_io_stall_total_ms
FROM sys.dm_resource_governor_resource_pools AS rp
OUTER APPLY (
            SELECT COUNT(1) AS workload_group_count
            FROM sys.dm_resource_governor_workload_groups AS wg
            WHERE wg.pool_id = rp.pool_id
            ) AS wg;
```

Disabling Microsoft SQL Server resource governor for your RDS for SQL Server instance

When you disable resource governor on RDS for SQL Server, the service stops managing workload resources. Before you disable resource governor, review how this affects your database connections and configurations.

Disabling resource governor has the following results:

- The classifier function isn't executed when a new connection is opened.
- New connections are automatically classified into the default workload group.
- All existing workload group and resource pool settings are reset to their default values.
- No events are fired when limits are reached.
- Resource governor configuration changes can be made, but the changes don't take effect until resource governor is enabled.

To disable resource governor, remove the `RESOURCE_GOVERNOR` option from its option group.

Console

The following procedure removes the RESOURCE_GOVERNOR option.

To remove the RESOURCE_GOVERNOR option from its option group

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Choose the option group with the RESOURCE_GOVERNOR option (`resource-governor-ee-2022` in the previous examples).
4. Choose **Delete option**.
5. Under **Deletion options**, choose **RESOURCE_GOVERNOR** for **Options to delete**.
6. Under **Apply immediately**, choose **Yes** to delete the option immediately, or **No** to delete it during the next maintenance window.
7. Choose **Delete**.

CLI

The following procedure removes the RESOURCE_GOVERNOR option.

To remove the RESOURCE_GOVERNOR option from its option group

- Run one of the following commands.

  Example

  For Linux, macOS, or Unix:

```
aws rds remove-option-from-option-group \
    --option-group-name resource-governor-ee-2022 \
    --options RESOURCE_GOVERNOR \
    --apply-immediately
```

  For Windows:

```
aws rds remove-option-from-option-group ^
    --option-group-name resource-governor-ee-2022 ^
    --options RESOURCE_GOVERNOR ^
```

```
    --apply-immediately
```

Best practices for configuring resource governor on RDS for SQL Server

To control resource consumption, RDS for SQL Server supports Microsoft SQL Server resource governor. The following best practices help you avoid common configuration issues and optimize database performance.

1. Resource governor configuration is stored in the `master` database. We recommend that you always save a copy of resource governor configuration scripts separately.
2. The classifier function extends login processing time hence it's recommended to avoid complex logic in the classifier. An overly complex function can cause login delays or connection timeouts including Amazon RDS automation sessions. This can impact the ability of Amazon RDS automation to monitor the instance health. Hence, it's always recommended to test the classifier function in a pre-production environment before implementing in production environments.
3. Avoid setting high values (above 70) for `REQUEST_MAX_MEMORY_GRANT_PERCENT` in workload groups, as this can prevent the database instance from allocating sufficient memory for other concurrent queries, potentially resulting in memory grant timeout errors (Error 8645). Conversely, setting this value too low (less than 1) or to 0 might prevent queries that need memory workspace (like those involving sort or hash operations) from executing properly in user-defined workload groups. RDS enforces these limits by restricting values to between 1 and 70 on default workload groups.
4. For binding tempdb to resource pool, after binding memory optimized tempdb metadata to a pool, the pool might reach its maximum setting, and any queries that use `tempdb` might fail with out-of-memory errors. Under certain circumstances, the SQL Server could potentially stop if an out-of-memory error occurs. To reduce the chance of this happening, set the memory pool's `MAX_MEMORY_PERCENT` to a high value.

Common DBA tasks for Amazon RDS for Microsoft SQL Server

This section describes the Amazon RDS-specific implementations of some common DBA tasks for DB instances that are running the Microsoft SQL Server database engine. In order to deliver a managed service experience, Amazon RDS does not provide shell access to DB instances, and it restricts access to certain system procedures and tables that require advanced privileges.

> **Note**
>
> When working with a SQL Server DB instance, you can run scripts to modify a newly created database, but you cannot modify the [model] database, the database used as the model for new databases.

Topics

- Accessing the tempdb database on Microsoft SQL Server DB instances on Amazon RDS
- Analyzing your database workload on an Amazon RDS for SQL Server DB instance with Database Engine Tuning Advisor
- Changing the db_owner to the rdsa account for your Amazon RDS for SQL Server database
- Managing collations and character sets for Amazon RDS for Microsoft SQL Server
- Creating a database user for Amazon RDS for SQL Server
- Determining a recovery model for your Amazon RDS for SQL Server database
- Determining the last failover time for Amazon RDS for SQL Server
- Troubleshooting point-in-time-recovery failures due to a log sequence number gap
- Deny or allow viewing database names for Amazon RDS for SQL Server
- Disabling fast inserts during bulk loading for Amazon RDS for SQL Server
- Dropping a database in an Amazon RDS for Microsoft SQL Server DB instance
- Renaming a Amazon RDS for Microsoft SQL Server database in a Multi-AZ deployment
- Resetting the db_owner role membership for master user for Amazon RDS for SQL Server
- Restoring license-terminated DB instances for Amazon RDS for SQL Server
- Transitioning a Amazon RDS for SQL Server database from OFFLINE to ONLINE
- Using change data capture for Amazon RDS for SQL Server
- Using SQL Server Agent for Amazon RDS

- Working with Amazon RDS for Microsoft SQL Server logs
- Working with trace and dump files for Amazon RDS for SQL Server

Accessing the tempdb database on Microsoft SQL Server DB instances on Amazon RDS

You can access the tempdb database on your Microsoft SQL Server DB instances on Amazon RDS. You can run code on tempdb by using Transact-SQL through Microsoft SQL Server Management Studio (SSMS), or any other standard SQL client application. For more information about connecting to your DB instance, see Connecting to your Microsoft SQL Server DB instance.

The master user for your DB instance is granted CONTROL access to tempdb so that this user can modify the tempdb database options. The master user isn't the database owner of the tempdb database. If necessary, the master user can grant CONTROL access to other users so that they can also modify the tempdb database options.

> **Note**
>
> You can't run Database Console Commands (DBCC) on the tempdb database.

Modifying tempdb database options

You can modify the database options on the tempdb database on your Amazon RDS DB instances. For more information about which options can be modified, see tempdb database in the Microsoft documentation.

Database options such as the maximum file size options are persistent after you restart your DB instance. You can modify the database options to optimize performance when importing data, and to prevent running out of storage.

Optimizing performance when importing data

To optimize performance when importing large amounts of data into your DB instance, set the SIZE and FILEGROWTH properties of the tempdb database to large numbers. For more information about how to optimize tempdb, see Optimizing tempdb performance in the Microsoft documentation.

The following example demonstrates setting the size to 100 GB and file growth to 10 percent.

```
alter database[tempdb] modify file (NAME = N'templog', SIZE=100GB, FILEGROWTH = 10%)
```

Preventing storage problems

To prevent the tempdb database from using all available disk space, set the MAXSIZE property. The following example demonstrates setting the property to 2048 MB.

```
alter database [tempdb] modify file (NAME = N'templog', MAXSIZE = 2048MB)
```

Shrinking the tempdb database

There are two ways to shrink the tempdb database on your Amazon RDS DB instance. You can use the rds_shrink_tempdbfile procedure, or you can set the SIZE property,

Using the rds_shrink_tempdbfile procedure

You can use the Amazon RDS procedure msdb.dbo.rds_shrink_tempdbfile to shrink the tempdb database. You can only call rds_shrink_tempdbfile if you have CONTROL access to tempdb. When you call rds_shrink_tempdbfile, there is no downtime for your DB instance.

The rds_shrink_tempdbfile procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
@temp_filename	SYSNAME	—	required	The logical name of the file to shrink.
@target_size	int	null	optional	The new size for the file, in megabytes.

The following example gets the names of the files for the tempdb database.

```
use tempdb;
GO

select name, * from sys.sysfiles;
GO
```

The following example shrinks a tempdb database file named test_file, and requests a new size of 10 megabytes:

```
exec msdb.dbo.rds_shrink_tempdbfile @temp_filename = N'test_file', @target_size = 10;
```

Setting the SIZE property

You can also shrink the tempdb database by setting the SIZE property and then restarting your DB instance. For more information about restarting your DB instance, see Rebooting a DB instance.

The following example demonstrates setting the SIZE property to 1024 MB.

```
alter database [tempdb] modify file (NAME = N'templog', SIZE = 1024MB)
```

TempDB configuration for Multi-AZ deployments

If your RDS for SQL Server DB instance is in a Multi-AZ Deployment using Database Mirroring (DBM) or Always On Availability Groups (AGs), keep in mind the following considerations for using the tempdb database.

You can't replicate tempdb data from your primary DB instance to your secondary DB instance. When you fail over to a secondary DB instance, tempdb on that secondary DB instance will be empty.

You can synchronize the configuration of the tempdb database options, including its file sizing and autogrowth settings, from your primary DB instance to your secondary DB instance. Synchronizing the tempDB configuration is supported on all RDS for SQL Server versions. You can turn on automatic synchronization of the tempdb configuration by using the following stored procedure:

```
EXECUTE msdb.dbo.rds_set_system_database_sync_objects @object_types = 'TempDbFile';
```

> **Important**
>
> Before using the rds_set_system_database_sync_objects stored procedure, ensure you've set your preferred tempdb configuration on your primary DB instance, rather than on your secondary DB instance. If you made the configuration change on your secondary DB instance, your preferred tempdb configuration could be deleted when you turn on automatic synchronization.

You can use the following function to confirm whether automatic synchronization of the tempdb configuration is turned on:

```
SELECT * from msdb.dbo.rds_fn_get_system_database_sync_objects();
```

When automatic synchronization of the tempdb configuration is turned on, there will be a return value for the object_class field. When it's turned off, no value is returned.

You can use the following function to find the last time objects were synchronized, in UTC time:

```
SELECT * from msdb.dbo.rds_fn_server_object_last_sync_time();
```

For example, if you modified the tempdb configuration at 01:00 and then run the rds_fn_server_object_last_sync_time function, the value returned for last_sync_time should be after 01:00, indicating that an automatic synchronization occurred.

If you are also using SQL Server Agent job replication, you can enable replication for both SQL Agent jobs and the tempdb configuration by providing them in the @object_type parameter:

```
EXECUTE msdb.dbo.rds_set_system_database_sync_objects @object_types =
 'SQLAgentJob,TempDbFile';
```

For more information on SQL Server Agent job replication, see Turning on SQL Server Agent job replication.

As an alternative to using the rds_set_system_database_sync_objects stored procedure to ensure that tempdb configuration changes are automatically synchronized, you can use one of the following manual methods:

> **Note**
>
> We recommend turning on automatic synchronization of the tempdb configuration by using the rds_set_system_database_sync_objects stored procedure. Using automatic synchronization prevents the need to perform these manual tasks each time you change your tempdb configuration.

- First modify your DB instance and turn Multi-AZ off, then modify tempdb, and finally turn Multi-AZ back on. This method doesn't involve any downtime.

  For more information, see Modifying an Amazon RDS DB instance.

- First modify `tempdb` in the original primary instance, then fail over manually, and finally modify `tempdb` in the new primary instance. This method involves downtime.

  For more information, see Rebooting a DB instance.

Analyzing your database workload on an Amazon RDS for SQL Server DB instance with Database Engine Tuning Advisor

Database Engine Tuning Advisor is a client application provided by Microsoft that analyzes database workload and recommends an optimal set of indexes for your Microsoft SQL Server databases based on the kinds of queries you run. Like SQL Server Management Studio, you run Tuning Advisor from a client computer that connects to your Amazon RDS DB instance that is running SQL Server. The client computer can be a local computer that you run on premises within your own network or it can be an Amazon EC2 Windows instance that is running in the same region as your Amazon RDS DB instance.

This section shows how to capture a workload for Tuning Advisor to analyze. This is the preferred process for capturing a workload because Amazon RDS restricts host access to the SQL Server instance. For more information, see Database Engine Tuning Advisor in the Microsoft documentation.

To use Tuning Advisor, you must provide what is called a workload to the advisor. A workload is a set of Transact-SQL statements that run against a database or databases that you want to tune. Database Engine Tuning Advisor uses trace files, trace tables, Transact-SQL scripts, or XML files as workload input when tuning databases. When working with Amazon RDS, a workload can be a file on a client computer or a database table on an Amazon RDS for SQL Server DB accessible to your client computer. The file or the table must contain queries against the databases you want to tune in a format suitable for replay.

For Tuning Advisor to be most effective, a workload should be as realistic as possible. You can generate a workload file or table by performing a trace against your DB instance. While a trace is running, you can either simulate a load on your DB instance or run your applications with a normal load.

There are two types of traces: client-side and server-side. A client-side trace is easier to set up and you can watch trace events being captured in real-time in SQL Server Profiler. A server-side trace is more complex to set up and requires some Transact-SQL scripting. In addition, because the trace is written to a file on the Amazon RDS DB instance, storage space is consumed by the trace. It is

important to track of how much storage space a running server-side trace uses because the DB instance could enter a storage-full state and would no longer be available if it runs out of storage space.

For a client-side trace, when a sufficient amount of trace data has been captured in the SQL Server Profiler, you can then generate the workload file by saving the trace to either a file on your local computer or in a database table on a DB instance that is available to your client computer. The main disadvantage of using a client-side trace is that the trace may not capture all queries when under heavy loads. This could weaken the effectiveness of the analysis performed by the Database Engine Tuning Advisor. If you need to run a trace under heavy loads and you want to ensure that it captures every query during a trace session, you should use a server-side trace.

For a server-side trace, you must get the trace files on the DB instance into a suitable workload file or you can save the trace to a table on the DB instance after the trace completes. You can use the SQL Server Profiler to save the trace to a file on your local computer or have the Tuning Advisor read from the trace table on the DB instance.

Running a client-side trace on a SQL Server DB instance

To run a client-side trace on a SQL Server DB instance

1. Start SQL Server Profiler. It is installed in the Performance Tools folder of your SQL Server instance folder. You must load or define a trace definition template to start a client-side trace.
2. In the SQL Server Profiler File menu, choose **New Trace**. In the **Connect to Server** dialog box, enter the DB instance endpoint, port, master user name, and password of the database you would like to run a trace on.
3. In the **Trace Properties** dialog box, enter a trace name and choose a trace definition template. A default template, TSQL_Replay, ships with the application. You can edit this template to define your trace. Edit events and event information under the **Events Selection** tab of the **Trace Properties** dialog box.

   For more information about trace definition templates and using the SQL Server Profiler to specify a client-side trace, see Database Engine Tuning Advisor in the Microsoft documentation.
4. Start the client-side trace and watch SQL queries in real-time as they run against your DB instance.
5. Select **Stop Trace** from the **File** menu when you have completed the trace. Save the results as a file or as a trace table on you DB instance.

Running a server-side trace on a SQL Server DB instance

Writing scripts to create a server-side trace can be complex and is beyond the scope of this document. This section contains sample scripts that you can use as examples. As with a client-side trace, the goal is to create a workload file or trace table that you can open using the Database Engine Tuning Advisor.

The following is an abridged example script that starts a server-side trace and captures details to a workload file. The trace initially saves to the file RDSTrace.trc in the D:\RDSDBDATA\Log directory and rolls-over every 100 MB so subsequent trace files are named RDSTrace_1.trc, RDSTrace_2.trc, etc.

```
DECLARE @file_name NVARCHAR(245) = 'D:\RDSDBDATA\Log\RDSTrace';
DECLARE @max_file_size BIGINT = 100;
DECLARE @on BIT = 1
DECLARE @rc INT
DECLARE @traceid INT

EXEC @rc = sp_trace_create @traceid OUTPUT, 2, @file_name, @max_file_size
IF (@rc = 0) BEGIN
   EXEC sp_trace_setevent @traceid, 10, 1, @on
   EXEC sp_trace_setevent @traceid, 10, 2, @on
   EXEC sp_trace_setevent @traceid, 10, 3, @on
 . . .
   EXEC sp_trace_setfilter @traceid, 10, 0, 7, N'SQL Profiler'
   EXEC sp_trace_setstatus @traceid, 1
   END
```

The following example is a script that stops a trace. Note that a trace created by the previous script continues to run until you explicitly stop the trace or the process runs out of disk space.

```
DECLARE @traceid INT
SELECT @traceid = traceid FROM ::fn_trace_getinfo(default)
WHERE property = 5 AND value = 1 AND traceid <> 1

IF @traceid IS NOT NULL BEGIN
   EXEC sp_trace_setstatus @traceid, 0
   EXEC sp_trace_setstatus @traceid, 2
END
```

You can save server-side trace results to a database table and use the database table as the workload for the Tuning Advisor by using the fn_trace_gettable function. The following commands load the results of all files named RDSTrace.trc in the D:\rdsdbdata\Log directory, including all rollover files like RDSTrace_1.trc, into a table named RDSTrace in the current database.

```
SELECT * INTO RDSTrace
FROM fn_trace_gettable('D:\rdsdbdata\Log\RDSTrace.trc', default);
```

To save a specific rollover file to a table, for example the RDSTrace_1.trc file, specify the name of the rollover file and substitute 1 instead of default as the last parameter to fn_trace_gettable.

```
SELECT * INTO RDSTrace_1
FROM fn_trace_gettable('D:\rdsdbdata\Log\RDSTrace_1.trc', 1);
```

Running Tuning Advisor with a trace

Once you create a trace, either as a local file or as a database table, you can then run Tuning Advisor against your DB instance. Using Tuning Advisor with Amazon RDS is the same process as when working with a standalone, remote SQL Server instance. You can either use the Tuning Advisor UI on your client machine or use the dta.exe utility from the command line. In both cases, you must connect to the Amazon RDS DB instance using the endpoint for the DB instance and provide your master user name and master user password when using Tuning Advisor.

The following code example demonstrates using the dta.exe command line utility against an Amazon RDS DB instance with an endpoint of **`dta.cnazcmklsdei.us-east-1.rds.amazonaws.com`**. The example includes the master user name **`admin`** and the master user password **`test`**, the example database to tune is named machine named **`C:\RDSTrace.trc`**. The example command line code also specifies a trace session named **`RDSTrace1`** and specifies output files to the local machine named **`RDSTrace.sql`** for the SQL output script, **`RDSTrace.txt`** for a result file, and **`RDSTrace.xml`** for an XML file of the analysis. There is also an error table specified on the RDSDTA database named **`RDSTraceErrors`**.

```
dta -S dta.cnazcmklsdei.us-east-1.rds.amazonaws.com -U admin -P test -D RDSDTA -
if C:\RDSTrace.trc -s RDSTrace1 -of C:\ RDSTrace.sql -or C:\ RDSTrace.txt -ox C:\
 RDSTrace.xml -e RDSDTA.dbo.RDSTraceErrors
```

Here is the same example command line code except the input workload is a table on the remote Amazon RDS instance named **`RDSTrace`** which is on the **`RDSDTA`** database.

```
dta -S dta.cnazcmklsdei.us-east-1.rds.amazonaws.com -U admin -P test -D RDSDTA -it
 RDSDTA.dbo.RDSTrace -s RDSTrace1 -of C:\ RDSTrace.sql -or C:\ RDSTrace.txt -ox C:\
 RDSTrace.xml -e RDSDTA.dbo.RDSTraceErrors
```

For a full list of dta utility command-line parameters, see dta Utility in the Microsoft documentation.

Changing the db_owner to the rdsa account for your Amazon RDS for SQL Server database

When you create or restore a database in an RDS for SQL Server DB instance, Amazon RDS sets the owner of the database to `rdsa`. If you have a Multi-AZ deployment using SQL Server Database Mirroring (DBM) or Always On Availability Groups (AGs), Amazon RDS sets the owner of the database on the secondary DB instance to `NT AUTHORITY\SYSTEM`. The owner of the secondary database can't be changed until the secondary DB instance is promoted to the primary role. In most cases, setting the owner of the database to `NT AUTHORITY\SYSTEM` isn't problematic when executing queries, however, can throw errors when executing system stored procedures such as `sys.sp_updatestats` that require elevated permissions to execute.

You can use the following query to identify the owner of the databases owned by `NT AUTHORITY\SYSTEM`:

```
SELECT name FROM sys.databases WHERE SUSER_SNAME(owner_sid) = 'NT AUTHORITY\SYSTEM';
```

You can use the Amazon RDS stored procedure `rds_changedbowner_to_rdsa` to change the owner of the database to `rdsa`. The following databases are not allowed to be used with `rds_changedbowner_to_rdsa`: `master, model, msdb, rdsadmin, rdsadmin_ReportServer, rdsadmin_ReportServerTempDB, SSISDB`.

To change the owner of the database to `rdsa`, call the `rds_changedbowner_to_rdsa` stored procedure and provide the name of the database.

Example usage:

```
exec msdb.dbo.rds_changedbowner_to_rdsa 'TestDB1';
```

The following parameter is required:

- @db_name – The name of the database to change the owner of the database to `rdsa`.

Important

You can't use `rds_changedbowner_to_rdsa` to change ownership of a database to a login other than `rdsa`. For example, you can't change the ownership to the login with which you created the database. To restore lost membership in the `db_owner` role for your master user when no other database user can be used to grant the membership, reset the master user password to obtain membership in the `db_owner` role. For more information, see Resetting the db_owner role membership for master user for Amazon RDS for SQL Server.

Managing collations and character sets for Amazon RDS for Microsoft SQL Server

This topic provide guidance on how to manage collations and character sets for Microsoft SQL Server in Amazon RDS. It explains how to configure collations during database creation and modify them later, ensuring proper handling of text data based on language and locale requirements. Additionally, it covers best practices for maintaining compatibility and performance in SQL Server environments in Amazon RDS.

SQL Server supports collations at multiple levels. You set the default server collation when you create the DB instance. You can override the collation in the database, table, or column level.

Topics

- Server-level collation for Microsoft SQL Server
- Database-level collation for Microsoft SQL Server

Server-level collation for Microsoft SQL Server

When you create a Microsoft SQL Server DB instance, you can set the server collation that you want to use. If you don't choose a different collation, the server-level collation defaults to SQL_Latin1_General_CP1_CI_AS. The server collation is applied by default to all databases and database objects.

Note

You can't change the collation when you restore from a DB snapshot.

Currently, Amazon RDS supports the following server collations:

Collation	Description
Arabic_CI_AS	Arabic, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Chinese_PRC_BIN2	Chinese-PRC, binary code point sort order
Chinese_PRC_CI_AS	Chinese-PRC, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Chinese_Taiwan_Stroke_CI_AS	Chinese-Taiwan-Stroke, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Danish_Norwegian_CI_AS	Danish-Norwegian, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Danish_Norwegian_CI_AS_KS	Danish-Norwegian, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive
Danish_Norwegian_CI_AS_KS_WS	Danish-Norwegian, case-insensitive, accent-sensitive, kanatype-sensitive, width-sensitive
Danish_Norwegian_CI_AS_WS	Danish-Norwegian, case-insensitive, accent-sensitive, kanatype-insensitive, width-sensitive
Danish_Norwegian_CS_AI	Danish-Norwegian, case-sensitive, accent-insensitive, kanatype-insensitive, width-insensitive
Danish_Norwegian_CS_AI_KS	Danish-Norwegian, case-sensitive, accent-insensitive, kanatype-sensitive, width-insensitive
Finnish_Swedish_100_BIN	Finnish-Swedish-100, binary sort
Finnish_Swedish_100_BIN2	Finnish-Swedish-100, binary code point comparison sort

Collation	Description
Finnish_Swedish_100_CI_AI	Finnish-Swedish-100, case-insensitive, accent-insensitive, kanatype-insensitive, width-ins ensitive
Finnish_Swedish_100_CI_AS	Finnish-Swedish-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive
Finnish_Swedish_CI_AS	Finnish, Swedish, and Swedish (Finland), case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
French_CI_AS	French, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Greek_CI_AS	Greek, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Greek_CS_AS	Greek, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Hebrew_BIN	Hebrew, binary sort
Hebrew_CI_AS	Hebrew, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_BIN	Japanese, binary sort
Japanese_CI_AS	Japanese, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Japanese_CS_AS	Japanese, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive

Collation	Description
Japanese_XJIS_140_CI_AS	Japanese, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector insensitive
Japanese_XJIS_140_CI_AS_KS_VSS	Japanese, case-insensitive, accent-sensitive, kanatype-sensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CI_AS_VSS	Japanese, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive, supplementary characters, variation selector sensitive
Japanese_XJIS_140_CS_AS_KS_WS	Japanese, case-sensitive, accent-sensitive, kanatype-sensitive, width-sensitive, supplementary characters, variation selector insensitive
Korean_Wansung_CI_AS	Korean-Wansung, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Latin1_General_100_BIN	Latin1-General-100, binary sort
Latin1_General_100_BIN2	Latin1-General-100, binary code point sort order
Latin1_General_100_BIN2_UTF8	Latin1-General-100, binary code point sort order, UTF-8 encoded
Latin1_General_100_CI_AS	Latin1-General-100, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive

Collation	Description
Latin1_General_100_CI_AS_SC_UTF8	Latin1-General-100, case-insensitive, accent-sensitive, supplementary characters, UTF-8 encoded
Latin1_General_BIN	Latin1-General, binary sort
Latin1_General_BIN2	Latin1-General, binary code point sort order
Latin1_General_CI_AI	Latin1-General, case-insensitive, accent-in sensitive, kanatype-insensitive, width-ins ensitive
Latin1_General_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Latin1_General_CI_AS_KS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-sensitive, width-insensitive
Latin1_General_CS_AS	Latin1-General, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Modern_Spanish_CI_AS	Modern-Spanish, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive
Polish_CI_AS	Polish, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
SQL_1xCompat_CP850_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 49 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP1_CI_AI	Latin1-General, case-insensitive, accent-in sensitive, kanatype-insensitive, width-ins ensitive for Unicode Data, SQL Server Sort Order 54 on Code Page 1252 for non-Unicode Data

Collation	Description
SQL_Latin1_General_CP1_CI_AS (default)	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 52 on Code Page 1252 for non-Unicode Data
SQL_Latin1_General_CP1_CS_AS	Latin1-General, case-sensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 51 on Code Page 1252 for non-Unicode Data
SQL_Latin1_General_CP437_CI_AI	Latin1-General, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 34 on Code Page 437 for non-Unicode Data
SQL_Latin1_General_CP850_BIN	Latin1-General, binary sort order for Unicode Data, SQL Server Sort Order 40 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP850_BIN2	Latin1-General, binary code point sort order for Unicode Data, SQL Server Sort Order 40 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP850_CI_AI	Latin1-General, case-insensitive, accent-insensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 44 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP850_CI_AS	Latin1-General, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 42 on Code Page 850 for non-Unicode Data

Collation	Description
SQL_Latin1_General_Pref_CP850_CI_AS	Latin1-General-Pref, case-insensitive, accent-sensitive, kanatype-insensitive, width-ins ensitive for Unicode Data, SQL Server Sort Order 183 on Code Page 850 for non-Unicode Data
SQL_Latin1_General_CP1256_CI_AS	Latin1-General, case-insensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 146 on Code Page 1256 for non-Unicode Data
SQL_Latin1_General_CP1255_CS_AS	Latin1-General, case-sensitive, accent-se nsitive, kanatype-insensitive, width-insensitive for Unicode Data, SQL Server Sort Order 137 on Code Page 1255 for non-Unicode Data
Thai_CI_AS	Thai, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive
Turkish_CI_AS	Turkish, case-insensitive, accent-sensitive, kanatype-insensitive, width-insensitive

You can also retrieve the list of supported collations programmatically using the AWS CLI:

```
aws rds describe-db-engine-versions --engine sqlserver-ee --list-supported-character-
sets --query 'DBEngineVersions[].SupportedCharacterSets[].CharacterSetName' | sort -u
```

To choose the collation:

- If you're using the Amazon RDS console, when creating a new DB instance choose **Additional configuration**, then enter the collation in the **Collation** field. For more information, see Creating an Amazon RDS DB instance.
- If you're using the AWS CLI, use the `--character-set-name` option with the `create-db-instance` command. For more information, see create-db-instance.

- If you're using the Amazon RDS API, use the `CharacterSetName` parameter with the `CreateDBInstance` operation. For more information, see CreateDBInstance.

Database-level collation for Microsoft SQL Server

You can change the default collation at the database, table, or column level by overriding the collation when creating a new database or database object. For example, if your default server collation is SQL_Latin1_General_CP1_CI_AS, you can change it to Mohawk_100_CI_AS for Mohawk collation support. Even arguments in a query can be type-cast to use a different collation if necessary.

For example, the following query would change the default collation for the AccountName column to Mohawk_100_CI_AS

```
CREATE TABLE [dbo].[Account]
 (
     [AccountID] [nvarchar](10) NOT NULL,
     [AccountName] [nvarchar](100) COLLATE Mohawk_100_CI_AS NOT NULL
 ) ON [PRIMARY];
```

The Microsoft SQL Server DB engine supports Unicode by the built-in NCHAR, NVARCHAR, and NTEXT data types. For example, if you need CJK support, use these Unicode data types for character storage and override the default server collation when creating your databases and tables. Here are several links from Microsoft covering collation and Unicode support for SQL Server:

- Working with collations
- Collation and international terminology
- Using SQL Server collations
- International considerations for databases and database engine applications

Creating a database user for Amazon RDS for SQL Server

You can create a database user for your Amazon RDS for Microsoft SQL Server DB instance by running a T-SQL script like the following example. Use an application such as SQL Server Management Suite (SSMS). You log into the DB instance as the master user that was created when you created the DB instance.

```
--Initially set context to master database
USE [master];
GO
--Create a server-level login named theirname with password theirpassword
CREATE LOGIN [theirname] WITH PASSWORD = 'theirpassword';
GO
--Set context to msdb database
USE [msdb];
GO
--Create a database user named theirname and link it to server-level login theirname
CREATE USER [theirname] FOR LOGIN [theirname];
GO
```

For an example of adding a database user to a role, see Adding a user to the SQLAgentUser role.

Note

If you get permission errors when adding a user, you can restore privileges by modifying the DB instance master user password. For more information, see Resetting the db_owner role membership for master user for Amazon RDS for SQL Server.
It is not a best practice to clone master user permissions in your applications. For more information, see How to clone master user permissions in Amazon RDS for SQL Server.

Determining a recovery model for your Amazon RDS for SQL Server database

In Amazon RDS, the recovery model, retention period, and database status are linked.

It's important to understand the consequences before making a change to one of these settings. Each setting can affect the others. For example:

- If you change a database's recovery model to SIMPLE or BULK_LOGGED while backup retention is enabled, Amazon RDS resets the recovery model to FULL within five minutes. This also results in RDS taking a snapshot of the DB instance.
- If you set backup retention to `0` days, RDS sets the recovery mode to SIMPLE.
- If you change a database's recovery model from SIMPLE to any other option while backup retention is set to `0` days, RDS resets the recovery model to SIMPLE.

Important

Never change the recovery model on Multi-AZ instances, even if it seems you can do so—for example, by using ALTER DATABASE. Backup retention, and therefore FULL recovery mode, is required for Multi-AZ. If you alter the recovery model, RDS immediately changes it back to FULL.

This automatic reset forces RDS to completely rebuild the mirror. During this rebuild, the availability of the database is degraded for about 30-90 minutes until the mirror is ready for failover. The DB instance also experiences performance degradation in the same way it does during a conversion from Single-AZ to Multi-AZ. How long performance is degraded depends on the database storage size—the bigger the stored database, the longer the degradation.

For more information on SQL Server recovery models, see Recovery models (SQL Server) in the Microsoft documentation.

Determining the last failover time for Amazon RDS for SQL Server

To determine the last failover time, use the following stored procedure:

```
execute msdb.dbo.rds_failover_time;
```

This procedure returns the following information.

Output parameter	Description
errorlog_available_from	Shows the time from when error logs are available in the log directory.
recent_failover_time	Shows the last failover time if it's available from the error logs. Otherwise it shows `null`.

> **Note**
>
> The stored procedure searches all of the available SQL Server error logs in the log directory to retrieve the most recent failover time. If the failover messages have been overwritten by SQL Server, then the procedure doesn't retrieve the failover time.

Example of no recent failover

This example shows the output when there is no recent failover in the error logs. No failover has happened since 2020-04-29 23:59:00.01.

errorlog_available_from	recent_failover_time
2020-04-29 23:59:00.0100000	null

Example of recent failover

This example shows the output when there is a failover in the error logs. The most recent failover was at 2020-05-05 18:57:51.89.

errorlog_available_from	recent_failover_time
2020-04-29 23:59:00.0100000	2020-05-05 18:57:51.8900000

Troubleshooting point-in-time-recovery failures due to a log sequence number gap

When attempting point-in-time-recovery (PITR) in RDS for SQL Server, you might encounter failures due to gaps in log sequence numbers (LSNs). These gaps prevent RDS from restoring your database to the requested time and RDS places your restoring instance in `incompatible-restore` state.

Common causes for this issue are:

- Manual changes to the database recovery model.
- Automatic recovery model changes by RDS due to insufficient resources for completing transaction log backups.

To identify LSN gaps in your database, run this query:

```
SELECT * FROM msdb.dbo.rds_fn_list_tlog_backup_metadata(database_name)
ORDER BY backup_file_time_utc desc;
```

If you discover an LSN gap, you can:

- Choose a restore point before the LSN gap.
- Wait and restore to a point after the next instance backup completes.

To prevent this issue, we recommend you don't manually change the recovery model of your RDS for SQL Server databases, as it interrupts instance durability. We also recommend you choose an instance type with sufficient resources for your workload to ensure regular transaction log backups.

For more information about transaction log management, see SQL Server transaction log architecture and management guide in the Microsoft SQL Server documentation.

Deny or allow viewing database names for Amazon RDS for SQL Server

The master user cannot set `DENY VIEW ANY DATABASE TO LOGIN`
to hide databases from a user.
To change this permission, use the following stored procedure instead:

- Denying database view access to *LOGIN*:

```
EXEC msdb.dbo.rds_manage_view_db_permission @permission='DENY',
 @server_principal='LOGIN'

go
```

- Allowing database view access to *LOGIN*:

```
EXEC msdb.dbo.rds_manage_view_db_permission @permission='GRANT',
 @server_principal='LOGIN'
```

```
go
```

Consider the following when using this stored procedure:

- Database names are hidden from the SSMS and internal DMV (dynamic management views). However, database names are still visible from audit, logs and metadata tables. These are secured `VIEW ANY DATABASE` server permissions. For more information, see DENY Server Permissions.
- Once the permission is reverted to `GRANT` (allowed), the *`LOGIN`* can view all databases.
- If you delete and recreate *`LOGIN`*, the view permission related to the LOGIN is reset to `ALLOW`.
- For Multi-AZ instances, set the `DENY` or `GRANT` permission only for the *`LOGIN`* on the primary host. The changes are propagated to the secondary host automatically.
- This permission only changes whether a login can view the database names. However, access to databases and objects within are managed separately.

Disabling fast inserts during bulk loading for Amazon RDS for SQL Server

Starting with SQL Server 2016, fast inserts are enabled by default. Fast inserts leverage the minimal logging that occurs while the database is in the simple or bulk logged recovery model to optimize insert performance. With fast inserts, each bulk load batch acquires new extents, bypassing the allocation lookup for existing extents with available free space to optimize insert performance.

However, with fast inserts bulk loads with small batch sizes can lead to increased unused space consumed by objects. If increasing batch size isn't feasible, enabling trace flag 692 can help reduce unused reserved space, but at the expense of performance. Enabling this trace flag disables fast inserts while bulk loading data into heap or clustered indexes.

You enable trace flag 692 as a startup parameter using DB parameter groups. For more information, see Parameter groups for Amazon RDS.

Trace flag 692 is supported for Amazon RDS on SQL Server 2016 and later. For more information on trace flags, see DBCC TRACEON - trace flags in the Microsoft documentation.

Dropping a database in an Amazon RDS for Microsoft SQL Server DB instance

You can drop a database on an Amazon RDS DB instance running Microsoft SQL Server in a Single-AZ or Multi-AZ deployment. To drop the database, use the following command:

```
--replace your-database-name with the name of the database you want to drop
EXECUTE msdb.dbo.rds_drop_database  N'your-database-name'
```

> **Note**
>
> Use straight single quotes in the command. Smart quotes will cause an error.

After you use this procedure to drop the database, Amazon RDS drops all existing connections to the database and removes the database's backup history.

To grant backup and restore allowance to other users, follow this procedure:

```
USE master
GO
CREATE LOGIN user1 WITH PASSWORD=N'changeThis', DEFAULT_DATABASE=master,
 CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO
USE msdb
GO
CREATE USER user1 FOR LOGIN user1
GO
use msdb
GO
GRANT EXECUTE ON msdb.dbo.rds_backup_database TO user1
GO
GRANT EXECUTE ON msdb.dbo.rds_restore_database TO user1
GO
```

Renaming a Amazon RDS for Microsoft SQL Server database in a Multi-AZ deployment

To rename a Microsoft SQL Server database instance that uses Multi-AZ, use the following procedure:

1. First, turn off Multi-AZ for the DB instance.
2. Rename the database by running `rdsadmin.dbo.rds_modify_db_name`.
3. Then, turn on Multi-AZ Mirroring or Always On Availability Groups for the DB instance, to return it to its original state.

For more information, see Adding Multi-AZ to a Microsoft SQL Server DB instance.

> **Note**
>
> If your instance doesn't use Multi-AZ, you don't need to change any settings before or after running `rdsadmin.dbo.rds_modify_db_name`.
> You can't rename a database on a read replica source instance.

Example: In the following example, the `rdsadmin.dbo.rds_modify_db_name` stored procedure renames a database from **MOO** to **ZAR**. This is similar to running the statement `DDL ALTER DATABASE [MOO] MODIFY NAME = [ZAR]`.

```
EXEC rdsadmin.dbo.rds_modify_db_name N'MOO', N'ZAR'
GO
```

Resetting the db_owner role membership for master user for Amazon RDS for SQL Server

If you lock your master user out of the `db_owner` role membership on your RDS for SQL Server database and no other database user can grant the membership, you can restore lost membership by modifying the DB instance master user password.

By changing the DB instance master user password, RDS grants the `db_owner` membership to the databases in the DB instance that might have been accidentally revoked. You can change the DB instance password by using the Amazon RDS console, the AWS CLI command modify-db-instance, or by using the ModifyDBInstance API operation. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

Restoring license-terminated DB instances for Amazon RDS for SQL Server

Microsoft has requested that some Amazon RDS customers who did not report their Microsoft License Mobility information terminate their DB instance. Amazon RDS takes snapshots of these DB instances, and you can restore from the snapshot to a new DB instance that has the License Included model.

You can restore from a snapshot of Standard Edition to either Standard Edition or Enterprise Edition.

You can restore from a snapshot of Enterprise Edition to either Standard Edition or Enterprise Edition.

To restore from a SQL Server snapshot after Amazon RDS has created a final snapshot of your instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the snapshot of your SQL Server DB instance. Amazon RDS creates a final snapshot of your DB instance. The name of the terminated instance snapshot is in the format *instance_name*`-final-snapshot`. For example, if your DB instance name is **`mytest.cdxgahslksma.us-east-1.rds.com`**, the final snapshot is called **`mytest-final-snapshot`** and is located in the same AWS Region as the original DB instance.
4. For **Actions**, choose **Restore Snapshot**.

   The **Restore DB Instance** window appears.
5. For **License Model**, choose **license-included**.
6. Choose the SQL Server DB engine that you want to use.
7. For **DB Instance Identifier**, enter the name for the restored DB instance.
8. Choose **Restore DB Instance**.

For more information about restoring from a snapshot, see Restoring to a DB instance.

Transitioning a Amazon RDS for SQL Server database from OFFLINE to ONLINE

You can transition your Microsoft SQL Server database on an Amazon RDS DB instance from `OFFLINE` to `ONLINE`.

SQL Server method	Amazon RDS method
ALTER DATABASE *db_name* SET ONLINE;	EXEC rdsadmin.dbo.rds_set_database_online *db_name*

Using change data capture for Amazon RDS for SQL Server

Amazon RDS supports change data capture (CDC) for your DB instances running Microsoft SQL Server. CDC captures changes that are made to the data in your tables. It stores metadata about each change, which you can access later. For more information about how CDC works, see Change data capture in the Microsoft documentation. Before you use CDC with your Amazon RDS DB instances, enable it in the database by running `msdb.dbo.rds_cdc_enable_db`. After CDC is enabled, any user who is db_owner of that database can enable or disable CDC on tables in that database.

> **Important**
>
> During restores, CDC will be disabled. All of the related metadata is automatically removed from the database. This applies to snapshot restores and point-in-time restores. After performing one of these types of restores, you can re-enable CDC and re-specify tables to track.

To enable CDC for a DB instance, run the `msdb.dbo.rds_cdc_enable_db` stored procedure.

```
exec msdb.dbo.rds_cdc_enable_db 'database_name'
```

To disable CDC for a DB instance, run the `msdb.dbo.rds_cdc_disable_db` stored procedure.

```
exec msdb.dbo.rds_cdc_disable_db 'database_name'
```

To grant CDC permissions to a user, use the following procedure:

```
go
  GRANT EXECUTE ON msdb.dbo.rds_cdc_enable_db TO User1
  GRANT EXECUTE ON msdb.dbo.rds_cdc_disable_db TO User1
```

Topics

- Tracking tables with change data capture
- Change data capture jobs
- Change data capture for Multi-AZ instances

Tracking tables with change data capture

After CDC is enabled on the database, you can start tracking specific tables. You can choose the tables to track by running sys.sp_cdc_enable_table.

```
--Begin tracking a table
exec sys.sp_cdc_enable_table
   @source_schema           = N'source_schema'
,  @source_name             = N'source_name'
,  @role_name               = N'role_name'

--The following parameters are optional:

--, @capture_instance       = 'capture_instance'
--, @supports_net_changes   = supports_net_changes
--, @index_name             = 'index_name'
--, @captured_column_list   = 'captured_column_list'
--, @filegroup_name         = 'filegroup_name'
--, @allow_partition_switch = 'allow_partition_switch'
;
```

To view the CDC configuration for your tables, run sys.sp_cdc_help_change_data_capture.

```
--View CDC configuration
exec sys.sp_cdc_help_change_data_capture

--The following parameters are optional and must be used together.
--  'schema_name', 'table_name'
```

```
;
```

For more information on CDC tables, functions, and stored procedures in SQL Server documentation, see the following:

- Change data capture stored procedures (Transact-SQL)
- Change data capture functions (Transact-SQL)
- Change data capture tables (Transact-SQL)

Change data capture jobs

When you enable CDC, SQL Server creates the CDC jobs. Database owners (`db_owner`) can view, create, modify, and delete the CDC jobs. However, the RDS system account owns them. Therefore, the jobs aren't visible from native views, procedures, or in SQL Server Management Studio.

To control behavior of CDC in a database, use native SQL Server procedures such as sp_cdc_enable_table and sp_cdc_start_job. To change CDC job parameters, like `maxtrans` and `maxscans`, you can use sp_cdc_change_job..

To get more information regarding the CDC jobs, you can query the following dynamic management views:

- sys.dm_cdc_errors
- sys.dm_cdc_log_scan_sessions
- sysjobs
- sysjobhistory

Change data capture for Multi-AZ instances

If you use CDC on a Multi-AZ instance, make sure the mirror's CDC job configuration matches the one on the principal. CDC jobs are mapped to the `database_id`. If the database IDs on the secondary are different from the principal, then the jobs won't be associated with the correct database. To try to prevent errors after failover, RDS drops and recreates the jobs on the new principal. The recreated jobs use the parameters that the principal recorded before failover.

Although this process runs quickly, it's still possible that the CDC jobs might run before RDS can correct them. Here are three ways to force parameters to be consistent between primary and secondary replicas:

- Use the same job parameters for all the databases that have CDC enabled.
- Before you change the CDC job configuration, convert the Multi-AZ instance to Single-AZ.
- Manually transfer the parameters whenever you change them on the principal.

To view and define the CDC parameters that are used to recreate the CDC jobs after a failover, use `rds_show_configuration` and `rds_set_configuration`.

The following example returns the value set for `cdc_capture_maxtrans`. For any parameter that is set to `RDS_DEFAULT`, RDS automatically configures the value.

```
-- Show configuration for each parameter on either primary and secondary replicas.
exec rdsadmin.dbo.rds_show_configuration 'cdc_capture_maxtrans';
```

To set the configuration on the secondary, run `rdsadmin.dbo.rds_set_configuration`. This procedure sets the parameter values for all of the databases on the secondary server. These settings are used only after a failover. The following example sets the `maxtrans` for all CDC capture jobs to *`1000`*:

```
--To set values on secondary. These are used after failover.
exec rdsadmin.dbo.rds_set_configuration 'cdc_capture_maxtrans', 1000;
```

To set the CDC job parameters on the principal, use sys.sp_cdc_change_job instead.

Using SQL Server Agent for Amazon RDS

With Amazon RDS, you can use SQL Server Agent on a DB instance running Microsoft SQL Server Enterprise Edition, Standard Edition, or Web Edition. SQL Server Agent is a Microsoft Windows service that runs scheduled administrative tasks that are called jobs. You can use SQL Server Agent to run T-SQL jobs to rebuild indexes, run corruption checks, and aggregate data in a SQL Server DB instance.

When you create a SQL Server DB instance, the master user is enrolled in the `SQLAgentUserRole` role.

SQL Server Agent can run a job on a schedule, in response to a specific event, or on demand. For more information, see SQL Server Agent in the Microsoft documentation.

Note

Avoid scheduling jobs to run during the maintenance and backup windows for your DB instance. The maintenance and backup processes that are launched by AWS could interrupt a job or cause it to be canceled.
In Multi-AZ deployments, SQL Server Agent jobs are replicated from the primary host to the secondary host when the job replication feature is turned on. For more information, see Turning on SQL Server Agent job replication.
Multi-AZ deployments have a limit of 10,000 SQL Server Agent jobs. If you need a higher limit, request an increase by contacting Support. Open the AWS Support Center page, sign in if necessary, and choose **Create case**. Choose **Service limit increase**. Complete and submit the form.

To view the history of an individual SQL Server Agent job in SQL Server Management Studio (SSMS), open Object Explorer, right-click the job, and then choose **View History**.

Because SQL Server Agent is running on a managed host in a DB instance, some actions aren't supported:

- Running replication jobs and running command-line scripts by using ActiveX, Windows command shell, or Windows PowerShell aren't supported.
- You can't manually start, stop, or restart SQL Server Agent.
- Email notifications through SQL Server Agent aren't available from a DB instance.
- SQL Server Agent alerts and operators aren't supported.
- Using SQL Server Agent to create backups isn't supported. Use Amazon RDS to back up your DB instance.
- Currently, RDS for SQL Server does not support the use SQL Server Agent tokens.

Turning on SQL Server Agent job replication

You can turn on SQL Server Agent job replication by using the following stored procedure:

```
EXECUTE msdb.dbo.rds_set_system_database_sync_objects @object_types = 'SQLAgentJob';
```

You can run the stored procedure on all SQL Server versions supported by Amazon RDS for SQL Server. Jobs in the following categories are replicated:

- [Uncategorized (Local)]
- [Uncategorized (Multi-Server)]
- [Uncategorized]
- Data Collector
- Database Engine Tuning Advisor
- Database Maintenance
- Full-Text

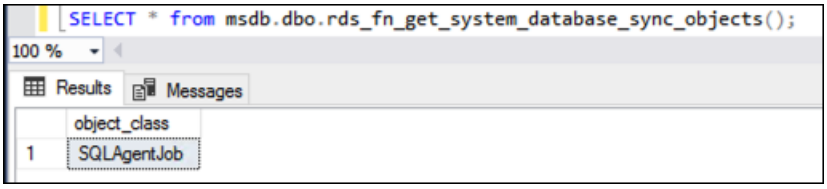
Only jobs that use T-SQL job steps are replicated. Jobs with step types such as SQL Server Integration Services (SSIS), SQL Server Reporting Services (SSRS), Replication, and PowerShell aren't replicated. Jobs that use Database Mail and server-level objects aren't replicated.

> **Important**
>
> The primary host is the source of truth for replication. Before turning on job replication, make sure that your SQL Server Agent jobs are on the primary. If you don't do this, it could lead to the deletion of your SQL Server Agent jobs if you turn on the feature when newer jobs are on the secondary host.

You can use the following function to confirm whether replication is turned on.

```
SELECT * from msdb.dbo.rds_fn_get_system_database_sync_objects();
```

The T-SQL query returns the following if SQL Server Agent jobs are replicating. If they're not replicating, it returns nothing for `object_class`.

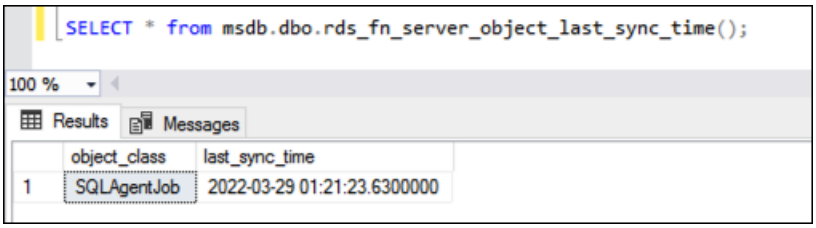


You can use the following function to find the last time objects were synchronized in UTC time.

```
SELECT * from msdb.dbo.rds_fn_server_object_last_sync_time();
```

For example, suppose that you modify a SQL Server Agent job at 01:00. You expect the most recent synchronization time to be after 01:00, indicating that synchronization has taken place.

After synchronization, the values returned for `date_created` and `date_modified` on the secondary node are expected to match.



If you are also using `tempdb` replication, you can enable replication for both SQL Agent jobs and the `tempdb` configuration by providing them in the `@object_type` parameter:

```
EXECUTE msdb.dbo.rds_set_system_database_sync_objects @object_types =
 'SQLAgentJob,TempDbFile';
```

For more information on `tempdb` replication, see TempDB configuration for Multi-AZ deployments.

SQL Server Agent roles

RDS for SQL Server supports the following SQL Server Agent roles with different levels of permissions for managing jobs:

- **SQLAgentUserRole**

  Permissions

  - Create and manage their own jobs, schedules, and operators
  - View the properties of their own jobs and schedules
  - Cannot view or manage jobs created by other users

  This role is suitable for users who need to create and manage their own jobs but do not require access to jobs created by other users.

- **SQLAgentReaderRole**

  Permissions

  - All permissions of SQLAgentUserRole
  - View a list of all jobs and schedules, including those created by others

- View the properties of all jobs
- Review job history

This role is suitable for users who need to monitor the status of all jobs but do not need to manage them.

- **SQLAgentOperatorRole**

  Permissions

  - All permissions of SQLAgentUserRole and SQLAgentReaderRole
  - Execute, stop, or start jobs
  - Manage job history
  - Enable/disable jobs and schedules
  - View operators and proxies

  This role provides the most comprehensive permissions and is suitable for users who need to have full control over all jobs.

Use the following command to assign the roles to your SQL Server login:

```
USE msdb;
EXEC sp_addrolemember 'SQLAgentOperatorRole', 'username';
```

Managing SQLAgentOperatorRole in RDS for SQL Server

To view the current jobs, you must add the SQLAgentOperatorRole to your SQL Server login and remove it before disconnecting from your database.

To visualize the SQL Server Agent tree in the SQL Server Management Studio, follow these instructions:

View SQL Server Agent on SQL Server Management Studio (SSMS)

1. Using the RDS master credentials, login into the RDS SQL Server instance and grant the desired user the SQLAgentUserRole.

   ```
   USE msdb
   GO
   IF NOT EXISTS(SELECT name FROM sys.database_principals WHERE name = 'UserName')
   ```

```
BEGIN
CREATE USER UserName FROM LOGIN UserName
END
GO
ALTER ROLE SQLAgentUserRole ADD MEMBER UserName
GO
GRANT ALTER ON ROLE::[SQLAgentOperatorRole] to UserName
GO
```

These commands create the user on the `msdb` database, in case it doesn't exists. It also adds the user on the SQLAgentUserRole, so the SQL Server Agent tree on SSMS can be seen. Finally, it grants alter permissions on the SQLAgentOperatorRole to the user. This allows the user to add/remove itself from that role.

2. To add yourself to the above-mentioned role, connect to the RDS SQL Server instance, with the user that needs to see the jobs, and run the following script.

```
use msdb
go
ALTER ROLE SQLAgentOperatorRole ADD MEMBER UserName
GO
```

After this, right click on the **Jobs** folder, and choose **Refresh**.

3. When you perform this action, the **Jobs** tab displays a **+** (plus) button. Click to expand the the list of SQL Server Agent Jobs.

4. **Important**

   Before you disconnect from the RDS SQL Server instance, you need to remove yourself from the SQLAgentOperatorRole.

   To remove your login from the SQLAgentOperatorRole, run the following query before disconnecting or closing the Management Studio:

```
USE msdb
GO
ALTER ROLE SQLAgentOperatorRole DROP MEMBER UserName
GO
```

For more information, see Leveraging SQLAgentOperatorRole in RDS SQL Server.

Adding a user to the SQLAgentUser role

To allow an additional login or user to use SQL Server Agent, log in as the master user and do the following:

1. Create another server-level login by using the `CREATE LOGIN` command.
2. Create a user in `msdb` using `CREATE USER` command, and then link this user to the login that you created in the previous step.
3. Add the user to the `SQLAgentUserRole` using the `sp_addrolemember` system stored procedure.

For example, suppose that your master user name is **`admin`** and you want to give access to SQL Server Agent to a user named **`theirname`** with a password **`theirpassword`**. In that case, you can use the following procedure.

To add a user to the SQLAgentUser role

1. Log in as the master user.
2. Run the following commands:

```
--Initially set context to master database
USE [master];
GO
--Create a server-level login named theirname with password theirpassword
CREATE LOGIN [theirname] WITH PASSWORD = 'theirpassword';
GO
--Set context to msdb database
USE [msdb];
GO
--Create a database user named theirname and link it to server-level login
 theirname
CREATE USER [theirname] FOR LOGIN [theirname];
GO
--Added database user theirname in msdb to SQLAgentUserRole in msdb
EXEC sp_addrolemember [SQLAgentUserRole], [theirname];
```

Deleting a SQL Server Agent job

You use the `sp_delete_job` stored procedure to delete SQL Server Agent jobs on Amazon RDS for Microsoft SQL Server.

You can't use SSMS to delete SQL Server Agent jobs. If you try to do so, you get an error message similar to the following:

```
The EXECUTE permission was denied on the object 'xp_regread', database
 'mssqlsystemresource', schema 'sys'.
```

As a managed service, RDS is restricted from running procedures that access the Windows registry. When you use SSMS, it tries to run a process (`xp_regread`) for which RDS isn't authorized.

> **Note**
>
> On RDS for SQL Server, only members of the sysadmin role are allowed to update or delete jobs owned by a different login. For more information, see Leveraging SQLAgentOperatorRole in RDS SQL Server.

To delete a SQL Server Agent job

- Run the following T-SQL statement:

```
EXEC msdb..sp_delete_job @job_name = 'job_name';
```

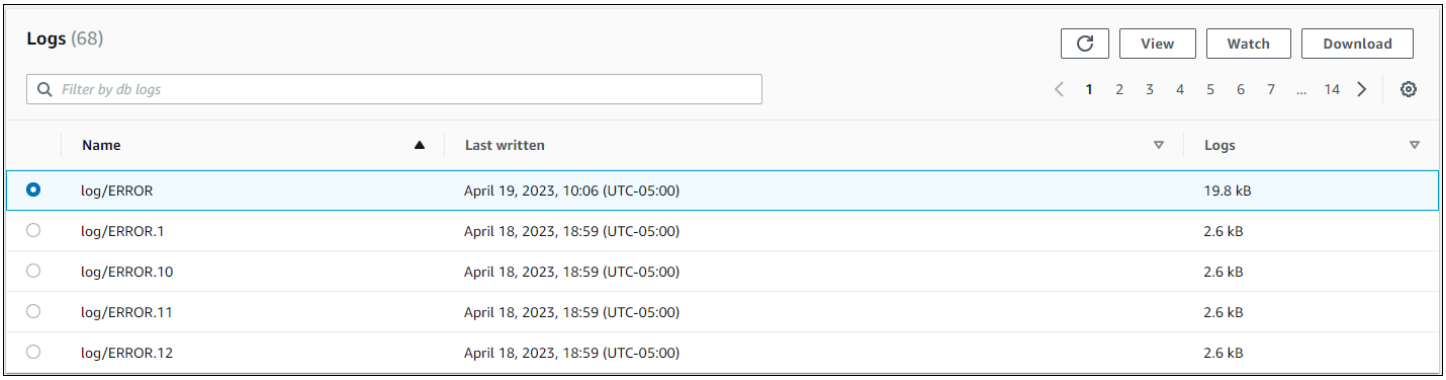
Working with Amazon RDS for Microsoft SQL Server logs

You can use the Amazon RDS console to view, watch, and download SQL Server Agent logs, Microsoft SQL Server error logs, and SQL Server Reporting Services (SSRS) logs.

Watching log files

If you view a log in the Amazon RDS console, you can see its contents as they exist at that moment. Watching a log in the console opens it in a dynamic state so that you can see updates to it in near-real time.

Only the latest log is active for watching. For example, suppose you have the following logs shown:



Only log/ERROR, as the most recent log, is being actively updated. You can choose to watch others, but they are static and will not update.

Archiving log files

The Amazon RDS console shows logs for the past week through the current day. You can download and archive logs to keep them for reference past that time. One way to archive logs is to load them into an Amazon S3 bucket. For instructions on how to set up an Amazon S3 bucket and upload a file, see Amazon S3 basics in the *Amazon Simple Storage Service Getting Started Guide* and click **Get Started**.

Viewing error and agent logs

To view Microsoft SQL Server error and agent logs, use the Amazon RDS stored procedure `rds_read_error_log` with the following parameters:

- **`@index`** – the version of the log to retrieve. The default value is 0, which retrieves the current error log. Specify 1 to retrieve the previous log, specify 2 to retrieve the one before that, and so on.
- **`@type`** – the type of log to retrieve. Specify 1 to retrieve an error log. Specify 2 to retrieve an agent log.

Example

The following example requests the current error log.

```
EXEC rdsadmin.dbo.rds_read_error_log @index = 0, @type = 1;
```

For more information on SQL Server errors, see Database engine errors in the Microsoft documentation.

Working with trace and dump files for Amazon RDS for SQL Server

This section describes working with trace files and dump files for your Amazon RDS DB instances running Microsoft SQL Server.

Generating a trace SQL query

```
declare @rc int
declare @TraceID int
declare @maxfilesize bigint

set @maxfilesize = 5

exec @rc = sp_trace_create @TraceID output,  0, N'D:\rdsdbdata\log\rdstest',
 @maxfilesize, NULL
```

Viewing an open trace

```
select * from ::fn_trace_getinfo(default)
```

Viewing trace contents

```
select * from ::fn_trace_gettable('D:\rdsdbdata\log\rdstest.trc', default)
```

Setting the retention period for trace and dump files

Trace and dump files can accumulate and consume disk space. By default, Amazon RDS purges trace and dump files that are older than seven days.

To view the current trace and dump file retention period, use the `rds_show_configuration` procedure, as shown in the following example.

```
exec rdsadmin..rds_show_configuration;
```

To modify the retention period for trace files, use the `rds_set_configuration` procedure and set the `tracefile retention` in minutes. The following example sets the trace file retention period to 24 hours.

```
exec rdsadmin..rds_set_configuration 'tracefile retention', 1440;
```

To modify the retention period for dump files, use the `rds_set_configuration` procedure and set the `dumpfile retention` in minutes. The following example sets the dump file retention period to 3 days.

```
exec rdsadmin..rds_set_configuration 'dumpfile retention', 4320;
```

For security reasons, you cannot delete a specific trace or dump file on a SQL Server DB instance. To delete all unused trace or dump files, set the retention period for the files to 0.

Amazon RDS for MySQL

Amazon RDS supports several versions of MySQL for DB instances. For complete information about the supported versions, see MySQL on Amazon RDS versions.

To create an Amazon RDS for MySQL DB instance, use the Amazon RDS management tools or interfaces. You can then do the following:

- Resize your DB instance
- Authorize connections to your DB instance
- Create and restore from backups or snapshots
- Create Multi-AZ secondaries
- Create read replicas
- Monitor the performance of your DB instance

To store and access the data in your DB instance, you use standard MySQL utilities and applications.

Amazon RDS for MySQL is compliant with many industry standards. For example, you can use RDS for MySQL databases to build HIPAA-compliant applications. You can use RDS for MySQL databases to store healthcare related information, including protected health information (PHI) under a Business Associate Agreement (BAA) with AWS. Amazon RDS for MySQL also meets Federal Risk and Authorization Management Program (FedRAMP) security requirements. In addition, Amazon RDS for MySQL has received a FedRAMP Joint Authorization Board (JAB) Provisional Authority to Operate (P-ATO) at the FedRAMP HIGH Baseline within the AWS GovCloud (US) Regions. For more information on supported compliance standards, see AWS cloud compliance.

For information about the features in each version of MySQL, see The main features of MySQL in the MySQL documentation.

Before creating a DB instance, complete the steps in Setting up your Amazon RDS environment. When you create a DB instance, the RDS master user gets DBA privileges, with some limitations. Use this account for administrative tasks such as creating additional database accounts.

You can create the following:

- DB instances
- DB snapshots

- Point-in-time restores
- Automated backups
- Manual backups

You can use DB instances running MySQL inside a virtual private cloud (VPC) based on Amazon VPC. You can also add features to your MySQL DB instance by turning on various options. Amazon RDS supports Multi-AZ deployments for MySQL as a high-availability, failover solution.

> **Important**
>
> To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. It also restricts access to certain system procedures and tables that need advanced privileges. You can access your database using standard SQL clients such as the mysql client. However, you can't access the host directly by using Telnet or Secure Shell (SSH).

Topics

- MySQL feature support on Amazon RDS
- MySQL on Amazon RDS versions
- Connecting to your MySQL DB instance
- Securing MySQL DB instance connections
- Improving query performance for RDS for MySQL with Amazon RDS Optimized Reads
- Improving write performance with RDS Optimized Writes for MySQL
- Upgrades of the RDS for MySQL DB engine
- Upgrading a MySQL DB snapshot engine version
- Importing data into an Amazon RDS for MySQL DB instance
- Working with MySQL replication in Amazon RDS
- Configuring active-active clusters for RDS for MySQL
- Exporting data from a MySQL DB instance by using replication
- Options for MySQL DB instances
- Parameters for MySQL
- Common DBA tasks for MySQL DB instances

- Local time zone for MySQL DB instances
- Known issues and limitations for Amazon RDS for MySQL
- RDS for MySQL stored procedure reference

MySQL feature support on Amazon RDS

RDS for MySQL supports most of the features and capabilities of MySQL. Some features might have limited support or restricted privileges.

You can filter new Amazon RDS features on the What's New with Database? page. For **Products**, choose **Amazon RDS**. Then search using keywords such as **`MySQL 2022`**.

> **Note**
>
> The following lists are not exhaustive.

Topics

- MySQL feature support on Amazon RDS for MySQL major versions
- Supported storage engines for RDS for MySQL
- Using memcached and other options with MySQL on Amazon RDS
- InnoDB cache warming for MySQL on Amazon RDS
- Inclusive language changes for RDS for MySQL 8.4
- MySQL features not supported by Amazon RDS

MySQL feature support on Amazon RDS for MySQL major versions

In the following sections, find information about MySQL feature support on Amazon RDS for MySQL major versions:

Topics

- MySQL 8.4 support on Amazon RDS

For information about supported minor versions of Amazon RDS for MySQL, see Supported MySQL minor versions on Amazon RDS.

MySQL 8.4 support on Amazon RDS

Amazon RDS supports the following new features for your DB instances running MySQL version 8.4 or higher.

- **Cryptographic library** – RDS for MySQL replaced OpenSSL with AWS Libcrypto (AWS-LC), which is FIPS 140-3 certified. For more information, see the AWS-LC GitHub repository at https://github.com/aws/aws-lc.
- **TLS changes** – RDS for MySQL only supports TLS 1.2 and TLS 1.3. For more information, see the section called "SSL/TLS support with MySQL".
- **memcached support** – The memcached interface is no longer available on MySQL 8.4. For more information, see MySQL memcached support.
- **Default authentication plugin** – The default authentication plugin is `caching_sha2_password`. For more information, see the section called "MySQL default authentication plugin".
- **`mysqlpump` client utility** – The `mysqlpump` client utility is no longer available in MySQL 8.4. For more information, see Role-based privilege model for RDS for MySQL and mysqldump and mysqlpump in the *AWS Prescriptive Guidance*.
- **Managed replication stored procedures** – When using stored procedures to manage replication with a replication user configured with `caching_sha2_password`, you must configure TLS by specifying `SOURCE_SSL=1`. `caching_sha2_password` is the default authentication plugin for RDS for MySQL 8.4.
- **Parameter behavior changes** – The following parameters changed for MySQL 8.4.
  - `innodb_dedicated_server` – This parameter is now enabled by default. For more information, see Configuring buffer pool size and redo log capacity in MySQL 8.4.
  - `innodb_buffer_pool` – The database engine now calculates this parameter, but you can override this setting. For more information, see Configuring buffer pool size and redo log capacity in MySQL 8.4.
  - `innodb_redo_log_capacity` – This parameter now controls the size of the redo log files. The database engine now calculates this parameter, but you can override this setting. For more information, see Configuring buffer pool size and redo log capacity in MySQL 8.4.
- **Deprecated or removed parameters** – RDS for MySQL removed the following parameters from parameter groups for MySQL 8.4 DB instances. The `innodb_redo_log_capacity` parameter now controls the size of the redo log files.
  - `innodb_log_file_size`
  - `innodb_log_files_in_group`
- **New default values for parameters** – The following parameters have new default values for MySQL 8.4 DB instances:

- Various MySQL community parameters related to performance changed. For more information, see What is New in MySQL 8.4 since MySQL 8.0.

  We recommend that you test your application's performance on RDS for MySQL 8.4 before migrating a production instance.

- `innodb_purge_threads` – The default value is set to the formula `LEAST({DBInstanceVCPU/2},4)` to prevent the InnoDB history list length from growing too large.
- `group_replication_exit_state_action` – The default value is `OFFLINE_MODE`, which aligns with the default in MySQL Community. For more information, see Considerations and best practices for RDS for MySQL active-active clusters.
- `binlog_format` – The default value is `ROW`, which aligns with the default in MySQL Community. You can modify the parameter for Single-AZ DB instances or Multi-AZ DB instances, but not for Multi-AZ DB clusters. Multi-AZ DB clusters use semisynchronous replication, and when `binlog_format` is set to `MIXED` or `STATEMENT`, replication fails.

- **Inclusive language changes** – RDS for MySQL 8.4 includes changes from RDS for MySQL 8.0 related to keywords and system schemas for inclusive language. For more information, see Inclusive language changes for RDS for MySQL 8.4.

For a list of all MySQL 8.4 features and changes, see What Is New in MySQL 8.4 since MySQL 8.0 in the MySQL documentation.

For a list of unsupported features, see the section called “Features not supported”.

Supported storage engines for RDS for MySQL

While MySQL supports multiple storage engines with varying capabilities, not all of them are optimized for recovery and data durability. Amazon RDS fully supports the InnoDB storage engine for MySQL DB instances. Amazon RDS features such as Point-In-Time restore and snapshot restore require a recoverable storage engine and are supported for the InnoDB storage engine only. For more information, see MySQL memcached support.

The Federated Storage Engine is currently not supported by Amazon RDS for MySQL.

For user-created schemas, the MyISAM storage engine does not support reliable recovery and can result in lost or corrupt data when MySQL is restarted after a recovery, preventing Point-In-Time

restore or snapshot restore from working as intended. However, if you still choose to use MyISAM with Amazon RDS, snapshots can be helpful under some conditions.

> **Note**
>
> System tables in the `mysql` schema can be in MyISAM storage.

If you want to convert existing MyISAM tables to InnoDB tables, you can use the `ALTER TABLE` command (for example, `alter table TABLE_NAME engine=innodb;`). Bear in mind that MyISAM and InnoDB have different strengths and weaknesses, so you should fully evaluate the impact of making this switch on your applications before doing so.

MySQL 5.1, 5.5, and 5.6 are no longer supported in Amazon RDS. However, you can restore existing MySQL 5.1, 5.5, and 5.6 snapshots. When you restore a MySQL 5.1, 5.5, or 5.6 snapshot, the DB instance is automatically upgraded to MySQL 5.7.

Using memcached and other options with MySQL on Amazon RDS

Most Amazon RDS DB engines support option groups that allow you to select additional features for your DB instance. RDS for MySQL DB instances support the `memcached` option, a simple, key-based cache. For more information about `memcached` and other options, see Options for MySQL DB instances. For more information about working with option groups, see Working with option groups.

InnoDB cache warming for MySQL on Amazon RDS

InnoDB cache warming can provide performance gains for your MySQL DB instance by saving the current state of the buffer pool when the DB instance is shut down, and then reloading the buffer pool from the saved information when the DB instance starts up. This bypasses the need for the buffer pool to "warm up" from normal database use and instead preloads the buffer pool with the pages for known common queries. The file that stores the saved buffer pool information only stores metadata for the pages that are in the buffer pool, and not the pages themselves. As a result, the file does not require much storage space. The file size is about 0.2 percent of the cache size. For example, for a 64 GiB cache, the cache warming file size is 128 MiB. For more information on InnoDB cache warming, see Saving and restoring the buffer pool state in the MySQL documentation.

RDS for MySQL DB instances support InnoDB cache warming. To enable InnoDB cache warming, set the `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` parameters to 1 in the parameter group for your DB instance. Changing these parameter values in a parameter group will affect all MySQL DB instances that use that parameter group. To enable InnoDB cache warming for specific MySQL DB instances, you might need to create a new parameter group for those instances. For information on parameter groups, see Parameter groups for Amazon RDS.

InnoDB cache warming primarily provides a performance benefit for DB instances that use standard storage. If you use PIOPS storage, you do not commonly see a significant performance benefit.

Important

If your MySQL DB instance does not shut down normally, such as during a failover, then the buffer pool state will not be saved to disk. In this case, MySQL loads whatever buffer pool file is available when the DB instance is restarted. No harm is done, but the restored buffer pool might not reflect the most recent state of the buffer pool prior to the restart. To ensure that you have a recent state of the buffer pool available to warm the InnoDB cache on startup, we recommend that you periodically dump the buffer pool "on demand." You can create an event to dump the buffer pool automatically and on a regular interval. For example, the following statement creates an event named `periodic_buffer_pool_dump` that dumps the buffer pool every hour.

```
CREATE EVENT periodic_buffer_pool_dump
ON SCHEDULE EVERY 1 HOUR
DO CALL mysql.rds_innodb_buffer_pool_dump_now();
```

For more information on MySQL events, see Event syntax in the MySQL documentation.

Dumping and loading the buffer pool on demand

You can save and load the InnoDB cache "on demand."

- To dump the current state of the buffer pool to disk, call the mysql.rds_innodb_buffer_pool_dump_now stored procedure.

- To load the saved state of the buffer pool from disk, call the mysql.rds_innodb_buffer_pool_load_now stored procedure.
- To cancel a load operation in progress, call the mysql.rds_innodb_buffer_pool_load_abort stored procedure.

Inclusive language changes for RDS for MySQL 8.4

RDS for MySQL 8.4 includes changes from the MySQL 8.4 community edition related to keywords and system schemas for inclusive language. For example, the `SHOW REPLICA STATUS` command replaces `SHOW SLAVE STATUS`.

Topics

- Configuration parameter name changes
- Stored procedure name changes

Configuration parameter name changes

The following configuration parameters have new names in RDS for MySQL 8.4.

For compatibility, you can check the parameter names in the `mysql` client by using either name. You can only use the new names when modifying values in a custom MySQL 8.4 parameter group. For more information, see Default and custom parameter groups.

Name to be removed	New or preferred name
`init_slave`	`init_replica`
`log_slave_updates`	`log_replica_updates`
`log_slow_slave_statements`	`log_slow_replica_statements`
`rpl_stop_slave_timeout`	`rpl_stop_replica_timeout`
`skip_slave_start`	`skip_replica_start`

Name to be removed	New or preferred name
slave_checkpoint_group	replica_checkpoint_group
slave_checkpoint_period	replica_checkpoint_period
slave_compressed_protocol	replica_compressed_protocol
slave_exec_mode	replica_exec_mode
slave_load_tmpdir	replica_load_tmpdir
slave_max_allowed_packet	replica_max_allowed_packet
slave_net_timeout	replica_net_timeout
slave_parallel_type	replica_parallel_type
slave_parallel_workers	replica_parallel_workers
slave_pending_jobs_size_max	replica_pending_jobs_size_max
slave_preserve_commit_order	replica_preserve_commit_order
slave_skip_errors	replica_skip_errors
slave_sql_verify_checksum	replica_sql_verify_checksum
slave_transaction_retries	replica_transaction_retries

Name to be removed	New or preferred name	
slave_type_conversions	replica_type_conversions	
sql_slave_skip_counter	sql_replica_skip_counter	

> **Note**
>
> The parameter replica_allow_batching isn't available because Amazon RDS doesn't support NDB clusters.

Stored procedure name changes

The following stored procedures have new names in RDS for MySQL 8.4.

For compatibility, you can use either name in the initial RDS for MySQL 8.4 release. The old procedure names are to be removed in a future release. For more information, see Configuring, starting, and stopping binary log (binlog) replication.

Name to be removed	New or preferred name	
mysql.rds_next_master_log	mysql.rds_next_source_log	
mysql.rds_reset_external_master	mysql.rds_reset_external_source	
mysql.rds_set_external_master	mysql.rds_set_external_source	
mysql.rds_set_external_master_with_auto_position	mysql.rds_set_external_source_with_auto_position	

Name to be removed	New or preferred name	
`mysql.rds_set_external_master_with_delay`	`mysql.rds_set_external_source_with_delay`	
`mysql.rds_set_master_auto_position`	`mysql.rds_set_source_auto_position`	

MySQL features not supported by Amazon RDS

Amazon RDS doesn't currently support the following MySQL features:

- Authentication Plugin
- Error Logging to the System Log
- InnoDB Tablespace Encryption
- NDB clusters
- Password Strength Plugin
- Persisted system variables
- Rewriter Query Rewrite Plugin
- Semisynchronous replication, except for Multi-AZ DB clusters
- Transportable tablespace
- X Plugin

To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. It also restricts access to certain system procedures and tables that require advanced privileges. Amazon RDS supports access to databases on a DB instance using any standard SQL client application. Amazon RDS doesn't allow direct host access to a DB instance by using Telnet, Secure Shell (SSH), or Windows Remote Desktop Connection. When you create a DB instance, you are assigned as *db_owner* for all databases on that instance, and you have all database-level permissions except for those used for backups. Amazon RDS manages backups for you.

MySQL on Amazon RDS versions

For MySQL, version numbers are organized as version = X.Y.Z. In Amazon RDS terminology, X.Y denotes the major version, and Z is the minor version number. For Amazon RDS implementations, a version change is considered major if the major version number changes—for example, going from version 5.7 to 8.0. A version change is considered minor if only the minor version number changes—for example, going from version 8.0.32 to 8.0.34.

Topics

- Supported MySQL minor versions on Amazon RDS
- Supported MySQL major versions on Amazon RDS
- Amazon RDS Extended Support versions for RDS for MySQL
- Working with the Database Preview environment
- MySQL version 9.5 in the Database Preview environment
- MySQL version 9.4 in the Database Preview environment
- MySQL version 9.3 in the Database Preview environment
- Deprecated versions for Amazon RDS for MySQL

Supported MySQL minor versions on Amazon RDS

Amazon RDS currently supports the following minor versions of MySQL.

> **Note**
>
> Amazon RDS Extended Support isn't available for minor versions.

The following table shows the minor versions of MySQL 8.4 that Amazon RDS currently supports.

MySQL engine version	Community release date	RDS release date	RDS end of standard support date
8.4.8	20 January 2026	3 February 2026	3 February 2027
8.4.7	21 October 2025	13 November 2025	30 November 2026

MySQL engine version	Community release date	RDS release date	RDS end of standard support date
8.4.6	22 July 2025	1 August 2025	30 September 2026
8.4.5	15 April 2025	29 April 2025	30 September 2026
8.4.4	21 January 2025	19 February 2025	31 May 2026
8.4.3	15 October 2024	21 November 2024	31 May 2026

The following table shows the minor versions of MySQL 8.0 that Amazon RDS currently supports.

Note

Minor versions can reach end of standard support before major versions do. For example, minor version 8.0.28 reached its end of standard support date on March 28, 2024 while major version 8.0 reaches this date on July 31, 2026. RDS will support additional 8.0.* minor versions that the MySQL community releases between these dates. We recommend that you upgrade to the latest available minor version as often as possible for all major versions.

MySQL engine version	Community release date	RDS release date	RDS end of standard support date
8.0.45	20 January 2026	3 February 2026	31 July 2026
8.0.44	21 October 2025	13 November 2025	31 July 2026
8.0.43	22 July 2025	1 August 2025	31 July 2026
8.0.42	15 April 2025	29 April 2025	31 July 2026
8.0.41	21 January 2025	19 February 2025	31 May 2026
8.0.40	15 October 2024	13 November 2024	31 May 2026

The following table shows the minor versions of MySQL 5.7 that are available under Amazon RDS Extended Support.

> **Note**
>
> Minor versions can reach end of Extended Support before major versions do. For example, minor version 5.7.44-RDS.20240529 reaches its end of Extended Support date in September 2025 while major version 5.7 reaches this date on February 28, 2027. RDS will generate and release additional 5.7.44-RDS.xxyyzz minor versions between these dates. We recommend that you upgrade to the latest available minor version as often as possible for all major versions.

MySQL engine version	Community release date	RDS release date	RDS end of Extended Support date
5.7.44-RDS.20260212*	Not applicable	26 February 2026	28 February 2027
5.7.44-RDS.20251212*	Not applicable	12 December 2025	30 December 2026
5.7.44-RDS.20250818*	Not applicable	15 September 2025	30 September 2026
5.7.44-RDS.20250508*	Not applicable	20 May 2025	30 September 2026
5.7.44-RDS.20250213*	Not applicable	12 March 2025	30 September 2026
5.7.44-RDS.20250103*	Not applicable	13 February 2025	31 May 2026

* MySQL Community retired major version 5.7 and won't be releasing new minor versions. This is a minor version that Amazon RDS released with critical security patches and bug fixes for MySQL 5.7 databases that are covered under RDS Extended Support. For more information about these minor

versions, see the section called "RDS Extended Support versions". For more information about RDS Extended Support, see Amazon RDS Extended Support with Amazon RDS.

You can specify any currently supported MySQL version when creating a new DB instance. You can specify the major version (such as MySQL 5.7), and any supported minor version for the specified major version. If no version is specified, Amazon RDS defaults to a supported version, typically the most recent version. If a major version is specified but a minor version is not, Amazon RDS defaults to a recent release of the major version you have specified. To see a list of supported versions, as well as defaults for newly created DB instances, run the `describe-db-engine-versions` AWS CLI command.

For example, to list the supported engine versions for RDS for MySQL, run the following CLI command:

```
aws rds describe-db-engine-versions --engine mysql --query "*[].
{Engine:Engine,EngineVersion:EngineVersion}" --output text
```

The default MySQL version might vary by AWS Region. To create a DB instance with a specific minor version, specify the minor version during DB instance creation. You can determine the default minor version for an AWS Region by running the following AWS CLI command:

```
aws rds describe-db-engine-versions --default-only --engine mysql
 --engine-version major_engine_version --region region --query "*[].
{Engine:Engine,EngineVersion:EngineVersion}" --output text
```

Replace *`major_engine_version`* with the major engine version, and replace *`region`* with the AWS Region. For example, the following AWS CLI command returns the default MySQL minor engine version for the 5.7 major version and the US West (Oregon) AWS Region (us-west-2):

```
aws rds describe-db-engine-versions --default-only --engine mysql --engine-version 5.7
 --region us-west-2 --query "*[].{Engine:Engine,EngineVersion:EngineVersion}" --output
 text
```

With Amazon RDS, you control when to upgrade your MySQL instance to a new major version supported by Amazon RDS. You can maintain compatibility with specific MySQL versions, test new versions with your application before deploying in production, and perform major version upgrades at times that best fit your schedule.

When automatic minor version upgrade is enabled, your DB instance will be upgraded automatically to new MySQL minor versions as they are supported by Amazon RDS. This patching occurs during your scheduled maintenance window. You can modify a DB instance to enable or disable automatic minor version upgrades.

If you opt out of automatically scheduled upgrades, you can manually upgrade to a supported minor version release by following the same procedure as you would for a major version update. For information, see Upgrading a DB instance engine version.

Amazon RDS currently supports the following upgrades for major versions of the MySQL database engine:

- MySQL 5.7 to MySQL 8.0
- MySQL 8.0 to MySQL 8.4

Because major version upgrades involve some compatibility risk, they do not occur automatically; you must make a request to modify the DB instance. You should thoroughly test any upgrade before upgrading your production instances. For information about upgrading a MySQL DB instance, see Upgrades of the RDS for MySQL DB engine.

You can test a DB instance against a new version before upgrading by creating a DB snapshot of your existing DB instance, restoring from the DB snapshot to create a new DB instance, and then initiating a version upgrade for the new DB instance. You can then experiment safely on the upgraded clone of your DB instance before deciding whether or not to upgrade your original DB instance.

MySQL minor versions on Amazon RDS

For the changes that the MySQL community made to the minor versions, see Critical Patch Updates, Security Alerts and Bulletins on the Oracle website. Under **Critical Patch Update**, choose the month when Oracle released the minor version. And then choose the MySQL minor version under **Affected Products and Versions**.

Minor versions

- MySQL version 8.4.8
- MySQL version 8.4.7
- MySQL version 8.4.6
- MySQL version 8.4.5

- MySQL version 8.4.4
- MySQL version 8.0.45
- MySQL version 8.0.44
- MySQL version 8.0.43
- MySQL version 8.0.42
- MySQL version 8.0.41
- MySQL version 8.0.40
- MySQL version 8.0.39
- MySQL version 8.0.37

MySQL version 8.4.8

MySQL version 8.4.8 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025c`.

- Fixed an issue which can cause some SQL statements to not get logged in the audit log.

MySQL version 8.4.7

MySQL version 8.4.7 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

MySQL version 8.4.6

MySQL version 8.4.6 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

MySQL version 8.4.5

MySQL version 8.4.5 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025b`.

MySQL version 8.4.4

MySQL version 8.4.4 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.
- Fixed a bug that caused a collation error while executing the Amazon RDS stored procedures `mysql.rds_set_configuration` and `mysql.rds_kill`.

MySQL version 8.0.45

MySQL version 8.0.45 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025c`.

- Fixed an issue which can cause some SQL statements to not get logged in the audit log.

MySQL version 8.0.44

MySQL version 8.0.44 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

MySQL version 8.0.43

MySQL version 8.0.43 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

MySQL version 8.0.42

MySQL version 8.0.42 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025b`.

MySQL version 8.0.41

MySQL version 8.0.41 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Updated the time zone information to base it on `tzdata2025a`.
- Fixed a bug that caused a collation error while executing the Amazon RDS stored procedures `mysql.rds_set_configuration` and `mysql.rds_kill`.

MySQL version 8.0.40

MySQL version 8.0.40 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Fixed a bug that caused character set mismatch failures during database upgrades.

MySQL version 8.0.39

MySQL version 8.0.39 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Fixed a bug that prevented `sql_log_off` from working properly with the `SESSION_VARIABLES_ADMIN` privilege.
- Fixed a bug that prevented the master user from being able to grant the `SESSION_VARIABLE_ADMIN` privilege other database users.
- Fixed a bug that caused an illegal mix of collation while executing RDS-provided stored procedures.

MySQL version 8.0.37

MySQL version 8.0.37 is now available on Amazon RDS. This release contains fixes and improvements added by the MySQL community and Amazon RDS.

New features and enhancements

- Fixed a bug with executing an instant DDL statement followed by an UPDATE that lead to an assertion failure.

Supported MySQL major versions on Amazon RDS

RDS for MySQL major versions are available under standard support at least until community end of life for the corresponding community version. You can continue running a major version past its RDS end of standard support date for a fee. For more information, see Amazon RDS Extended Support with Amazon RDS and Amazon RDS for MySQL pricing.

You can use the following dates to plan your testing and upgrade cycles.

> **Note**
>
> You can also view information about support dates for major engine versions by using the AWS CLI or the RDS API. For more information, see Viewing support dates for engine versions in Amazon RDS Extended Support.

MySQL major version	Communit release date	RDS release date	Communit end of life date	RDS end of standard support date	RDS start of Extended Support year 1 pricing date	RDS start of Extended Support year 3 pricing date	RDS end of Extended Support date
MySQL 8.4	30 April 2024	21 November 2024	30 April 2029	31 July 2029	1 August, 2029	1 August 2031	31 July 2032

MySQL major version	Communit release date	RDS release date	Communit end of life date	RDS end of standard support date	RDS start of Extended Support year 1 pricing date	RDS start of Extended Support year 3 pricing date	RDS end of Extended Support date
MySQL 8.0	19 April 2018	23 October 2018	30 April 2026	31 July 2026	1 August 2026	1 August 2028	31 July 2029
MySQL 5.7*	21 October 2015	22 February 2016	31 October 2023	29 February 2024	1 March 2024	1 March 2026	28 February 2027

* MySQL 5.7 is now only available under RDS Extended Support. For more information, see Amazon RDS Extended Support with Amazon RDS.

Amazon RDS Extended Support versions for RDS for MySQL

The following content lists all releases of RDS Extended Support for RDS for MySQL versions.

Releases

- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20260212
- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20251212
- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250818
- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250508
- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250213
- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250103
- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240808
- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240529
- RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240408

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20260212

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20260212 is available.

Bugs fixed:

- Refresh test certificate used to test fix for bug 22295186.
- Fixes a memory leak with prefix index on blob columns.

CVEs fixed:

- CVE-2026-21936
- CVE-2026-21968
- CVE-2026-21941
- CVE-2026-21948

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20251212

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20251212 is available.

Bugs fixed:

- Fixed an issue with server startup when the buffer pool size exceeded the upper limit.
- Fixed read from `INFORMATION_SCHEMA.INNODB_LOCKS` that caused the server to exit abnormally.
- Fixed an issue with JUnit reporting support in MySQL Test Run (MTR).
- Fixed compilation issues when building with `-DWITH_INNODB_MEMCACHED=ON` option.
- Fixed an issue with MySQL Test Run (MTR) execution for bug 25182306.

CVEs fixed:

- CVE-2025-53054
- CVE-2025-53044
- CVE-2025-53045
- CVE-2025-53062

- CVE-2025-53040
- CVE-2025-53042
- CVE-2025-53067

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250818

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250818 is available.

Bugs fixed:

- Fixed an issue where the query rewrite plugin failed when the server operated with `autocommit=OFF`.
- Fixed a permission issue that prevented Debian and Ubuntu builds from running in rootless mode.
- Fixed missing update for bug 30875669.

CVEs fixed:

- CVE-2025-50082
- CVE-2025-50083
- CVE-2025-50079
- CVE-2025-50084
- CVE-2025-50087
- CVE-2025-50091
- CVE-2025-50101
- CVE-2025-50102
- CVE-2025-50098
- CVE-2025-53023
- CVE-2025-50081
- CVE-2025-50085
- CVE-2025-50077
- CVE-2025-50088
- CVE-2025-50092

- CVE-2025-50099
- CVE-2025-50096

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250508

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250508 is available.

Bugs fixed:

- Fixed virtual index unstable after rollback when `index_id` is greater than max `uint32`.
- Fixed Tests fails with memory issue.
- Fixed `<COMMAND_CLASS>` is empty for `<NAME>Execute</NAME>`.
- Fixed Compile MySQL with GCC 14 [noclose 5.7].

CVEs fixed:

- CVE-2025-30682
- CVE-2025-30687
- CVE-2025-30688
- CVE-2025-21581
- CVE-2025-21585
- CVE-2025-30689
- CVE-2025-30722
- CVE-2025-21577
- CVE-2025-30693
- CVE-2025-30695
- CVE-2025-30703

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250213

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250213 is available.

Bugs fixed:

- Fixed InnoDB failing assertion `result != FTS_INVALID`.

- Fixed crashing and widespread corruption of spatial indexes after `ALTER TABLE` operation rebuilds InnoDB table using the `INPLACE` algorithm.
- Fixed `ON DELETE CASCADE` with generated column crashes in `innobase_get_computed_value`.
- Fixed assertion failure in `row_MySQL_pad_col`.
- Fixed an issue where online DDL causes the following error: `ERROR 1712 (HY000): Index PRIMARY is corrupted`.
- Fixed crashes at `Item_rollup_sum_switcher::current_arg`.
- Fixed an issue where the database cache is not flushed on `DROP USER`.
- Fixed buffer overrun in `my_print_help`.
- Fixed an InnoDB issue where `FULLTEXT` index limits `FTS_DOC_ID` to max unsigned 32-bit value.

CVEs fixed:

- CVE-2025-21497
- CVE-2025-21555
- CVE-2025-21559
- CVE-2025-21490
- CVE-2025-21491
- CVE-2025-21500
- CVE-2025-21501
- CVE-2025-21540
- CVE-2025-21543
- CVE-2025-21520

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250103

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20250103 is available.

Bugs fixed:

- Fixed FTS clean-up issue when dropping and adding a `FULLTEXT` index in the same transaction.
- Optimized the memory allocation timing in the MySQL client to prevent any potential leaks.

- Fixed the truncation of results at 34 bytes when using the `UNION` operator.
- Fixed potential out-of-bounds access due to `ulong bitmask` in the authorization code.

CVEs fixed:

- CVE-2024-21230
- CVE-2024-21201
- CVE-2024-21241
- CVE-2024-21203

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240808

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240808 is available.

Bugs fixed:

- Fixed assertion failure related to dictionary column index.
- Fixed issue with the `is_binlog_cache_empty()` function.
- Fixed `heap-use-after-free` errors in `sql/item.cc` files.
- Fixed several spatial index issues by disabling them for `index-only` reads.
- Fixed instrumentation issue with the `LOCK_ORDER: CONNECTION_CONTROL` plugin.
- Fixed threads getting stuck with the `CONNECTION_CONTROL` plugin.
- Fixed `PSI_THREAD_INFO` not updating for `PREPARED STATEMENTS`.
- Fixed double processing of FTS index words with `innodb_optimize_fulltext_only`.

CVEs fixed:

- CVE-2024-21177

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240529

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240529 is available.

Bugs fixed:

- Fixed `field.cc` assertion failure by implementing `fix_after_pullout`.

- Fixed a null pointer failure when returning metadata to the client for certain SQL queries. These queries contained dynamic parameters and subqueries in `SELECT` clauses.
- Fixed incorrect results when using `GROUP BY` for loose index scans, or scans of noncontiguous ranges of an index.
- Fixed loss of GTID information on MySQL crash during persistence.
- Fixed a race condition that could cause an InnoDB transaction to hang indefinitely.
- Fixed a race condition in Group Replication's certification information cleanup.
- Fixed backward index scan issue with concurrent page operations.
- Fixed an inconsistent full-text search (FTS) state issue in concurrent scenarios.
- Fixed assertion issue with change buffer on deleting tables.
- Unified behavior for calling `deinit` function across all plugin types.

CVEs fixed:

- CVE-2024-20963
- CVE-2024-20993
- CVE-2024-20998
- CVE-2024-21009
- CVE-2024-21054
- CVE-2024-21055
- CVE-2024-21057
- CVE-2024-21062
- CVE-2024-21008
- CVE-2024-21013
- CVE-2024-21047
- CVE-2024-21087
- CVE-2024-21096

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240408

RDS Extended Support for RDS for MySQL version 5.7.44-RDS.20240408 is available.

This release contains patches for the following CVEs:

- CVE-2024-20963

Working with the Database Preview environment

In July 2023, Oracle announced a new release model for MySQL. This model includes two types of releases: Innovation Releases and LTS releases. Amazon RDS makes MySQL Innovation Releases available in the RDS Preview environment. To learn more about the MySQL Innovation releases, see Introducing MySQL Innovation and Long-Term Support (LTS) versions.

RDS for MySQL DB instances in the Database Preview environment are functionally similar to other RDS for MySQL DB instances. However, you can't use the Database Preview environment for production workloads.

Preview environments have the following limitations:

- Amazon RDS deletes all DB instances 60 days after you create them, along with any backups and snapshots.
- You can only use General Purpose SSD and Provisioned IOPS SSD storage.
- You can't get help from Support with DB instances. Instead, you can post your questions to the AWS-managed Q&A community, AWS re:Post.
- You can't copy a snapshot of a DB instance to a production environment.

The following options are supported by the preview.

- You can create DB instances using db.m6i, db.r6i, db.m6g, db.m5, db.t3, db.r6g, and db.r5 DB instance classes. For more information about RDS instance classes, see DB instance classes.
- You can use both single-AZ and multi-AZ deployments.
- You can use standard MySQL dump and load functions to export databases from or import databases to the Database Preview environment.

Features not supported in the Database Preview environment

The following features aren't available in the Database Preview environment:

- Cross-Region snapshot copy
- Cross-Region read replicas

- RDS Proxy

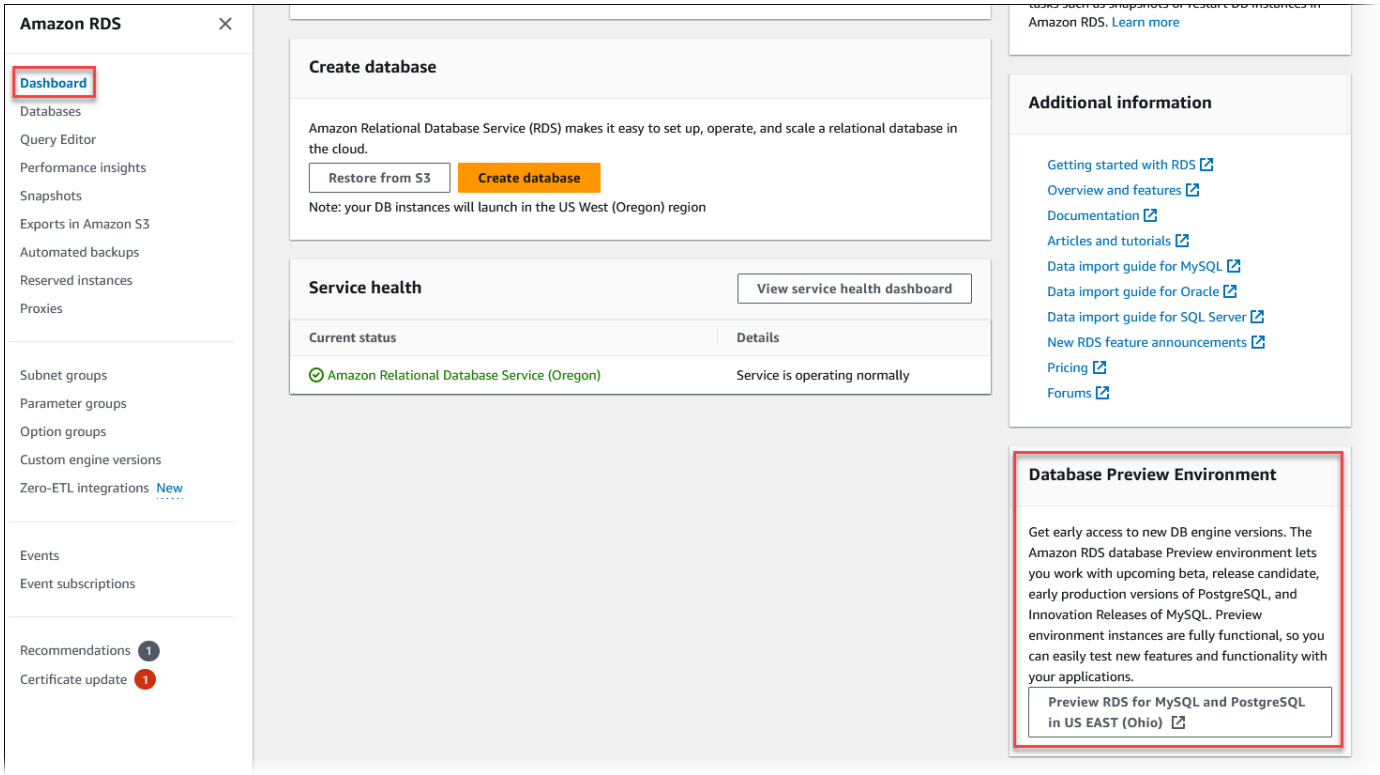
Creating a new DB instance in the Database Preview environment

You can create a DB instance in the Database Preview environment using the AWS Management Console, AWS CLI, or RDS API.

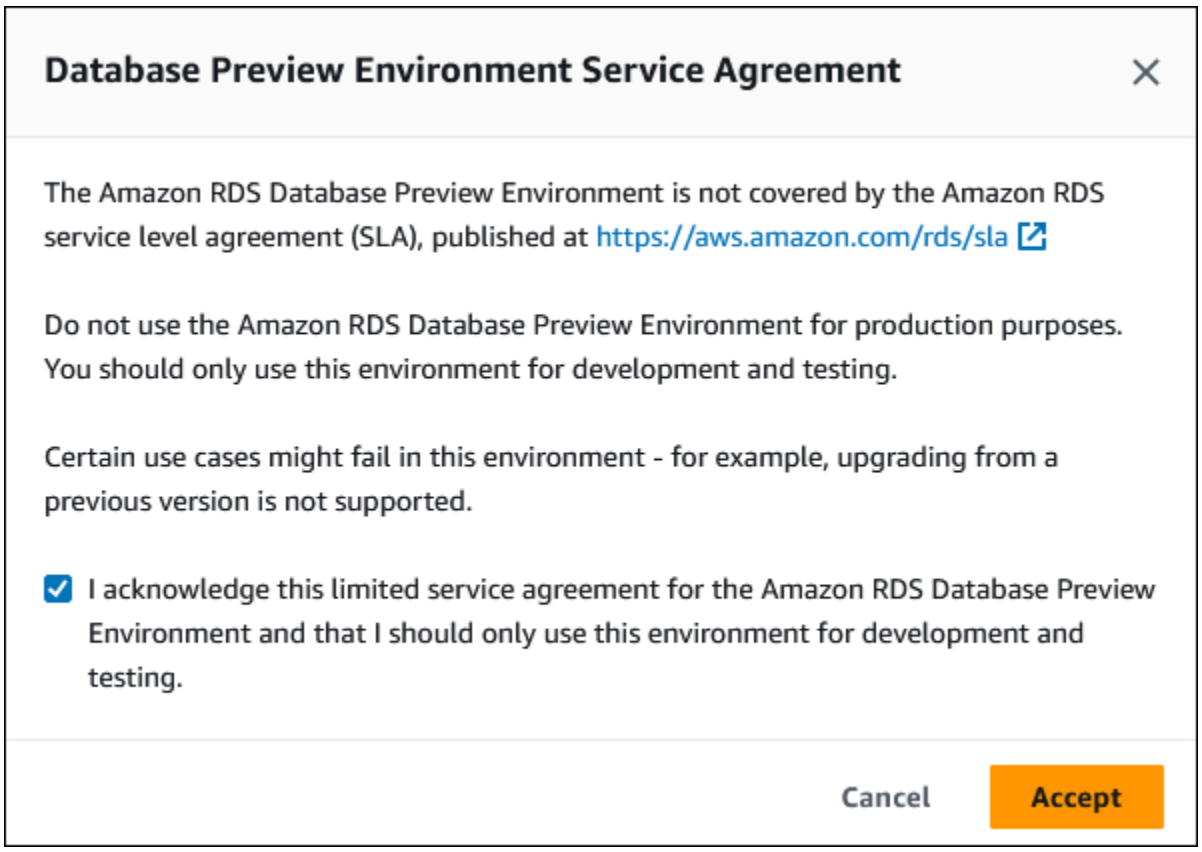
Console

To create a DB instance in the Database Preview environment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Dashboard** from the navigation pane.
3. In the **Dashboard** page, locate the **Database Preview Environment** section, as shown in the following image.



You can navigate directly to the Database Preview environment. Before you can proceed, you must acknowledge and accept the limitations.



4. To create the RDS for MySQL DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the Console procedure in Creating a DB instance.

AWS CLI

To create a DB instance in the Database Preview environment using the AWS CLI, use the following endpoint.

```
rds-preview.us-east-2.amazonaws.com
```

To create the RDS for MySQL DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the AWS CLI procedure in Creating a DB instance.

RDS API

To create a DB instance in the Database Preview environment using the RDS API, use the following endpoint.

```
rds-preview.us-east-2.amazonaws.com
```

To create the RDS for MySQL DB instance, follow the same process that you would for creating any Amazon RDS DB instance. For more information, see the RDS API procedure in Creating a DB instance.

MySQL version 9.5 in the Database Preview environment

MySQL version 9.5 is now available in the Amazon RDS Database Preview environment. MySQL version 9.5 contains several improvements that are described in Changes in MySQL 9.5.0.

For information on the Database Preview environment, see the section called "Database Preview environment". To access the Preview Environment from the console, select https://console.aws.amazon.com/rds-preview/.

MySQL version 9.4 in the Database Preview environment

MySQL version 9.4 is now available in the Amazon RDS Database Preview environment. MySQL version 9.4 contains several improvements that are described in Changes in MySQL 9.4.0.

For information on the Database Preview environment, see the section called "Database Preview environment". To access the Preview Environment from the console, select https://console.aws.amazon.com/rds-preview/.

MySQL version 9.3 in the Database Preview environment

MySQL version 9.3 is now available in the Amazon RDS Database Preview environment. MySQL version 9.3 contains several improvements that are described in Changes in MySQL 9.3.0.

For information on the Database Preview environment, see the section called "Database Preview environment". To access the Preview Environment from the console, select https://console.aws.amazon.com/rds-preview/.

Deprecated versions for Amazon RDS for MySQL

Amazon RDS for MySQL version 5.1, 5.5, and 5.6 are deprecated.

Amazon RDS for MySQL version 9.1 and 9.2 are deprecated in the Database Preview environment.

For information about the Amazon RDS deprecation policy for MySQL, see Amazon RDS FAQs.

Connecting to your MySQL DB instance

Before you can connect to a DB instance running the MySQL database engine, you must create a DB instance. For information, see Creating an Amazon RDS DB instance. After Amazon RDS provisions your DB instance, you can use any standard MySQL client application or utility to connect to the instance. In the connection string, you specify the DNS address from the DB instance endpoint as the host parameter, and specify the port number from the DB instance endpoint as the port parameter.

To authenticate to your RDS DB instance, you can use one of the authentication methods for MySQL and AWS Identity and Access Management (IAM) database authentication:

- To learn how to authenticate to MySQL using one of the authentication methods for MySQL, see Authentication method in the MySQL documentation.
- To learn how to authenticate to MySQL using IAM database authentication, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.

You can connect to a MySQL DB instance by using tools like the MySQL command-line client. For more information on using the MySQL command-line client, see mysql - the MySQL command-line client in the MySQL documentation. One GUI-based application you can use to connect is MySQL Workbench. For more information, see the Download MySQL Workbench page. For information about installing MySQL (including the MySQL command-line client), see Installing and upgrading MySQL.

To connect to a DB instance from outside of its Amazon VPC, the DB instance must be publicly accessible, access must be granted using the inbound rules of the DB instance's security group, and other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance.

You can use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) encryption on connections to a MySQL DB instance. For information, see SSL/TLS support for MySQL DB instances on Amazon RDS. If you are using AWS Identity and Access Management (IAM) database authentication, make sure to use an SSL/TLS connection. For information, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.

You can also connect to a DB instance from a web server. For more information, see Tutorial: Create a web server and an Amazon RDS DB instance.

> **Note**
>
> For information on connecting to a MariaDB DB instance, see Connecting to your MariaDB DB instance.

To find and connect to a RDS for MySQL DB instance, see the following topics.

Topics

- Finding the connection information for an RDS for MySQL DB instance
- Installing the MySQL command-line client
- Connecting from the MySQL command-line client (unencrypted)
- Connecting from MySQL Workbench
- Connecting to RDS for MySQL with the AWS JDBC Driver, AWS Python Driver, and AWS ODBC Driver for MySQL
- Troubleshooting connections to your MySQL DB instance

Finding the connection information for an RDS for MySQL DB instance

The connection information for a DB instance includes its endpoint, port, and a valid database user, such as the master user. For example, suppose that an endpoint value is `mydb.123456789012.us-east-1.rds.amazonaws.com`. In this case, the port value is `3306`, and the database user is `admin`. Given this information, you specify the following values in a connection string:

- For host or host name or DNS name, specify `mydb.123456789012.us-east-1.rds.amazonaws.com`.
- For port, specify `3306`.
- For user, specify `admin`.

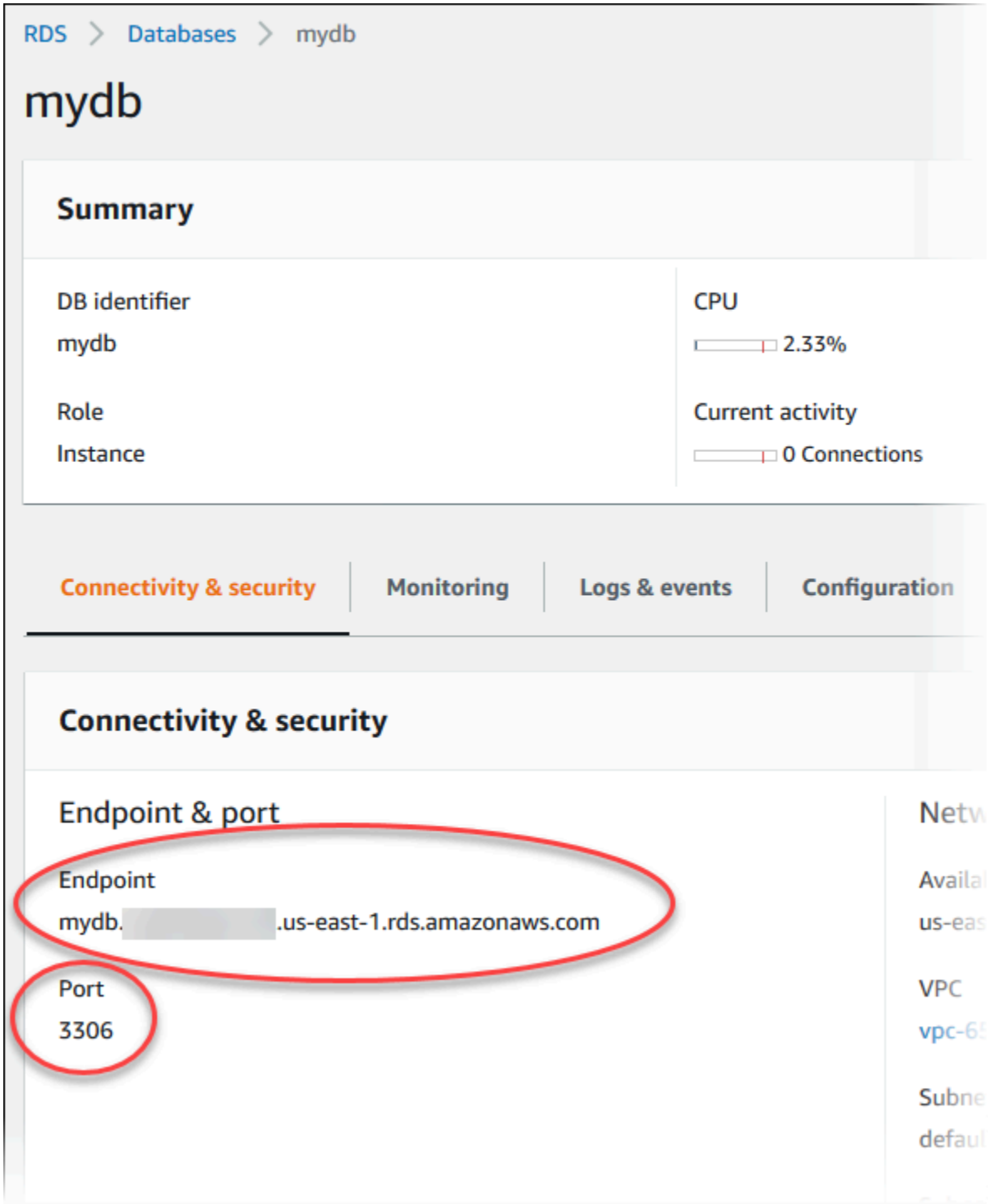
To connect to a DB instance, use any client for the MySQL DB engine. For example, you might use the MySQL command-line client or MySQL Workbench.

To find the connection information for a DB instance, you can use the AWS Management Console, the AWS CLI describe-db-instances command, or the Amazon RDS API DescribeDBInstances operation to list its details.

Console

To find the connection information for a DB instance in the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases** to display a list of your DB instances.
3. Choose the name of the MySQL DB instance to display its details.
4. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



5. If you need to find the master user name, choose the **Configuration** tab and view the **Master username** value.

AWS CLI

To find the connection information for a MySQL DB instance by using the AWS CLI, run the describe-db-instances command. In the call, query for the DB instance ID, endpoint, port, and master user name.

For Linux, macOS, or Unix:

```
aws rds describe-db-instances \
  --filters "Name=engine,Values=mysql" \
  --query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

For Windows:

```
aws rds describe-db-instances ^
  --filters "Name=engine,Values=mysql" ^
  --query "*[].[DBInstanceIdentifier,Endpoint.Address,Endpoint.Port,MasterUsername]"
```

Your output should be similar to the following.

```
[
    [
        "mydb1",
        "mydb1.123456789012.us-east-1.rds.amazonaws.com",
        3306,
        "admin"
    ],
    [
        "mydb2",
        "mydb2.123456789012.us-east-1.rds.amazonaws.com",
        3306,
        "admin"
    ]
]
```

RDS API

To find the connection information for a DB instance by using the Amazon RDS API, call the DescribeDBInstances operation. In the output, find the values for the endpoint address, endpoint port, and master user name.

Installing the MySQL command-line client

Most Linux distributions include the MariaDB client instead of the Oracle MySQL client. To install the MySQL command-line client on Amazon Linux 2023, run the following command:

```
sudo dnf install mariadb105
```

To install the MySQL command-line client on Amazon Linux 2, run the following command:

```
sudo yum install mariadb
```

To install the MySQL command-line client on most DEB-based Linux distributions, run the following command:

```
apt-get install mariadb-client
```

To check the version of your MySQL command-line client, run the following command:

```
mysql --version
```

To read the MySQL documentation for your current client version, run the following command:

```
man mysql
```

Connecting from the MySQL command-line client (unencrypted)

> **Important**
>
> Only use an unencrypted MySQL connection when the client and server are in the same VPC and the network is trusted. For information about using encrypted connections, see Connecting to your MySQL DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted).

To connect to a DB instance using the MySQL command-line client, enter the following command at the command prompt. For the -h parameter, substitute the DNS name (endpoint) for your DB instance. For the -P parameter, substitute the port for your DB instance. For the -u parameter,

substitute the user name of a valid database user, such as the master user. Enter the master user password when prompted.

```
mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com -P 3306 -
u mymasteruser -p
```

After you enter the password for the user, you should see output similar to the following.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9738
Server version: 8.0.28 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Connecting from MySQL Workbench

To connect from MySQL Workbench

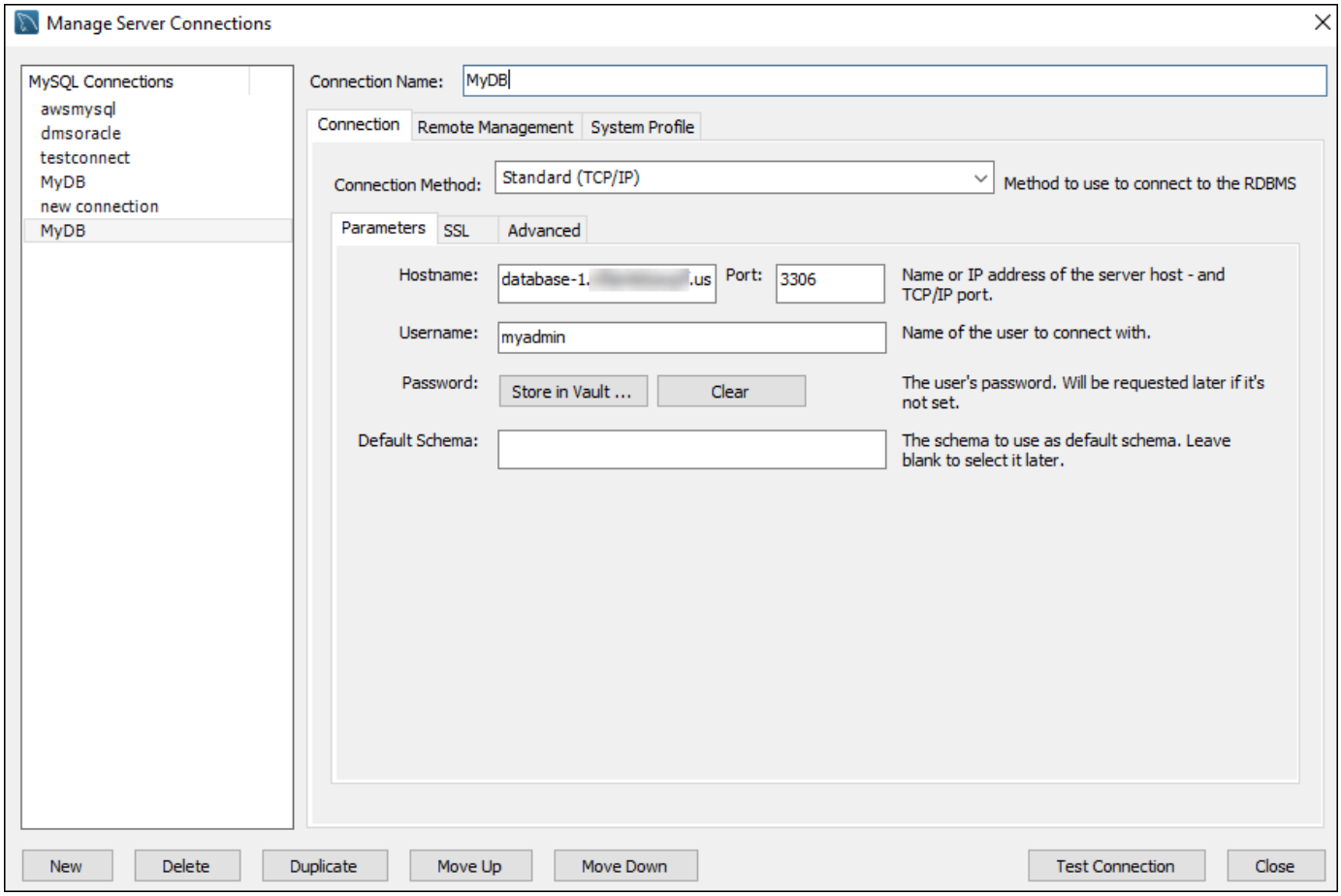
1. Download and install MySQL Workbench at Download MySQL Workbench.
2. Open MySQL Workbench.



3. From **Database**, choose **Manage Connections**.
4. In the **Manage Server Connections** window, choose **New**.
5. In the **Connect to Database** window, enter the following information:

- **Stored Connection** – Enter a name for the connection, such as **MyDB**.
- **Hostname** – Enter the DB instance endpoint.
- **Port** – Enter the port used by the DB instance.
- **Username** – Enter the user name of a valid database user, such as the master user.
- **Password** – Optionally, choose **Store in Vault** and then enter and save the password for the user.

The window looks similar to the following:



You can use the features of MySQL Workbench to customize connections. For example, you can use the **SSL** tab to configure SSL/TLS connections. For information about using MySQL Workbench, see the MySQL Workbench documentation. Encrypting client connections to MySQL DB instances with SSL/TLS, see Encrypting client connections with SSL/TLS to MySQL DB instances on Amazon RDS.

6. Optionally, choose **Test Connection** to confirm that the connection to the DB instance is successful.
7. Choose **Close**.
8. From **Database**, choose **Connect to Database**.
9. From **Stored Connection**, choose your connection.
10. Choose **OK**.

Connecting to RDS for MySQL with the AWS JDBC Driver, AWS Python Driver, and AWS ODBC Driver for MySQL

Connect to RDS for MySQL DB instances with the AWS JDBC Driver, the AWS Python Driver, and the AWS ODBC Driver for MySQL. For more information, see the following topics.

Topics

- Connecting to RDS for MySQL with the Amazon Web Services (AWS) JDBC Driver
- Connecting to RDS for MySQL with the Amazon Web Services (AWS) Python Driver
- Connecting to RDS for MySQL with the Amazon Web Services (AWS) ODBC Driver for MySQL

Connecting to RDS for MySQL with the Amazon Web Services (AWS) JDBC Driver

The Amazon Web Services (AWS) JDBC Driver is designed as an advanced JDBC wrapper. This wrapper is complementary to and extends the functionality of an existing JDBC driver. The driver is drop-in compatible with the community MySQL Connector/J driver and the community MariaDB Connector/J driver.

To install the AWS JDBC Driver, append the AWS JDBC Driver .jar file (located in the application CLASSPATH), and keep references to the respective community driver. Update the respective connection URL prefix as follows:

- `jdbc:mysql://` to `jdbc:aws-wrapper:mysql://`
- `jdbc:mariadb://` to `jdbc:aws-wrapper:mariadb://`

For more information about the AWS JDBC Driver and complete instructions for using it, see the Amazon Web Services (AWS) JDBC Driver GitHub repository.

Connecting to RDS for MySQL with the Amazon Web Services (AWS) Python Driver

The Amazon Web Services (AWS) Python Driver is designed as an advanced Python wrapper. This wrapper is complementary to and extends the functionality of the open-source Psycopg driver. The AWS Python Driver supports Python versions 3.8 and higher. You can install the `aws-advanced-python-wrapper` package using the `pip` command, along with the `psycopg` open-source packages.

For more information about the AWS Python Driver and complete instructions for using it, see the Amazon Web Services (AWS) Python Driver GitHub repository.

Connecting to RDS for MySQL with the Amazon Web Services (AWS) ODBC Driver for MySQL

The AWS ODBC Driver for MySQL is a client driver designed for the high availability of RDS for MySQL. The driver can exist alongside the MySQL Connector/ODBC driver and is compatible with the same workflows.

For more information about the AWS ODBC Driver for MySQL and complete instructions for installing and using it, see the Amazon Web Services (AWS) ODBC Driver for MySQL GitHub repository.

Troubleshooting connections to your MySQL DB instance

Two common causes of connection failures to a new DB instance are:

- The DB instance was created using a security group that doesn't authorize connections from the device or Amazon EC2 instance where the MySQL application or utility is running. The DB instance must have a VPC security group that authorizes the connections. For more information, see Amazon VPC and Amazon RDS.

  You can add or edit an inbound rule in the security group. For **Source**, choose **My IP**. This allows access to the DB instance from the IP address detected in your browser.
- The DB instance was created using the default port of 3306, and your company has firewall rules blocking connections to that port from devices in your company network. To fix this failure, recreate the instance with a different port.

For more information on connection issues, see Can't connect to Amazon RDS DB instance.

Securing MySQL DB instance connections

You can implement robust security measures to protect MySQL DB instances from unauthorized access and potential threats. Security groups, SSL/TLS encryption, and IAM database authentication work together to create multiple layers of connection security for your MySQL DB instances. These security controls help you meet compliance requirements, prevent data breaches, and maintain secure communication channels between applications and databases. You can secure your MySQL DB instances by encrypting data in transit, restricting access to specific IP ranges, and managing user authentication through IAM roles rather than database passwords.

Security for MySQL DB instances is managed at three levels:

- AWS Identity and Access Management controls who can perform Amazon RDS management actions on DB instances. When you connect to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see Identity and access management for Amazon RDS.
- When you create a DB instance, you use a VPC security group to control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance. These connections can be made using Secure Sockets Layer (SSL) and Transport Layer Security (TLS). In addition, firewall rules at your company can control whether devices running at your company can open connections to the DB instance.
- To authenticate login and permissions for a MySQL DB instance, you can take either of the following approaches, or a combination of them:
  - You can take the same approach as with a stand-alone instance of MySQL. Commands such as `CREATE USER`, `RENAME USER`, `GRANT`, `REVOKE`, and `SET PASSWORD` work just as they do in on-premises databases, as does directly modifying database schema tables. However, directly modifying the database schema tables isn't a best practice, and starting from RDS for MySQL version 8.0.36, it isn't supported. For information, see Access control and account management in the MySQL documentation.
  - You can also use IAM database authentication. With IAM database authentication, you authenticate to your DB instance by using an IAM user or IAM role and an authentication token. An *authentication token* is a unique value that is generated using the Signature Version 4 signing process. By using IAM database authentication, you can use the same credentials to control access to your AWS resources and your databases. For more information, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.

- Another option is Kerberos authentication for RDS for MySQL. The DB instance works with AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) to enable Kerberos authentication. When users authenticate with a MySQL DB instance joined to the trusting domain, authentication requests are forwarded. Forwarded requests go to the domain directory that you create with Directory Service. For more information, see Using Kerberos authentication for Amazon RDS for MySQL.

When you create an Amazon RDS DB instance, the master user has the following default privileges:

Engine version	System privilege	Database role
RDS for MySQL version 8.4.3 and higher	`GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES ,INDEX, ALTER, SHOW DATABASES , CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE , REPLICATION CLIENT , CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE ROLE, DROP ROLE, APPLICATION_PASSWORD_ADMIN , FLUSH_OPTIMIZER_COSTS , FLUSH_PRIVILEGES , FLUSH_STATUS , FLUSH_TABLES , FLUSH_USER_RESOURCES , ROLE_ADMIN , SENSITIVE_VARIABLES_OBSERVER , SESSION_VARIABLES_ADMIN , SET_ANY_DEFINER , SHOW_ROUTINE , XA_RECOVER_ADMIN`	`rds_superuser_role` For more information about `rds_superuser_role` , see Role-based privilege model for RDS for MySQL.
RDS for MySQL version 8.0.36 and higher	`SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES , INDEX, ALTER, SHOW DATABASES , CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT , CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE ROLE, DROP ROLE, APPLICATION_PASSWORD_ADMIN ,`	`rds_superuser_role` For more information about `rds_superuser_role` , see Role-based privilege model for RDS for MySQL.

Engine version	System privilege	Database role
	ROLE_ADMIN , SET_USER_ID , XA_RECOVER_ADMIN	
RDS for MySQL versions lower than 8.0.36	SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES , INDEX, ALTER, SHOW DATABASES , CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION CLIENT , CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, REPLICATION SLAVE	None

Note

Although it is possible to delete the master user on the DB instance, it is not recommended. To recreate the master user, use the ModifyDBInstance RDS API operation or run the modify-db-instance AWS CLI command and specify a new master user password with the appropriate parameter. If the master user does not exist in the instance, the master user is created with the specified password.

To provide management services for each DB instance, the `rdsadmin` user is created when the DB instance is created. Attempting to drop, rename, change the password, or change privileges for the `rdsadmin` account will result in an error.

To allow management of the DB instance, the standard `kill` and `kill_query` commands have been restricted. The Amazon RDS commands `rds_kill` and `rds_kill_query` are provided to allow you to end user sessions or queries on DB instances.

Password validation for RDS for MySQL

MySQL provides the `validate_password` plugin for improved security. The plugin enforces password policies using parameters in the DB parameter group for your MySQL DB instance. The plugin is supported for DB instances running MySQL version 5.7, 8.0, and 8.4. For more

information about the `validate_password` plugin, see The Password Validation Plugin in the MySQL documentation.

To enable the `validate_password` plugin for a MySQL DB instance

1. Connect to your MySQL DB instance and run the following command.

```
INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

2. Configure the parameters for the plugin in the DB parameter group used by the DB instance.

   For more information about the parameters, see Password Validation Plugin Options and Variables in the MySQL documentation.

   For more information about modifying DB instance parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

3. Restart the DB instance.

After enabling the `validate_password` plugin, reset existing passwords to comply with your new validation policies.

Your MySQL DB instance handles password validation for Amazon RDS. To change a password, you first submit a password update request through the AWS Management Console, `modify-db-instance` CLI command, or `ModifyDBInstance` API operation. RDS initially accepts your request, even if the password doesn't meet your policies. RDS then processes the request asynchronously. It updates the password in your MySQL DB instance only if the password meets your defined policies. If the password doesn't meet these policies, RDS keeps the existing password and logs an error event.

```
   Unable to reset your password. Error information: Password failed to meet
validation rules.
```

For more information about Amazon RDS events, see Working with Amazon RDS event notification.

Encrypting client connections with SSL/TLS to MySQL DB instances on Amazon RDS

Secure Sockets Layer (SSL) is an industry-standard protocol for securing network connections between client and server. After SSL version 3.0, the name was changed to Transport Layer Security (TLS). Amazon RDS supports SSL/TLS encryption for MySQL DB instances. Using SSL/TLS, you can encrypt a connection between your application client and your MySQL DB instance. SSL/TLS support is available in all AWS Regions for MySQL.

With Amazon RDS, you can secure data in transit by encrypting client connections to MySQL DB instances with SSL/TLS, requiring SSL/TLS for all connections to a MySQL DB instance, and connecting from the MySQL command-line client with SSL/TLS (encrypted). The following sections provide guidance on configuring and utilizing SSL/TLS encryption for MySQL DB instances on Amazon RDS.

Topics

- SSL/TLS support for MySQL DB instances on Amazon RDS
- Requiring SSL/TLS for specific user accounts to a MySQL DB instance on Amazon RDS
- Requiring SSL/TLS for all connections to a MySQL DB instance on Amazon RDS
- Connecting to your MySQL DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted)

SSL/TLS support for MySQL DB instances on Amazon RDS

Amazon RDS creates an SSL/TLS certificate and installs the certificate on the DB instance when Amazon RDS provisions the instance. These certificates are signed by a certificate authority. The SSL/TLS certificate includes the DB instance endpoint as the Common Name (CN) for the SSL/TLS certificate to guard against spoofing attacks.

An SSL/TLS certificate created by Amazon RDS is the trusted root entity and should work in most cases, but might fail if your application doesn't accept certificate chains. If your application doesn't accept certificate chains, try using an intermediate certificate to connect to your AWS Region. For example, you must use an intermediate certificate to connect to the AWS GovCloud (US) Regions with SSL/TLS.

For information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . For more information about using SSL/TLS with MySQL, see Updating applications to connect to MySQL DB instances using new SSL/TLS certificates.

For MySQL version 8.0 and lower, Amazon RDS for MySQL uses OpenSSL for secure connections. For MySQL version 8.4 and higher, Amazon RDS for MySQL uses AWS-LC. TLS support depends on the MySQL version. The following table shows the TLS support for MySQL versions.

MySQL version	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
MySQL 8.4	Not supported	Not supported	Supported	Supported
MySQL 8.0	Not supported	Not supported	Supported	Supported
MySQL 5.7	Supported	Supported	Supported	Not supported

Requiring SSL/TLS for specific user accounts to a MySQL DB instance on Amazon RDS

You can require SSL/TLS encryption for specified user account connections to your MySQL DB instances on Amazon RDS. Protecting sensitive information from unauthorized access or interception is crucial to enforce security policies where data confidentiality is a concern.

To require SSL/TLS connections for specific users' accounts, use one of the following statements, depending on your MySQL version, to require SSL/TLS connections on the user account `encrypted_user`.

To do so, use the following statement.

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

For more information on SSL/TLS connections with MySQL, see the Using encrypted connections in the MySQL documentation.

Requiring SSL/TLS for all connections to a MySQL DB instance on Amazon RDS

Use the `require_secure_transport` parameter to require that all user connections to your MySQL DB instance use SSL/TLS. By default, the `require_secure_transport` parameter is set

to OFF. You can set the `require_secure_transport` parameter to ON to require SSL/TLS for connections to your DB instance.

You can set the `require_secure_transport` parameter value by updating the DB parameter group for your DB instance. You don't need to reboot your DB instance for the change to take effect.

When the `require_secure_transport` parameter is set to ON for a DB instance, a database client can connect to it if it can establish an encrypted connection. Otherwise, an error message similar to the following is returned to the client:

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --
require_secure_transport=ON.
```

For information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

For more information about the `require_secure_transport` parameter, see the MySQL documentation.

Connecting to your MySQL DB instance on Amazon RDS with SSL/TLS from the MySQL command-line client (encrypted)

The `mysql` client program parameters are slightly different depending on which version of MySQL or MariaDB you are using.

To find out which version you have, run the `mysql` command with the `--version` option. In the following example, the output shows that the client program is from MariaDB.

```
$ mysql --version
mysql  Ver 15.1 Distrib 10.5.15-MariaDB, for osx10.15 (x86_64) using readline 5.1
```

Most Linux distributions, such as Amazon Linux, CentOS, SUSE, and Debian have replaced MySQL with MariaDB, and the `mysql` version in them is from MariaDB.

To connect to your DB instance using SSL/TLS, follow these steps:

To connect to a DB instance with SSL/TLS using the MySQL command-line client

1. Download a root certificate that works for all AWS Regions.

For information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

2. Use a MySQL command-line client to connect to a DB instance with SSL/TLS encryption. For the `-h` parameter, substitute the DNS name (endpoint) for your DB instance. For the `--ssl-ca` parameter, substitute the SSL/TLS certificate file name. For the `-P` parameter, substitute the port for your DB instance. For the `-u` parameter, substitute the user name of a valid database user, such as the master user. Enter the master user password when prompted.

   The following example shows how to launch the client using the `--ssl-ca` parameter using the MySQL 5.7 client or later:

   ```
   mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-bundle.pem --ssl-mode=REQUIRED -P 3306 -u myadmin -p
   ```

   To require that the SSL/TLS connection verifies the DB instance endpoint against the endpoint in the SSL/TLS certificate, enter the following command:

   ```
   mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-bundle.pem --ssl-mode=VERIFY_IDENTITY -P 3306 -u myadmin -p
   ```

   The following example shows how to launch the client using the `--ssl-ca` parameter using the MariaDB client:

   ```
   mysql -h mysql-instance1.123456789012.us-east-1.rds.amazonaws.com --ssl-ca=global-bundle.pem --ssl -P 3306 -u myadmin -p
   ```

3. Enter the master user password when prompted.

You will see output similar to the following.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9738
Server version: 8.0.28 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Updating applications to connect to MySQL DB instances using new SSL/TLS certificates

As of January 13, 2023, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your RDS DB instances using Secure Socket Layer or Transport Layer Security (SSL/TLS). Following, you can find information about updating your applications to use the new certificates.

This topic can help you to determine whether any client applications use SSL/TLS to connect to your DB instances. If they do, you can further check whether those applications require certificate verification to connect.

> **Note**
>
> Some applications are configured to connect to MySQL DB instances only if they can successfully verify the certificate on the server. For such applications, you must update your client application trust stores to include the new CA certificates.
> You can specify the following SSL modes: `disabled`, `preferred`, and `required`. When you use the `preferred` SSL mode and the CA certificate doesn't exist or isn't up to date, the connection falls back to not using SSL and connects without encryption.
> We recommend avoiding `preferred` mode. In `preferred` mode, if the connection encounters an invalid certificate, it stops using encryption and proceeds unencrypted.

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB instances. We strongly recommend testing these procedures in a development or staging environment before implementing them in your production environments.

For more information about certificate rotation, see Rotating your SSL/TLS certificate. For more information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . For information about using SSL/TLS with MySQL DB instances, see SSL/TLS support for MySQL DB instances on Amazon RDS.

Topics

- Determining whether any applications are connecting to your MySQL DB instance using SSL
- Determining whether a client requires certificate verification to connect
- Updating your application trust store

- Example Java code for establishing SSL connections

Determining whether any applications are connecting to your MySQL DB instance using SSL

If you are using Amazon RDS for MySQL version 5.7, 8.0, or 8.4 and the Performance Schema is enabled, run the following query to check if connections are using SSL/TLS. For information about enabling the Performance Schema, see Performance Schema quick start in the MySQL documentation.

```
mysql> SELECT id, user, host, connection_type
       FROM performance_schema.threads pst
       INNER JOIN information_schema.processlist isp
       ON pst.processlist_id = isp.id;
```

In this sample output, you can see both your own session (`admin`) and an application logged in as `webapp1` are using SSL.

```
+----+-----------------+------------------+-----------------+
| id | user            | host             | connection_type |
+----+-----------------+------------------+-----------------+
|  8 | admin           | 10.0.4.249:42590 | SSL/TLS         |
|  4 | event_scheduler | localhost        | NULL            |
| 10 | webapp1         | 159.28.1.1:42189 | SSL/TLS         |
+----+-----------------+------------------+-----------------+
3 rows in set (0.00 sec)
```

Determining whether a client requires certificate verification to connect

You can check whether JDBC clients and MySQL clients require certificate verification to connect.

JDBC

The following example with MySQL Connector/J 8.0 shows one way to check an application's JDBC connection properties to determine whether successful connections require a valid certificate. For

more information on all of the JDBC connection options for MySQL, see Configuration properties in the MySQL documentation.

When using the MySQL Connector/J 8.0, an SSL connection requires verification against the DB server certificate if your connection properties have `sslMode` set to `VERIFY_CA` or `VERIFY_IDENTITY`, as in the following example.

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

Note

If you use either the MySQL Java Connector v5.1.38 or later, or the MySQL Java Connector v8.0.9 or later to connect to your databases, even if you haven't explicitly configured your applications to use SSL/TLS when connecting to your databases, these client drivers default to using SSL/TLS. In addition, when using SSL/TLS, they perform partial certificate verification and fail to connect if the database server certificate is expired.

MySQL

The following examples with the MySQL Client show two ways to check a script's MySQL connection to determine whether successful connections require a valid certificate. For more information on all of the connection options with the MySQL Client, see Client-side configuration for encrypted connections in the MySQL documentation.

When using the MySQL Client version 5.7 and higher, an SSL connection requires verification against the server CA certificate if for the `--ssl-mode` option you specify `VERIFY_CA` or `VERIFY_IDENTITY`, as in the following example.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

Updating your application trust store

For information about updating the trust store for MySQL applications, see Installing SSL certificates in the MySQL documentation.

For information about downloading the root certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

For sample scripts that import certificates, see Sample script for importing certificates into your trust store.

> **Note**
>
> When you update the trust store, you can retain older certificates in addition to adding the new certificates.

If you are using the mysql JDBC driver in an application, set the following properties in the application.

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

When you start the application, set the following properties.

```
java -Djavax.net.ssl.trustStore=/path_to_trust_store/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_trust_store_password com.companyName.MyApplication
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Example Java code for establishing SSL connections

The following code example shows how to set up the SSL connection that validates the server certificate using JDBC.

```
public class MySQLSSLTest {

        private static final String DB_USER = "username";
        private static final String DB_PASSWORD = "password";
        // This trust store has only the prod root ca.
        private static final String TRUST_STORE_FILE_PATH = "file-path-to-trust-store";
        private static final String TRUST_STORE_PASS = "trust-store-password";

        public static void test(String[] args) throws Exception {
            Class.forName("com.mysql.jdbc.Driver");

            System.setProperty("javax.net.ssl.trustStore", TRUST_STORE_FILE_PATH);
            System.setProperty("javax.net.ssl.trustStorePassword", TRUST_STORE_PASS);

            Properties properties = new Properties();
            properties.setProperty("sslMode", "VERIFY_IDENTITY");
            properties.put("user", DB_USER);
            properties.put("password", DB_PASSWORD);


            Connection connection = null;
            Statement stmt = null;
            ResultSet rs = null;
            try {
                connection =
 DriverManager.getConnection("jdbc:mysql://mydatabase.123456789012.us-
east-1.rds.amazonaws.com:3306",properties);
                stmt = connection.createStatement();
                rs=stmt.executeQuery("SELECT 1 from dual");
            } finally {
                if (rs != null) {
                    try {
                        rs.close();
                    } catch (SQLException e) {
                    }
                }
                if (stmt != null) {
```

```
                try {
                     stmt.close();
                 } catch (SQLException e) {
                }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
        return;
    }
}
```

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-rsa2048-g1 certificates. For instructions, see step 3 in Updating your CA certificate by modifying your DB instance or cluster.
Specify a password other than the prompt shown here as a security best practice.

Using Kerberos authentication for Amazon RDS for MySQL

You can use Kerberos authentication to authenticate users when they connect to your MySQL DB instance. The DB instance works with AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) to enable Kerberos authentication. When users authenticate with a MySQL DB instance joined to the trusting domain, authentication requests are forwarded. Forwarded requests go to the domain directory that you create with Directory Service.

Keeping all of your credentials in the same directory can save you time and effort. With this approach, you have a centralized place for storing and managing credentials for multiple DB instances. Using a directory can also improve your overall security profile.

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability of Amazon RDS with Kerberos authentication, see Supported Regions and DB engines for Kerberos authentication in Amazon RDS.

Overview of Setting up Kerberos authentication for MySQL DB instances

To set up Kerberos authentication for a MySQL DB instance, complete the following general steps, described in more detail later:

1. Use AWS Managed Microsoft AD to create an AWS Managed Microsoft AD directory. You can use the AWS Management Console, the AWS CLI, or the Directory Service to create the directory. For details about doing so, see Create your AWS Managed Microsoft AD directory in the *AWS Directory Service Administration Guide*.
2. Create an AWS Identity and Access Management (IAM) role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`. The role allows Amazon RDS to make calls to your directory.

   For the role to allow access, the AWS Security Token Service (AWS STS) endpoint must be activated in the AWS Region for your AWS account. AWS STS endpoints are active by default in all AWS Regions, and you can use them without any further actions. For more information, see Activating and deactivating AWS STS in an AWS Region in the *IAM User Guide*.
3. Create and configure users in the AWS Managed Microsoft AD directory using the Microsoft Active Directory tools. For more information about creating users in your Active Directory, see Manage users and groups in AWS managed Microsoft AD in the *AWS Directory Service Administration Guide*.
4. Create or modify a MySQL DB instance. If you use either the CLI or RDS API in the create request, specify a domain identifier with the `Domain` parameter. Use the `d-*` identifier that was generated when you created your directory and the name of the role that you created.

   If you modify an existing MySQL DB instance to use Kerberos authentication, set the domain and IAM role parameters for the DB instance. Locate the DB instance in the same VPC as the domain directory.
5. Use the Amazon RDS master user credentials to connect to the MySQL DB instance. Create the user in MySQL using the `CREATE USER` clause `IDENTIFIED WITH 'auth_pam'`. Users that you create this way can log in to the MySQL DB instance using Kerberos authentication.

Setting up Kerberos authentication for MySQL DB instances

You use AWS Managed Microsoft AD to set up Kerberos authentication for a MySQL DB instance. To set up Kerberos authentication, you take the following steps.

Step 1: Create a directory using AWS Managed Microsoft AD

Directory Service creates a fully managed Active Directory in the AWS Cloud. When you create an AWS Managed Microsoft AD directory, Directory Service creates two domain controllers and Domain Name System (DNS) servers on your behalf. The directory servers are created in different subnets in a VPC. This redundancy helps make sure that your directory remains accessible even if a failure occurs.

When you create an AWS Managed Microsoft AD directory, Directory Service performs the following tasks on your behalf:

- Sets up an Active Directory within the VPC.
- Creates a directory administrator account with the user name Admin and the specified password. You use this account to manage your directory.

  > **Note**
  >
  > Be sure to save this password. Directory Service doesn't store it. You can reset it, but you can't retrieve it.

- Creates a security group for the directory controllers.

When you launch an AWS Managed Microsoft AD, AWS creates an Organizational Unit (OU) that contains all of your directory's objects. This OU has the NetBIOS name that you typed when you created your directory and is located in the domain root. The domain root is owned and managed by AWS.

The Admin account that was created with your AWS Managed Microsoft AD directory has permissions for the most common administrative activities for your OU:

- Create, update, or delete users
- Add resources to your domain such as file or print servers, and then assign permissions for those resources to users in your OU
- Create additional OUs and containers

- Delegate authority
- Restore deleted objects from the Active Directory Recycle Bin
- Run AD and DNS Windows PowerShell modules on the Active Directory Web Service

The Admin account also has rights to perform the following domain-wide activities:

- Manage DNS configurations (add, remove, or update records, zones, and forwarders)
- View DNS event logs
- View security event logs

To create a directory with AWS Managed Microsoft AD

1. Sign in to the AWS Management Console and open the Directory Service console at https://console.aws.amazon.com/directoryservicev2/.
2. In the navigation pane, choose **Directories** and choose **Set up Directory**.
3. Choose **AWS Managed Microsoft AD**. AWS Managed Microsoft AD is the only option that you can currently use with Amazon RDS.
4. Enter the following information:

   Directory DNS name

   The fully qualified name for the directory, such as **`corp.example.com`**.

   Directory NetBIOS name

   The short name for the directory, such as **`CORP`**.

   Directory description

   (Optional) A description for the directory.

   Admin password

   The password for the directory administrator. The directory creation process creates an administrator account with the user name Admin and this password.

   The directory administrator password and can't include the word "admin." The password is case-sensitive and must be 8–64 characters in length. It must also contain at least one character from three of the following four categories:

- Lowercase letters (a–z)
- Uppercase letters (A–Z)
- Numbers (0–9)
- Non-alphanumeric characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Confirm password

The administrator password retyped.

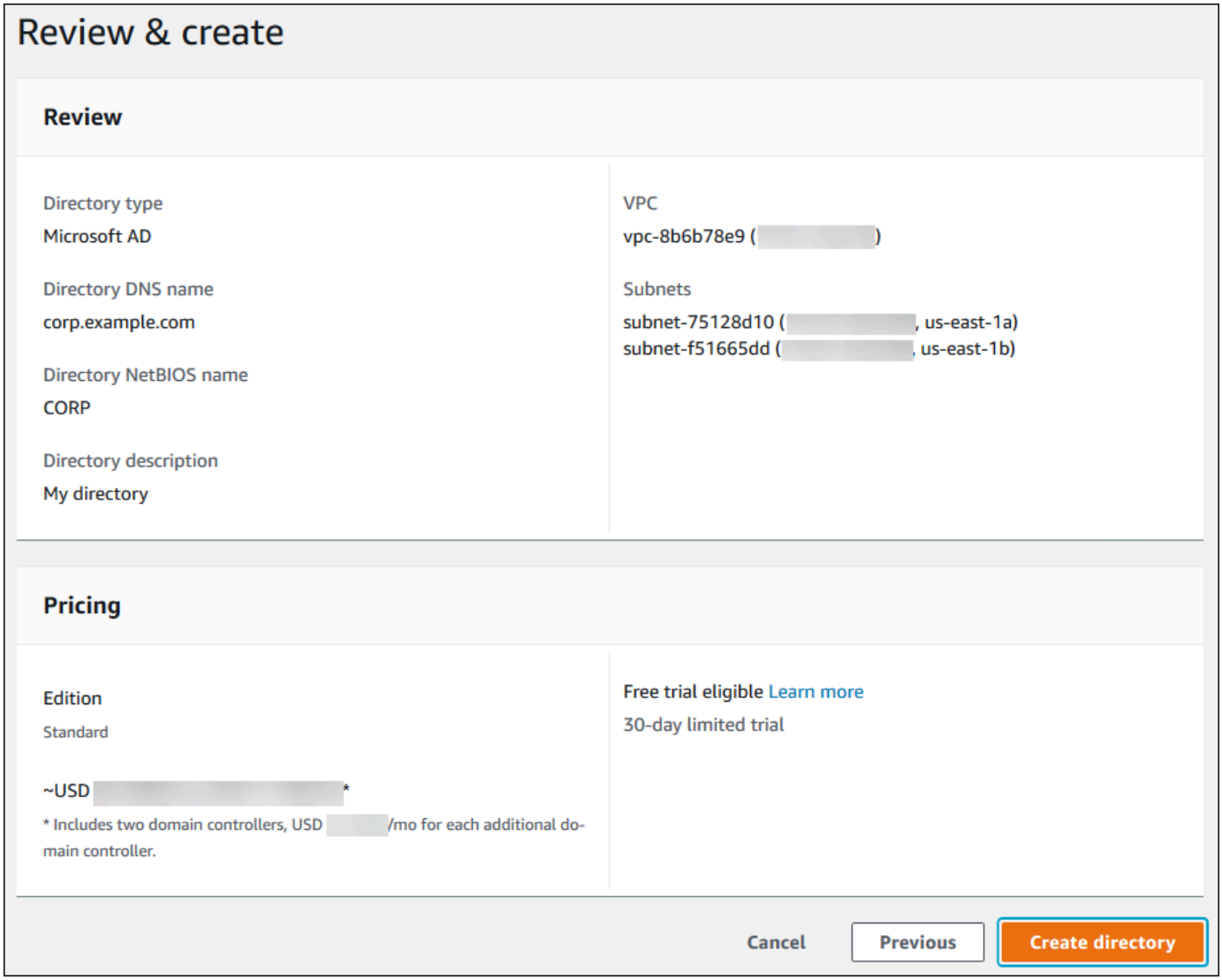
5. Choose **Next**.
6. Enter the following information in the **Networking** section and then choose **Next**:

   VPC

   The VPC for the directory. Create the MySQL DB instance in this same VPC.

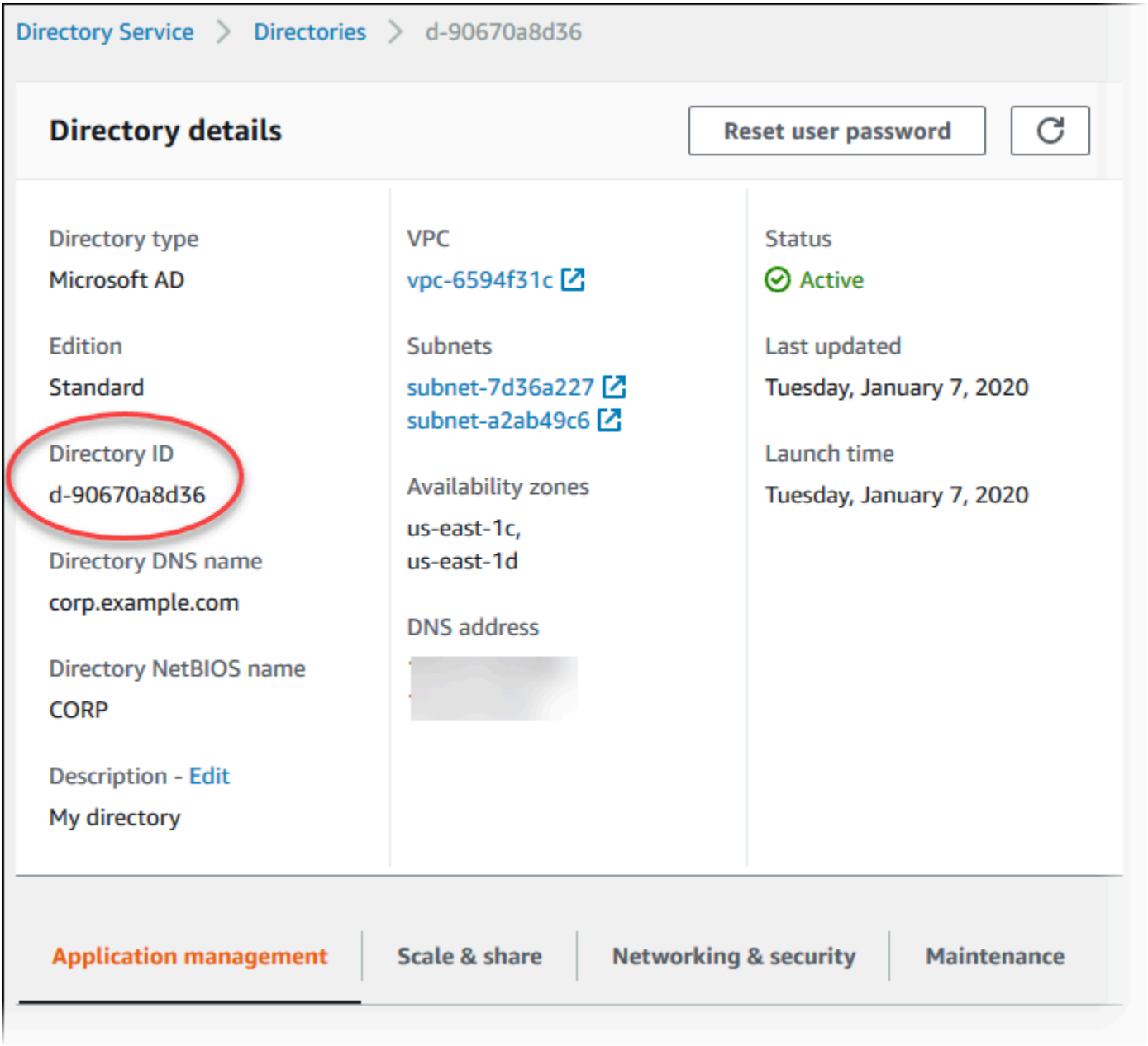
   Subnets

   Subnets for the directory servers. The two subnets must be in different Availability Zones.
7. Review the directory information and make any necessary changes. When the information is correct, choose **Create directory**.



It takes several minutes for the directory to be created. When it has been successfully created, the **Status** value changes to **Active**.

To see information about your directory, choose the directory name in the directory listing. Note the **Directory ID** value because you need this value when you create or modify your MySQL DB instance.



Step 2: Create the IAM role for use by Amazon RDS

For Amazon RDS to call Directory Service for you, an IAM role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess` is required. This role allows Amazon RDS to make calls to the Directory Service.

When a DB instance is created using the AWS Management Console and the console user has the `iam:CreateRole` permission, the console creates this role automatically. In this case, the role name is `rds-directoryservice-kerberos-access-role`. Otherwise, you must create the

IAM role manually. When you create this IAM role, choose `Directory Service`, and attach the AWS managed policy `AmazonRDSDirectoryServiceAccess` to it.

For more information about creating IAM roles for a service, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Note**
>
> The IAM role used for Windows Authentication for RDS for SQL Server can't be used for RDS for MySQL.

Optionally, you can create policies with the required permissions instead of using the managed IAM policy `AmazonRDSDirectoryServiceAccess`. In this case, the IAM role must have the following IAM trust policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The role must also have the following IAM role policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

Step 3: Create and configure users

You can create users with the Active Directory Users and Computers tool. This tool is part of the Active Directory Domain Services and Active Directory Lightweight Directory Services tools. Users represent individual people or entities that have access to your directory.

To create users in an Directory Service directory, you must be connected to an Amazon EC2 instance based on Microsoft Windows. This instance must be a member of the Directory Service directory and be logged in as a user that has privileges to create users. For more information, see Manage users and groups in AWS Managed Microsoft AD in the *AWS Directory Service Administration Guide*.

Step 4: Create or modify a MySQL DB instance

Create or modify a MySQL DB instance for use with your directory. You can use the console, CLI, or RDS API to associate a DB instance with a directory. You can do this in one of the following ways:

- Create a new MySQL DB instance using the console, the create-db-instance CLI command, or the CreateDBInstance RDS API operation.

  For instructions, see Creating an Amazon RDS DB instance.

- Modify an existing MySQL DB instance using the console, the modify-db-instance CLI command, or the ModifyDBInstance RDS API operation.

  For instructions, see Modifying an Amazon RDS DB instance.

- Restore a MySQL DB instance from a DB snapshot using the console, the restore-db-instance-from-db-snapshot CLI command, or the RestoreDBInstanceFromDBSnapshot RDS API operation.

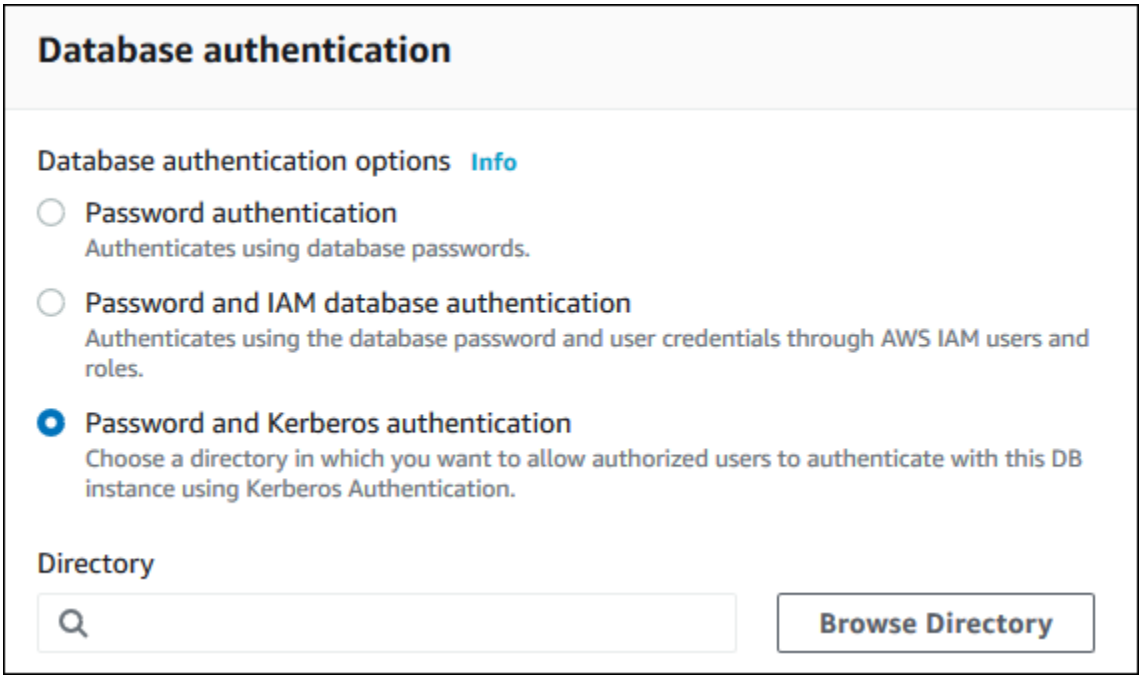
  For instructions, see Restoring to a DB instance.

- Restore a MySQL DB instance to a point-in-time using the console, the restore-db-instance-to-point-in-time CLI command, or the RestoreDBInstanceToPointInTime RDS API operation.

  For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

Kerberos authentication is only supported for MySQL DB instances in a VPC. The DB instance can be in the same VPC as the directory, or in a different VPC. The DB instance must use a security group that allows egress within the directory's VPC so the DB instance can communicate with the directory.

When you use the console to create, modify, or restore a DB instance, choose **Password and Kerberos authentication** in the **Database authentication** section. Choose **Browse Directory** and then select the directory, or choose **Create a new directory**.



When you use the AWS CLI or RDS API, associate a DB instance with a directory. The following parameters are required for the DB instance to use the domain directory you created:

- For the `--domain` parameter, use the domain identifier ("d-*" identifier) generated when you created the directory.
- For the `--domain-iam-role-name` parameter, use the role you created that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

For example, the following CLI command modifies a DB instance to use a directory.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --domain d-ID \
    --domain-iam-role-name role-name
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

Important

If you modify a DB instance to enable Kerberos authentication, reboot the DB instance after making the change.

Step 5: Create Kerberos authentication MySQL logins

Use the Amazon RDS master user credentials to connect to the MySQL DB instance as you do any other DB instance. The DB instance is joined to the AWS Managed Microsoft AD domain. Thus, you can provision MySQL logins and users from the Active Directory users in your domain. Database permissions are managed through standard MySQL permissions that are granted to and revoked from these logins.

You can allow an Active Directory user to authenticate with MySQL. To do this, first use the Amazon RDS master user credentials to connect to the MySQL DB instance as with any other DB instance. After you're logged in, create an externally authenticated user with PAM (Pluggable Authentication Modules) in MySQL by running the following command. Replace *testuser* with the user name.

```
CREATE USER 'testuser'@'%' IDENTIFIED WITH 'auth_pam';
```

Users (both humans and applications) from your domain can now connect to the DB instance from a domain joined client machine using Kerberos authentication.

Important

We strongly recommended that clients use SSL/TLS connections when using PAM authentication. If they don't use SSL/TLS connections, the password might be sent as clear text in some cases. To require an SSL/TLS encrypted connection for your AD user, run the following command and replace *testuser* with the user name:

```
ALTER USER 'testuser'@'%' REQUIRE SSL;
```

For more information, see SSL/TLS support for MySQL DB instances on Amazon RDS.

Managing a DB instance in a domain

You can use the CLI or the RDS API to manage your DB instance and its relationship with your managed Active Directory. For example, you can associate an Active Directory for Kerberos authentication and disassociate an Active Directory to disable Kerberos authentication. You can also move a DB instance to be externally authenticated by one Active Directory to another.

For example, using the Amazon RDS API, you can do the following:

- To reattempt enabling Kerberos authentication for a failed membership, use the `ModifyDBInstance` API operation and specify the current membership's directory ID.
- To update the IAM role name for membership, use the `ModifyDBInstance` API operation and specify the current membership's directory ID and the new IAM role.
- To disable Kerberos authentication on a DB instance, use the `ModifyDBInstance` API operation and specify none as the domain parameter.

- To move a DB instance from one domain to another, use the `ModifyDBInstance` API operation and specify the domain identifier of the new domain as the domain parameter.
- To list membership for each DB instance, use the `DescribeDBInstances` API operation.

Understanding domain membership

After you create or modify your DB instance, it becomes a member of the domain. You can view the status of the domain membership for the DB instance by running the describe-db-instances CLI command. The status of the DB instance can be one of the following:

- `kerberos-enabled` – The DB instance has Kerberos authentication enabled.
- `enabling-kerberos` – AWS is in the process of enabling Kerberos authentication on this DB instance.
- `pending-enable-kerberos` – The enabling of Kerberos authentication is pending on this DB instance.
- `pending-maintenance-enable-kerberos` – AWS will attempt to enable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `pending-disable-kerberos` – The disabling of Kerberos authentication is pending on this DB instance.
- `pending-maintenance-disable-kerberos` – AWS will attempt to disable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `enable-kerberos-failed` – A configuration problem has prevented AWS from enabling Kerberos authentication on the DB instance. Check and fix your configuration before reissuing the DB instance modify command.
- `disabling-kerberos` – AWS is in the process of disabling Kerberos authentication on this DB instance.

A request to enable Kerberos authentication can fail because of a network connectivity issue or an incorrect IAM role. For example, suppose that you create a DB instance or modify an existing DB instance and the attempt to enable Kerberos authentication fails. If this happens, re-issue the modify command or modify the newly created DB instance to join the domain.

Connecting to MySQL with Kerberos authentication

To connect to MySQL with Kerberos authentication, you must log in using the Kerberos authentication type.

To create a database user that you can connect to using Kerberos authentication, use an `IDENTIFIED WITH` clause on the `CREATE USER` statement. For instructions, see Step 5: Create Kerberos authentication MySQL logins.

To avoid errors, use the MariaDB `mysql` client. You can download MariaDB software at https://downloads.mariadb.org/.

At a command prompt, connect to one of the endpoints associated with your MySQL DB instance. Follow the general procedures in Connecting to your MySQL DB instance. When you're prompted for the password, enter the Kerberos password associated with that user name.

Restoring a MySQL DB instance and adding it to a domain

You can restore a DB snapshot or complete a point-in-time restore for a MySQL DB instance and then add it to a domain. After the DB instance is restored, modify the DB instance using the process explained in Step 4: Create or modify a MySQL DB instance to add the DB instance to a domain.

Kerberos authentication MySQL limitations

The following limitations apply to Kerberos authentication for MySQL:

- Only an AWS Managed Microsoft AD is supported. However, you can join RDS for MySQL DB instances to shared Managed Microsoft AD domains owned by different accounts in the same AWS Region.
- You must reboot the DB instance after enabling the feature.
- The domain name length can't be longer than 61 characters.
- You can't enable Kerberos authentication and IAM authentication at the same time. Choose one authentication method or the other for your MySQL DB instance.
- Don't modify the DB instance port after enabling the feature.
- Don't use Kerberos authentication with read replicas.
- If you have auto minor version upgrade turned on for a MySQL DB instance that is using Kerberos authentication, you must turn off Kerberos authentication and then turn it back on after an automatic upgrade. For more information about auto minor version upgrades, see Automatic minor version upgrades for RDS for MySQL.
- To delete a DB instance with this feature enabled, first disable the feature. To do so, run the `modify-db-instance` CLI command for the DB instance and specify `none` for the `--domain` parameter.

If you use the CLI or RDS API to delete a DB instance with this feature enabled, expect a delay.

- RDS for MySQL doesn't support Kerberos authentication across a forest trust between your on-premise or self-hosted AD and the AWS Managed Microsoft AD.

Improving query performance for RDS for MySQL with Amazon RDS Optimized Reads

You can achieve faster query processing for RDS for MySQL with Amazon RDS Optimized Reads. An RDS for MySQL DB instance or Multi-AZ DB cluster that uses RDS Optimized Reads can achieve up to 2x faster query processing compared to a DB instance or cluster that doesn't use it.

Topics

- Overview of RDS Optimized Reads
- Use cases for RDS Optimized Reads
- Best practices for RDS Optimized Reads
- Using RDS Optimized Reads
- Monitoring DB instances that use RDS Optimized Reads
- Limitations for RDS Optimized Reads

Overview of RDS Optimized Reads

When you use an RDS for MySQL DB instance or Multi-AZ DB cluster that has RDS Optimized Reads turned on, it achieves faster query performance through the use of an instance store. An *instance store* provides temporary block-level storage for your DB instance or Multi-AZ DB cluster. The storage is located on Non-Volatile Memory Express (NVMe) solid state drives (SSDs) that are physically attached to the host server. This storage is optimized for low latency, high random I/O performance, and high sequential read throughput.

RDS Optimized Reads is turned on by default when a DB instance or Multi-AZ DB cluster uses a DB instance class with an instance store, such as db.m5d or db.m6gd. With RDS Optimized Reads, some temporary objects are stored on the instance store. These temporary objects include internal temporary files, internal on-disk temp tables, memory map files, and binary log (binlog) cache files. For more information about the instance store, see Amazon EC2 instance store in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

The workloads that generate temporary objects in MySQL for query processing can take advantage of the instance store for faster query processing. This type of workload includes queries involving sorts, hash aggregations, high-load joins, Common Table Expressions (CTEs), and queries on unindexed columns. These instance store volumes provide higher IOPS and performance,

regardless of the storage configurations used for persistent Amazon EBS storage. Because RDS Optimized Reads offloads operations on temporary objects to the instance store, the input/output operations per second (IOPS) or throughput of the persistent storage (Amazon EBS) can now be used for operations on persistent objects. These operations include regular data file reads and writes, and background engine operations, such as flushing and insert buffer merges.

> **Note**
>
> Both manual and automated RDS snapshots only contain engine files for persistent objects. The temporary objects created in the instance store aren't included in RDS snapshots.

Use cases for RDS Optimized Reads

If you have workloads that rely heavily on temporary objects, such as internal tables or files, for their query execution, then you can benefit from turning on RDS Optimized Reads. The following use cases are candidates for RDS Optimized Reads:

- Applications that run analytical queries with complex common table expressions (CTEs), derived tables, and grouping operations
- Read replicas that serve heavy read traffic with unoptimized queries
- Applications that run on-demand or dynamic reporting queries that involve complex operations, such as queries with `GROUP BY` and `ORDER BY` clauses
- Workloads that use internal temporary tables for query processing

  You can monitor the engine status variable `created_tmp_disk_tables` to determine the number of disk-based temporary tables created on your DB instance.
- Applications that create large temporary tables, either directly or in procedures, to store intermediate results
- Database queries that perform grouping or ordering on non-indexed columns

Best practices for RDS Optimized Reads

Use the following best practices for RDS Optimized Reads:

- Add retry logic for read-only queries in case they fail because the instance store is full during the execution.

- Monitor the storage space available on the instance store with the CloudWatch metric `FreeLocalStorage`. If the instance store is reaching its limit because of workload on the DB instance, modify the DB instance to use a larger DB instance class.
- When your DB instance or Multi-AZ DB cluster has sufficient memory but is still reaching the storage limit on the instance store, increase the `binlog_cache_size` value to maintain the session-specific binlog entries in memory. This configuration prevents writing the binlog entries to temporary binlog cache files on disk.

  The `binlog_cache_size` parameter is session-specific. You can change the value for each new session. The setting for this parameter can increase the memory utilization on the DB instance during peak workload. Therefore, consider increasing the parameter value based on the workload pattern of your application and available memory on the DB instance.
- For MySQL 8.0 versions and lower, use the default value of `MIXED` for the `binlog_format` parameter. Depending on the size of the transactions, setting `binlog_format` to `ROW` can result in large binlog cache files on the instance store. For MySQL 8.4 and higher, use the default value of `ROW` for the `binlog_format` parameter.
- Set the internal_tmp_mem_storage_engine parameter to `TempTable`, and set the temptable_max_mmap parameter to match the size of the available storage on the instance store.
- Avoid performing bulk changes in a single transaction. These types of transactions can generate large binlog cache files on the instance store and can cause issues when the instance store is full. Consider splitting writes into multiple small transactions to minimize storage use for binlog cache files.
- Use the default value of `ABORT_SERVER` for the `binlog_error_action` parameter. Doing so avoids issues with the binary logging on DB instances with backups enabled.

Using RDS Optimized Reads

When you provision an RDS for MySQL DB instance with one of the following DB instance classes in a Single-AZ DB instance deployment, Multi-AZ DB instance deployment, or Multi-AZ DB cluster deployment, the DB instance automatically uses RDS Optimized Reads.

To turn on RDS Optimized Reads, do one of the following:

- Create an RDS for MySQL DB instance or Multi-AZ DB cluster using one of these DB instance classes. For more information, see Creating an Amazon RDS DB instance.

- Modify an existing RDS for MySQL DB instance or Multi-AZ DB cluster to use one of these DB instance classes. For more information, see Modifying an Amazon RDS DB instance.

RDS Optimized Reads is available in all AWS Regions RDS where one or more of the DB instance classes with local NVMe SSD storage are supported. For information about DB instance classes, see the section called "DB instance classes".

DB instance class availability differs for AWS Regions. To determine whether a DB instance class is supported in a specific AWS Region, see the section called "Determining DB instance class support in AWS Regions".

If you don't want to use RDS Optimized Reads, modify your DB instance or Multi-AZ DB cluster so that it doesn't use a DB instance class that supports the feature.

Monitoring DB instances that use RDS Optimized Reads

You can monitor DB instances that use RDS Optimized Reads with the following CloudWatch metrics:

- `FreeLocalStorage`
- `ReadIOPSLocalStorage`
- `ReadLatencyLocalStorage`
- `ReadThroughputLocalStorage`
- `WriteIOPSLocalStorage`
- `WriteLatencyLocalStorage`
- `WriteThroughputLocalStorage`

These metrics provide data about available instance store storage, IOPS, and throughput. For more information about these metrics, see Amazon CloudWatch instance-level metrics for Amazon RDS.

Limitations for RDS Optimized Reads

The following limitations apply to RDS Optimized Reads:

- RDS Optimized Reads is supported for the following versions:
  - RDS for MySQL version 8.0.28 and higher major and minor versions

For information about RDS for MySQL versions, see MySQL on Amazon RDS versions.

- You can't change the location of temporary objects to persistent storage (Amazon EBS) on the DB instance classes that support RDS Optimized Reads.
- When binary logging is enabled on a DB instance, the maximum transaction size is limited by the size of the instance store. In MySQL, any session that requires more storage than the value of `binlog_cache_size` writes transaction changes to temporary binlog cache files, which are created on the instance store.
- Transactions can fail when the instance store is full.

Improving write performance with RDS Optimized Writes for MySQL

You can improve the performance of write transactions with RDS Optimized Writes for MySQL. When your RDS for MySQL database uses RDS Optimized Writes, it can achieve up to two times higher write transaction throughput.

Topics

- Overview of RDS Optimized Writes
- Using RDS Optimized Writes
- Enabling RDS Optimized Writes on an existing database
- Limitations for RDS Optimized Writes

Overview of RDS Optimized Writes

When you turn on RDS Optimized Writes, your RDS for MySQL databases write only once when flushing data to durable storage without the need for the doublewrite buffer. The databases continue to provide ACID property protections for reliable database transactions, along with improved performance.

Relational databases, like MySQL, provide the *ACID properties* of atomicity, consistency, isolation, and durability for reliable database transactions. To help provide these properties, MySQL uses a data storage area called the *doublewrite buffer* that prevents partial page write errors. These errors occur when there is a hardware failure while the database is updating a page, such as in the case of a power outage. A MySQL database can detect partial page writes and recover with a copy of the page in the doublewrite buffer. While this technique provides protection, it also results in extra write operations. For more information about the MySQL doublewrite buffer, see Doublewrite Buffer in the MySQL documentation.

With RDS Optimized Writes turned on, RDS for MySQL databases write only once when flushing data to durable storage without using the doublewrite buffer. RDS Optimized Writes is useful if you run write-heavy workloads on your RDS for MySQL databases. Examples of databases with write-heavy workloads include ones that support digital payments, financial trading, and gaming applications.

These databases run on DB instance classes that use the AWS Nitro System. Because of the hardware configuration in these systems, the database can write 16-KiB pages directly to data files reliably and durably in one step. The AWS Nitro System makes RDS Optimized Writes possible.

You can set the new database parameter `rds.optimized_writes` to control the RDS Optimized Writes feature for RDS for MySQL databases. Access this parameter in the DB parameter groups of RDS for MySQL version 8.0 and RDS for MySQL version 8.4. Set the parameter using the following values:

- `AUTO` – Turn on RDS Optimized Writes if the database supports it. Turn off RDS Optimized Writes if the database doesn't support it. This setting is the default.
- `OFF` – Turn off RDS Optimized Writes even if the database supports it.

If you have an existing database with an engine version, DB instance class, and/or file system format that doesn't support RDS Optimized Writes, you can enable the feature by creating a blue/green deployment. For more information, see the section called “Enabling on an existing database”.

If you migrate an RDS for MySQL database that is configured to use RDS Optimized Writes to a DB instance class that doesn't support the feature, RDS automatically turns off RDS Optimized Writes for the database.

When RDS Optimized Writes is turned off, the database uses the MySQL doublewrite buffer.

To determine whether an RDS for MySQL database is using RDS Optimized Writes, view the current value of the `innodb_doublewrite` parameter for the database. If the database is using RDS Optimized Writes, this parameter is set to `FALSE` (`0`).

Using RDS Optimized Writes

You can turn on RDS Optimized Writes when you create an RDS for MySQL database with the RDS console, the AWS CLI, or the RDS API. RDS Optimized Writes is turned on automatically when both of the following conditions apply during database creation:

- You specify a DB engine version and DB instance class that support RDS Optimized Writes.
  - RDS Optimized Writes is supported for RDS for MySQL version 8.0.30 and higher. For information about RDS for MySQL versions, see MySQL on Amazon RDS versions.
  - RDS Optimized Writes is supported for RDS for MySQL databases that use the following DB instance classes:

- db.m7i
- db.m7g
- db.m6g
- db.m6gd
- db.m6i
- db.m5
- db.m5d
- db.r7i
- db.r7g
- db.r6g
- db.r6gd
- db.r6i
- db.r5
- db.r5b
- db.r5d
- db.x2idn
- db.x2iedn

  For information about DB instance classes, see the section called "DB instance classes".

  DB instance class availability differs for AWS Regions. To determine whether a DB instance class is supported in a specific AWS Region, see the section called "Determining DB instance class support in AWS Regions".

  To upgrade your database to a DB instance class that supports RDS Optimized Writes, you can create a blue/green deployment. For more information, see the section called "Enabling on an existing database".

- In the parameter group associated with the database, the `rds.optimized_writes` parameter is set to AUTO. In default parameter groups, this parameter is always set to AUTO.

If you want to use a DB engine version and DB instance class that support RDS Optimized Writes, but you don't want to use this feature, then specify a custom parameter group when you create the database. In this parameter group, set the `rds.optimized_writes` parameter to OFF. If

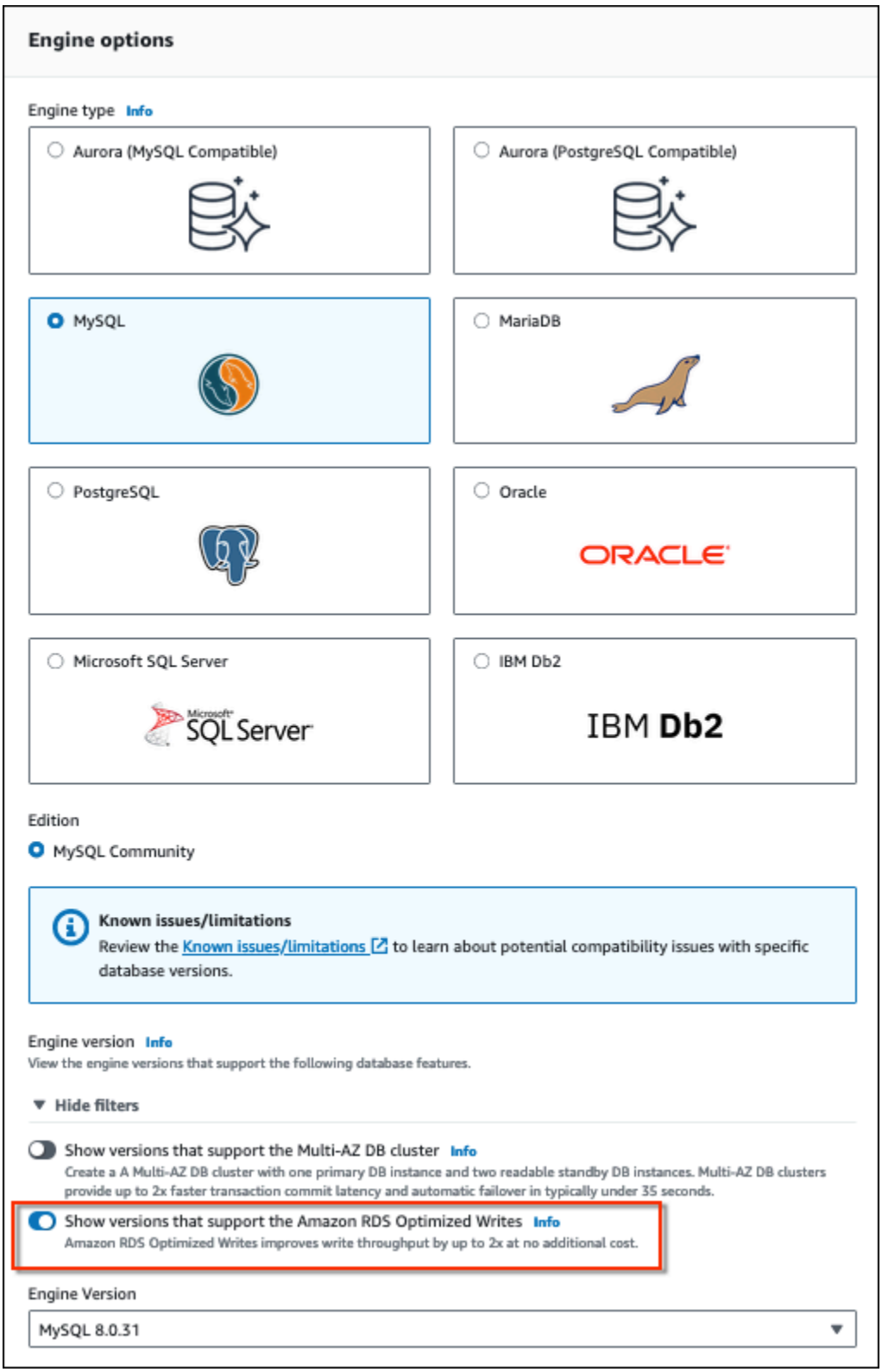
you want the database to use RDS Optimized Writes later, you can set the parameter to AUTO to turn it on. For information about creating custom parameter groups and setting parameters, see Parameter groups for Amazon RDS.

For information about creating a DB instance, see Creating an Amazon RDS DB instance.

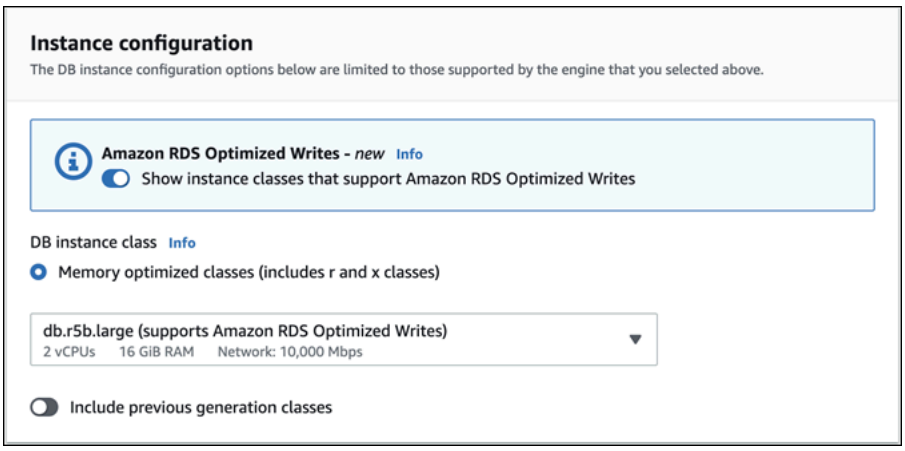
Console

When you use the RDS console to create an RDS for MySQL database, you can filter for the DB engine versions and DB instance classes that support RDS Optimized Writes. After you turn on the filters, you can choose from the available DB engine versions and DB instance classes.

To choose a DB engine version that supports RDS Optimized Writes, filter for the RDS for MySQL DB engine versions that support it in **Engine version**, and then choose a version.



In the **Instance configuration** section, filter for the DB instance classes that support RDS Optimized Writes, and then choose a DB instance class.



After you make these selections, you can choose other settings that meet your requirements and finish creating the RDS for MySQL database with the console.

AWS CLI

To create a DB instance by using the AWS CLI, run the create-db-instance command. Make sure the `--engine-version` and `--db-instance-class` values support RDS Optimized Writes. In addition, make sure the parameter group associated with the DB instance has the `rds.optimized_writes` parameter set to AUTO. This example associates the default parameter group with the DB instance.

Example Creating a DB instance that uses RDS Optimized Writes

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --engine mysql \
    --engine-version 8.0.30 \
    --db-instance-class db.r5b.large \
    --manage-master-user-password \
    --master-username admin \
    --allocated-storage 200
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --engine mysql ^
```

```
    --engine-version 8.0.30 ^
    --db-instance-class db.r5b.large ^
    --manage-master-user-password ^
    --master-username admin ^
    --allocated-storage 200
```

RDS API

You can create a DB instance using the CreateDBInstance operation. When you use this operation, make sure the `EngineVersion` and `DBInstanceClass` values support RDS Optimized Writes. In addition, make sure the parameter group associated with the DB instance has the `rds.optimized_writes` parameter set to AUTO.

Enabling RDS Optimized Writes on an existing database

In order to modify an existing RDS for MySQL database to turn on RDS Optimized Writes, the database must have been created with a supported DB engine version and DB instance class. In addition, the database must have been created *after* RDS Optimized Writes was released on November 27, 2022, as the required underlying file system configuration is incompatible with that of databases created before it was released. If these conditions are met, you can turn on RDS Optimized Writes by setting the `rds.optimized_writes` parameter to AUTO.

If your database was *not* created with a supported engine version, instance class, or file system configuration, you can use RDS Blue/Green Deployments to migrate to a supported configuration. While creating the blue/green deployment, do the following:

- Select **Enable Optimized Writes on green database**, then specify an engine version and DB instance class that supports RDS Optimized Writes. For a list of supported engine versions and instance classes, see Using RDS Optimized Writes.
- Under **Storage**, choose **Upgrade storage file system configuration**. This option upgrades the database to a compatible underlying file system configuration.

When you create the blue/green deployment, if the `rds.optimized_writes` parameter is set to AUTO, RDS Optimized Writes will be automatically enabled on the green environment. You can then switch over the blue/green deployment, which promotes the green environment to be the new production environment.

For more information, see the section called “Creating a blue/green deployment”.

Limitations for RDS Optimized Writes

When you're restoring an RDS for MySQL database from a snapshot, you can only turn on RDS Optimized Writes for the database if all of the following conditions apply:

- The snapshot was created from a database that supports RDS Optimized Writes.
- The snapshot was created from a database that was created *after* RDS Optimized Writes was released.
- The snapshot is restored to a database that supports RDS Optimized Writes.
- The restored database is associated with a parameter group that has the `rds.optimized_writes` parameter set to `AUTO`.

Upgrades of the RDS for MySQL DB engine

When Amazon RDS supports a new version of a database engine, you can upgrade your DB instances to the new version. There are two kinds of upgrades for MySQL databases: major version upgrades and minor version upgrades.

Major version upgrades

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, you must manually perform major version upgrades of your DB instances. You can initiate a major version upgrade by modifying your DB instance. Before you perform a major version upgrade, we recommend that you follow the instructions in Major version upgrades for RDS for MySQL.

For major version upgrades of Multi-AZ DB instance deployments, Amazon RDS simultaneously upgrades the primary and standby replicas. Your DB instance won't be available until the upgrade completes. For major version upgrades of Multi-AZ DB cluster deployments, Amazon RDS upgrades the cluster member instances one at a time.

> **Tip**
>
> You can minimize the downtime required for a major version upgrade by using a blue/green deployment. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

Minor version upgrades

Minor version upgrades include only changes that are backward-compatible with existing applications. You can initiate a minor version upgrade manually by modifying your DB instance. Or, you can enable the **Auto minor version upgrade** option when creating or modifying a DB instance. Doing so means that Amazon RDS automatically upgrades your DB instance after testing and approving the new version. For information about performing an upgrade, see Upgrading a DB instance engine version.

When you perform a minor version upgrade of a Multi-AZ DB cluster, Amazon RDS upgrades the reader DB instances one at a time. Then, one of the reader DB instances switches to be the new writer DB instance. Amazon RDS then upgrades the old writer instance (which is now a reader instance).

> **Note**
>
> The downtime for a minor version upgrade of a Multi-AZ DB *instance* deployment can last for several minutes. Multi-AZ DB clusters typically reduce the downtime of minor version upgrades to approximately 35 seconds. When used with RDS Proxy, you can further reduce downtime to one second or less. For more information, see *Amazon RDS Proxy*. Alternately, you can use an open source database proxy such as ProxySQL,PgBouncer, or the AWS Advanced JDBC Wrapper Driver.

Amazon RDS also supports upgrade rollout policy to manage automatic minor version upgrades across multiple database resources and AWS accounts. For more information, see Using AWS Organizations upgrade rollout policy for automatic minor version upgrades.

If your MySQL DB instance uses read replicas, then you must upgrade all of the read replicas before upgrading the source instance.

Topics

- Considerations for MySQL upgrades
- Finding valid upgrade targets
- MySQL version numbers
- RDS version numbers in RDS for MySQL
- Major version upgrades for RDS for MySQL
- Testing an RDS for MySQL upgrade
- Upgrading a MySQL DB instance
- Automatic minor version upgrades for RDS for MySQL
- Using a read replica to reduce downtime when upgrading an RDS for MySQL database
- Monitoring RDS for MySQL engine upgrades with events

Considerations for MySQL upgrades

Amazon RDS takes two or more DB snapshots during the upgrade process. Amazon RDS takes up to two snapshots of the DB instance *before* making any upgrade changes. If the upgrade doesn't work for your databases, you can restore one of these snapshots to create a DB instance running the old version. Amazon RDS takes another snapshot of the DB instance when the upgrade completes.

Amazon RDS takes these snapshots regardless of whether AWS Backup manages the backups for the DB instance.

> **Note**
>
> Amazon RDS only takes DB snapshots if you have set the backup retention period for your DB instance to a number greater than 0. To change your backup retention period, see Modifying an Amazon RDS DB instance.

After the upgrade is complete, you can't revert to the previous version of the database engine. If you want to return to the previous version, restore the first DB snapshot taken to create a new DB instance.

You control when to upgrade your DB instance to a new version supported by Amazon RDS. This level of control helps you maintain compatibility with specific database versions and test new versions with your application before deploying in production. When you are ready, you can perform version upgrades at the times that best fit your schedule.

If your DB instance uses read replication, then you must upgrade all of the read replicas before upgrading the source instance.

Finding valid upgrade targets

When you use the AWS Management Console to upgrade a DB instance, it shows the valid upgrade targets for the DB instance. You can also run the following AWS CLI command to identify the valid upgrade targets for a DB instance:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
  --engine mysql \
  --engine-version version_number \
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
```

```
  --engine mysql ^
  --engine-version version_number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For example, to identify the valid upgrade targets for a MySQL version 8.0.28 DB instance, run the following AWS CLI command:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
  --engine mysql \
  --engine-version 8.0.28 \
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
  --engine mysql ^
  --engine-version 8.0.28 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

MySQL version numbers

The version numbering sequence for the RDS for MySQL database engine is either in the form of *major.minor.patch.YYYYMMDD* or *major.minor.patch*, for example, 8.0.33.R2.20231201 or 5.7.44. The format used depends on the MySQL engine version. For information about RDS Extended Support version numbering, see Amazon RDS Extended Support version naming.

major

The major version number is both the integer and the first fractional part of the version number, for example, 8.0. A major version upgrade increases the major part of the version number. For example, an upgrade from *5.7*.44 to 8.0.33 is a major version upgrade, where *5.7* and *8.0* are the major version numbers.

minor

The minor version number is the third part of the version number, for example, the 33 in 8.0.33.

patch

The patch is the fourth part of the version number, for example, the R2 in 8.0.33.R2. An RDS patch version includes important bug fixes added to a minor version after its release.

YYYYMMDD

The date is the fifth part of the version number, for example, the 20231201 in 8.0.33.R2.20231201. An RDS date version is a security patch that includes important security fixes added to a minor version after its release. It doesn't include any fixes that might change an engine's behavior.

The following table explains the naming scheme for RDS for MySQL version 8.4.

8.4 minor version	Naming scheme
≥ 3	New DB instances use *major.minor.patch.YYMMDD*, for example, 8.4.3.R2.20241201. Existing DB instances might use *major.minor.patch*, for example, 8.4.3.R2, until your next major or minor version upgrade.

The following table explains the naming scheme for RDS for MySQL version 8.0.

8.0 minor version	Naming scheme
≥ 33	New DB instances use *major.minor.patch.YYMMDD*, for example, 8.0.33.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 8.0.33.R2, until your next major or minor version upgrade.
< 33	Existing DB instances use *major.minor.patch*, for example, 8.0.32.R2.

The following table explains the naming scheme for RDS for MySQL version 5.7.

5.7 minor version	Naming scheme
≥ 42	New DB instances use *major.minor.patch.YYMMDD*, for example, 5.7.42.R2.20231201. Existing DB instances might use *major.minor.patch*, for example, 5.7.42.R2, until your next major or minor version upgrade.

RDS version numbers in RDS for MySQL

RDS version numbers use either the *major*.*minor*.*patch* or the *major*.*minor*.*patch*.*YYYYMMDD* naming scheme. Amazon RDS Extended Support versions use the *minor-RDS*.*YYYYMMDD* minor version naming scheme.

An RDS patch version includes important bug fixes added to a minor version after its release. An RDS date version (*YYYYMMDD*) is a security patch. A security patch doesn't include any fixes that might change the engine's behavior. For information about RDS Extended Support version numbering, see Amazon RDS Extended Support version naming.

You can find out the RDS version number of your RDS for MySQL database with the following SQL query:

```
mysql> select mysql.rds_version();
```

For example, querying an RDS for MySQL 8.0.34 database returns the following output:

```
+---------------------+
| mysql.rds_version() |
+---------------------+
| 8.0.34.R2.20231201  |
+---------------------+
1 row in set (0.01 sec)
```

Major version upgrades for RDS for MySQL

Amazon RDS supports the following in-place upgrades for major versions of the MySQL database engine:

- MySQL 5.7 to MySQL 8.0
- MySQL 8.0 to MySQL 8.4

> **Note**
>
> You can only create MySQL version 5.7, 8.0, and 8.4 DB instances with latest-generation and current-generation DB instance classes.
> In some cases, you want to upgrade a DB instance running on a previous-generation DB instance class to a DB instance with a higher MySQL engine version. In these cases, first modify the DB instance to use a latest-generation or current-generation DB instance class. After you do this, you can then modify the DB instance to use the higher MySQL database engine version. For information on Amazon RDS DB instance classes, see DB instance classes.

Topics

- Overview of MySQL major version upgrades
- Prechecks for upgrades
- Rollback after failure to upgrade

Overview of MySQL major version upgrades

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, Amazon RDS doesn't apply major version upgrades automatically; you must manually modify your DB instance. We recommend that you thoroughly test any upgrade before applying it to your production instances.

To perform a major version upgrade, first perform any available OS updates. After OS updates are complete, upgrade to each major version, for example, 5.7 to 8.0 and then 8.0 to 8.4. For information about upgrading an RDS for MySQL Multi-AZ DB cluster, see Upgrading the engine version of a Multi-AZ DB cluster for Amazon RDS. MySQL DB instances created before April 24, 2014, show an available OS update until the update has been applied. For more information on OS updates, see Applying updates to a DB instance .

During a major version upgrade of MySQL, Amazon RDS runs the MySQL binary `mysql_upgrade` to upgrade tables, if necessary. Also, Amazon RDS empties the `slow_log` and `general_log`

tables during a major version upgrade. To preserve log information, save the log contents before the major version upgrade.

MySQL major version upgrades typically complete in about 10 minutes. Some upgrades might take longer because of the DB instance class size or because the instance doesn't follow certain operational guidelines in Best practices for Amazon RDS. If you upgrade a DB instance from the Amazon RDS console, the status of the DB instance indicates when the upgrade is complete. If you upgrade using the AWS Command Line Interface (AWS CLI), use the describe-db-instances command and check the `Status` value.

Prechecks for upgrades

Amazon RDS runs prechecks before upgrading to check for incompatibilities. These incompatibilities vary based on the MySQL version being upgraded to.

The prechecks include some that are included with MySQL and some that were created specifically by the Amazon RDS team. For information about the prechecks provided by MySQL, see Upgrade checker utility.

The prechecks run before the DB instance is stopped for the upgrade, meaning that they don't cause any downtime when they run. If the prechecks find an incompatibility, Amazon RDS automatically cancels the upgrade before the DB instance is stopped. Amazon RDS also generates an event for the incompatibility. For more information about Amazon RDS events, see Working with Amazon RDS event notification.

Amazon RDS records detailed information about each incompatibility in the log file `PrePatchCompatibility.log`. In most cases, the log entry includes a link to the MySQL documentation for correcting the incompatibility. For more information about viewing log files, see Viewing and listing database log files.

Because of the nature of the prechecks, they analyze the objects in your database. This analysis results in resource consumption and increases the time for the upgrade to complete.

Topics

- Prechecks for upgrades from MySQL 8.0 to 8.4
- Prechecks for upgrades from MySQL 5.7 to 8.0

Prechecks for upgrades from MySQL 8.0 to 8.4

MySQL 8.4 includes a number of incompatibilities with MySQL 8.0. These incompatibilities can cause problems during an upgrade from MySQL 8.0 to MySQL 8.4. So, some preparation might be required on your database for the upgrade to be successful. The following is a general list of these incompatibilities:

- There must be no tables that use obsolete data types or functions.
- Triggers must not have a missing or empty definer or an invalid creation context.
- There must be no keyword or reserved word violations. Some keywords might be reserved in MySQL 8.4 that were not reserved previously.

  For more information, see Keywords and reserved words in the MySQL documentation.
- There must be no tables in the MySQL 8.0 `mysql` system database that have the same name as a table used by the MySQL 8.4 data dictionary.
- There must be no obsolete SQL modes defined in your `sql_mode` system variable setting.
- There must be no tables or stored procedures with individual `ENUM` or `SET` column elements that exceed 255 characters or 1020 bytes in length.
- Your MySQL 8.0 installation must not use features that are not supported in MySQL 8.4.

  For more information, see Features removed in MySQL 8.4 in the MySQL documentation.
- There must be no foreign key constraint names longer than 64 characters.
- For improved Unicode support, review the following information:
  - Consider converting objects that use the `utf8mb3` charset to use the `utf8mb4` charset. The `utf8mb3` character set is deprecated.
  - Consider using `utf8mb4` for character set references instead of `utf8`, because currently `utf8` is an alias for the `utf8mb3` charset. If possible, change `utf8` to `utf8mb4` first, and then upgrade your database.
  - Because older clients can receive an unknown character set error for `utf8mb3`, upgrade your database clients before upgrading your database.

  For more information, see The utf8mb3 character set (3-byte UTF-8 unicode encoding) in the MySQL documentation.

To change the character sets, you can manually perform a backup, restore, and replication of your database. Or you can use Amazon RDS Blue/Green Deployments. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

When you start an upgrade from MySQL 8.0 to 8.4, Amazon RDS runs prechecks automatically to detect these incompatibilities. For information about upgrading to MySQL 8.4, see Upgrading MySQL in the MySQL documentation.

These prechecks are mandatory. You can't choose to skip them. The prechecks provide the following benefits:

- They enable you to avoid unplanned downtime during the upgrade.
- If there are incompatibilities, Amazon RDS prevents the upgrade and provides a log for you to learn about them. You can then use the log to prepare your database for the upgrade to MySQL 8.4 by reducing the incompatibilities. For detailed information about removing incompatibilities, see Preparing your installation for upgrade in the MySQL documentation.

Prechecks for upgrades from MySQL 5.7 to 8.0

MySQL 8.0 includes a number of incompatibilities with MySQL 5.7. These incompatibilities can cause problems during an upgrade from MySQL 5.7 to MySQL 8.0. So, some preparation might be required on your database for the upgrade to be successful. The following is a general list of these incompatibilities:

- There must be no tables that use obsolete data types or functions.
- There must be no orphan *.frm files.
- Triggers must not have a missing or empty definer or an invalid creation context.
- There must be no partitioned table that uses a storage engine that does not have native partitioning support.
- There must be no keyword or reserved word violations. Some keywords might be reserved in MySQL 8.0 that were not reserved previously.

  For more information, see Keywords and reserved words in the MySQL documentation.
- There must be no tables in the MySQL 5.7 `mysql` system database that have the same name as a table used by the MySQL 8.0 data dictionary.
- There must be no obsolete SQL modes defined in your `sql_mode` system variable setting.

- There must be no tables or stored procedures with individual `ENUM` or `SET` column elements that exceed 255 characters or 1020 bytes in length.
- Before upgrading to MySQL 8.0.13 or higher, there must be no table partitions that reside in shared InnoDB tablespaces.
- There must be no queries and stored program definitions from MySQL 8.0.12 or lower that use `ASC` or `DESC` qualifiers for `GROUP BY` clauses.
- Your MySQL 5.7 installation must not use features that are not supported in MySQL 8.0.

  For more information, see Features removed in MySQL 8.0 in the MySQL documentation.
- There must be no foreign key constraint names longer than 64 characters.
- For improved Unicode support, review the following information:
  - Consider converting objects that use the `utf8mb3` charset to use the `utf8mb4` charset. The `utf8mb3` character set is deprecated.
  - Consider using `utf8mb4` for character set references instead of `utf8`, because currently `utf8` is an alias for the `utf8mb3` charset. If possible, change `utf8` to `utf8mb4` first, and then upgrade your database.
  - Because older clients can receive an unknown character set error for `utf8mb3`, upgrade your database clients before upgrading your database.

  For more information, see The utf8mb3 character set (3-byte UTF-8 unicode encoding) in the MySQL documentation.

  To change the character sets, you can manually perform a backup, restore, and replication of your database. Or you can use Amazon RDS Blue/Green Deployments. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

When you start an upgrade from MySQL 5.7 to 8.0, Amazon RDS runs prechecks automatically to detect these incompatibilities. For information about upgrading to MySQL 8.0, see Upgrading MySQL in the MySQL documentation.

These prechecks are mandatory. You can't choose to skip them. The prechecks provide the following benefits:

- They enable you to avoid unplanned downtime during the upgrade.
- If there are incompatibilities, Amazon RDS prevents the upgrade and provides a log for you to learn about them. You can then use the log to prepare your database for the upgrade to MySQL

8.0 by reducing the incompatibilities. For detailed information about removing incompatibilities, see Preparing your installation for upgrade in the MySQL documentation and Upgrading to MySQL 8.0? Here is what you need to know... on the MySQL Server Blog.

Rollback after failure to upgrade

When you upgrade a DB instance from MySQL version 5.7 to MySQL version 8.0 or from MySQL version 8.0 to 8.4, the upgrade can fail. In particular, it can fail if the data dictionary contains incompatibilities that weren't captured by the prechecks. In this case, the database fails to start up successfully in the new MySQL 8.0 or 8.4 version. At this point, Amazon RDS rolls back the changes performed for the upgrade. After the rollback, the MySQL DB instance is running the original version:

- MySQL version 8.0 (for a rollback from MySQL 8.4)
- MySQL version 5.7 (for a rollback from MySQL 8.0)

When an upgrade fails and is rolled back, Amazon RDS generates an event with the event ID RDS-EVENT-0188.

Typically, an upgrade fails because there are incompatibilities in the metadata between the databases in your DB instance and the target MySQL version. When an upgrade fails, you can view the details about these incompatibilities in the `upgradeFailure.log` file. Resolve the incompatibilities before attempting to upgrade again.

During an unsuccessful upgrade attempt and rollback, your DB instance is restarted. Any pending parameter changes are applied during the restart and persist after the rollback.

For more information about upgrading to MySQL 8.0, see the following topics in the MySQL documentation:

- Preparing Your Installation for Upgrade
- Upgrading to MySQL 8.0? Here is what you need to know…

For more information about upgrading to MySQL 8.4, see Preparing Your Installation for Upgrade in the MySQL documentation.

Testing an RDS for MySQL upgrade

Before you perform a major version upgrade on your DB instance, thoroughly test your database for compatibility with the new version. In addition, thoroughly test all applications that access the database for compatibility with the new version. We recommend that you use the following procedure.

To test a major version upgrade

1. Review the upgrade documentation for the new version of the database engine to see if there are compatibility issues that might affect your database or applications:
   - Changes in MySQL 5.7
   - Changes in MySQL 8.0
   - Changes in MySQL 8.4
2. If your DB instance is a member of a custom DB parameter group, create a new DB parameter group with your existing settings that is compatible with the new major version. Specify the new DB parameter group when you upgrade your test instance, so your upgrade testing ensures that it works correctly. For more information about creating a DB parameter group, see Parameter groups for Amazon RDS.
3. Create a DB snapshot of the DB instance to be upgraded. For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.
4. Restore the DB snapshot to create a new test DB instance. For more information, see Restoring to a DB instance.
5. Modify this new test DB instance to upgrade it to the new version, using one of the methods detailed following. If you created a new parameter group in step 2, specify that parameter group.
6. Evaluate the storage used by the upgraded instance to determine if the upgrade requires additional storage.
7. Run as many of your quality assurance tests against the upgraded DB instance as needed to ensure that your database and application work correctly with the new version. Implement any new tests needed to evaluate the impact of any compatibility issues that you identified in step 1. Test all stored procedures and functions. Direct test versions of your applications to the upgraded DB instance.

8. If all tests pass, then perform the upgrade on your production DB instance. We recommend that you don't allow write operations to the DB instance until you confirm that everything is working correctly.

Upgrading a MySQL DB instance

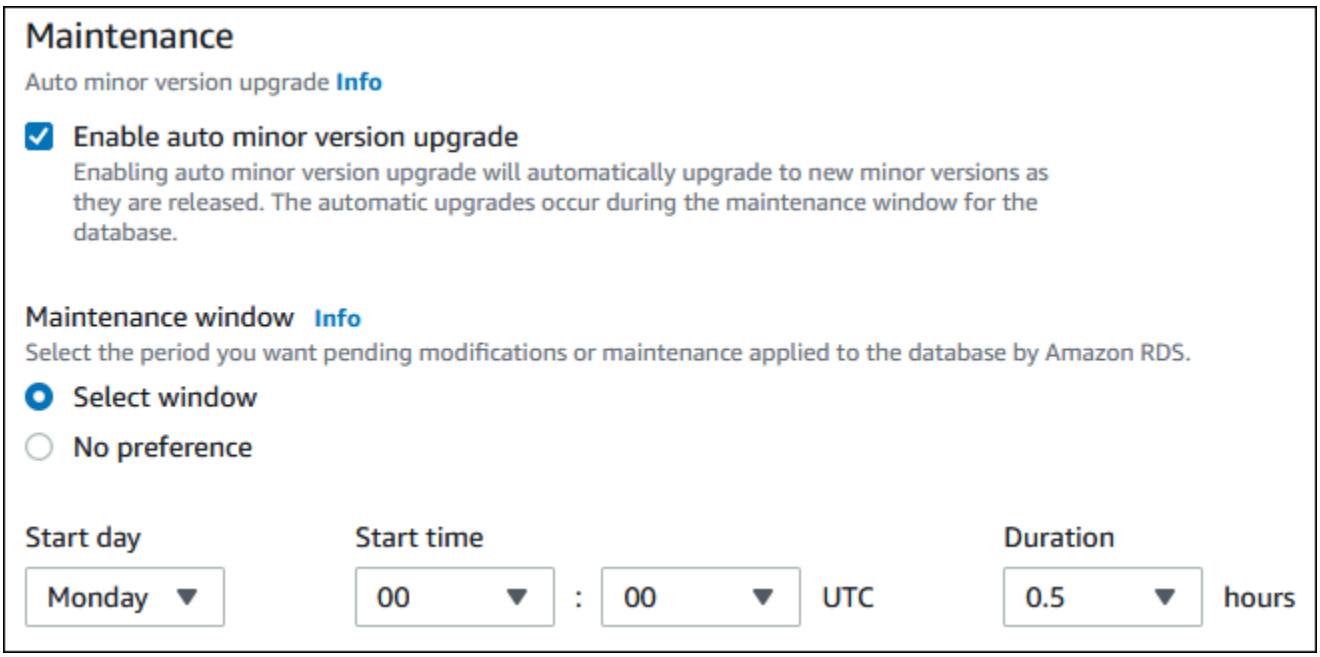
For information about manually or automatically upgrading a MySQL DB instance, see Upgrading a DB instance engine version.

Automatic minor version upgrades for RDS for MySQL

If you specify the following settings when creating or modifying a DB instance, you can have your DB instance automatically upgraded.

- The **Auto minor version upgrade** setting is enabled.
- The **Backup retention period** setting is greater than 0.

In the AWS Management Console, these settings are under **Additional configuration**. The following image shows the **Auto minor version upgrade** setting.



For more information about these settings, see Settings for DB instances.

For some RDS for MySQL major versions in some AWS Regions, one minor version is designated by RDS as the automatic upgrade version. After a minor version has been tested and approved by

Amazon RDS, the minor version upgrade occurs automatically during your maintenance window. RDS doesn't automatically set newer released minor versions as the automatic upgrade version. Before RDS designates a newer automatic upgrade version, several criteria are considered, such as the following:

- Known security issues
- Bugs in the MySQL community version
- Overall fleet stability since the minor version was released

You can run the following AWS CLI command to determine the current automatic minor upgrade target version for a specified MySQL minor version in a specific AWS Region.

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mysql \
--engine-version minor_version \
--region region \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mysql ^
--engine-version minor_version ^
--region region ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output text
```

For example, the following AWS CLI command determines the automatic minor upgrade target for MySQL minor version 8.0.11 in the US East (Ohio) AWS Region (us-east-2).

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine mysql \
```

```
--engine-version 8.0.11 \
--region us-east-2 \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output table
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine mysql ^
--engine-version 8.0.11 ^
--region us-east-2 ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output table
```

Your output is similar to the following.

```
---------------------------------
|    DescribeDBEngineVersions    |
+--------------+-----------------+
|  AutoUpgrade |  EngineVersion  |
+--------------+-----------------+
|  False       |  8.0.15         |
|  False       |  8.0.16         |
|  False       |  8.0.17         |
|  False       |  8.0.19         |
|  False       |  8.0.20         |
|  False       |  8.0.21         |
|  True        |  8.0.23         |
|  False       |  8.0.25         |
+--------------+-----------------+
```

In this example, the `AutoUpgrade` value is `True` for MySQL version 8.0.23. So, the automatic minor upgrade target is MySQL version 8.0.23, which is highlighted in the output.

A MySQL DB instance is automatically upgraded during your maintenance window if the following criteria are met:

- The **Auto minor version upgrade** setting is enabled.
- The **Backup retention period** setting is greater than 0.

- The DB instance is running a minor DB engine version that is less than the current automatic upgrade minor version.

For more information, see Automatically upgrading the minor engine version.

Using a read replica to reduce downtime when upgrading an RDS for MySQL database

In most cases, a blue/green deployment is the best option to reduce downtime when upgrading a MySQL DB instance. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.

If you can't use a blue/green deployment and your MySQL DB instance is currently in use with a production application, you can use the following procedure to upgrade the database version for your DB instance. This procedure can reduce the amount of downtime for your application.

By using a read replica, you can perform most of the maintenance steps ahead of time and minimize the necessary changes during the actual outage. With this technique, you can test and prepare the new DB instance without making any changes to your existing DB instance.

The following procedure shows an example of upgrading from MySQL version 5.7 to MySQL version 8.0. You can use the same general steps for upgrades to other major versions. You can use the same general steps for upgrades to other major versions.

> **Note**
>
> When you are upgrading from MySQL version 5.7 to MySQL version 8.0, or from MySQL version 8.0 to MySQL version 8.4, complete the prechecks before performing the upgrade. For more information, see Prechecks for upgrades from MySQL 5.7 to 8.0 and Prechecks for upgrades from MySQL 8.0 to 8.4.

To upgrade a MySQL database while a DB instance is in use

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Create a read replica of your MySQL 5.7 DB instance. This process creates an upgradable copy of your database. Other read replicas of the DB instance might also exist.

a. In the console, choose **Databases**, and then choose the DB instance that you want to upgrade.

b. For **Actions**, choose **Create read replica**.

c. Provide a value for **DB instance identifier** for your read replica and ensure that the **DB instance class** and other settings match your MySQL 5.7 DB instance.

d. Choose **Create read replica**.

3. (Optional) When the read replica has been created and **Status** shows **Available**, convert the read replica into a Multi-AZ deployment and enable backups.

   By default, a read replica is created with backups disabled. Because the read replica ultimately becomes the production DB instance, it is a best practice to configure a Multi-AZ deployment and enable backups.

   a. In the console, choose **Databases**, and then choose the read replica that you just created.

   b. Choose **Modify**.

   c. For **Multi-AZ deployment**, choose **Create a standby instance**.

   d. For **Backup Retention Period**, choose a positive nonzero value, such as 3 days, and then choose **Continue**.

   e. For **Scheduling of modifications**, choose **Apply immediately**.

   f. Choose **Modify DB instance**.

4. When the read replica **Status** shows **Available**, upgrade the read replica to MySQL 8.0:

   a. In the console, choose **Databases**, and then choose the read replica that you just created.

   b. Choose **Modify**.

   c. For **DB engine version**, choose the MySQL 8.0 version to upgrade to, and then choose **Continue**.

   d. For **Scheduling of modifications**, choose **Apply immediately**.

   e. Choose **Modify DB instance** to start the upgrade.

5. When the upgrade is complete and **Status** shows **Available**, verify that the upgraded read replica is up-to-date with the source MySQL 5.7 DB instance. To verify, connect to the read replica and run the `SHOW REPLICA STATUS` command. If the `Seconds_Behind_Master` field is `0`, then replication is up-to-date.

> **Note**
>
> Previous versions of MySQL used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MySQL version before 8.0.23, then use `SHOW SLAVE STATUS`.

6. (Optional) Create a read replica of your read replica.

   If you want the DB instance to have a read replica after it is promoted to a standalone DB instance, you can create the read replica now.

   a. In the console, choose **Databases**, and then choose the read replica that you just upgraded.

   b. For **Actions**, choose **Create read replica**.

   c. Provide a value for **DB instance identifier** for your read replica and ensure that the **DB instance class** and other settings match your MySQL 5.7 DB instance.

   d. Choose **Create read replica**.

7. (Optional) Configure a custom DB parameter group for the read replica.

   If you want the DB instance to use a custom parameter group after it is promoted to a standalone DB instance, you can create the DB parameter group now and associate it with the read replica.

   a. Create a custom DB parameter group for MySQL 8.0. For instructions, see Creating a DB parameter group in Amazon RDS.

   b. Modify the parameters that you want to change in the DB parameter group you just created. For instructions, see Modifying parameters in a DB parameter group in Amazon RDS.

   c. In the console, choose **Databases**, and then choose the read replica.

   d. Choose **Modify**.

   e. For **DB parameter group**, choose the MySQL 8.0 DB parameter group you just created, and then choose **Continue**.

   f. For **Scheduling of modifications**, choose **Apply immediately**.

   g. Choose **Modify DB instance** to start the upgrade.

8. Make your MySQL 8.0 read replica a standalone DB instance.

> **Important**
>
> When you promote your MySQL 8.0 read replica to a standalone DB instance, it is no longer a replica of your MySQL 5.7 DB instance. We recommend that you promote your MySQL 8.0 read replica during a maintenance window when your source MySQL 5.7 DB instance is in read-only mode and all write operations are suspended. When the promotion is completed, you can direct your write operations to the upgraded MySQL 8.0 DB instance to ensure that no write operations are lost.
> In addition, we recommend that, before promoting your MySQL 8.0 read replica, you perform all necessary data definition language (DDL) operations on your MySQL 8.0 read replica. An example is creating indexes. This approach avoids negative effects on the performance of the MySQL 8.0 read replica after it has been promoted. To promote a read replica, use the following procedure.

a. In the console, choose **Databases**, and then choose the read replica that you just upgraded.
b. For **Actions**, choose **Promote**.
c. Choose **Yes** to enable automated backups for the read replica instance. For more information, see Introduction to backups.
d. Choose **Continue**.
e. Choose **Promote Read Replica**.

9. You now have an upgraded version of your MySQL database. At this point, you can direct your applications to the new MySQL 8.0 DB instance.

Monitoring RDS for MySQL engine upgrades with events

When you upgrade the engine version of a RDS for MySQL database, Amazon RDS emits a specific event during each phase of the process. To track the progress of an upgrade, you can view or subscribe to these events.

For more information about RDS events, see Monitoring Amazon RDS events.

For detailed information about a specific Amazon RDS event that occurs during your engine upgrade, see Amazon RDS event categories and event messages.

Upgrading a MySQL DB snapshot engine version

With Amazon RDS, you can create a storage volume DB snapshot of your MySQL DB instance. When you create a DB snapshot, the snapshot is based on the engine version used by your DB instance. You can upgrade the engine version for your DB snapshots.

For RDS for MySQL, you can upgrade a version 5.7 snapshot to version 8.0, or a version 8.0 snapshot to version 8.4. You can upgrade encrypted or unencrypted DB snapshots.

To view the available engine versions for your RDS for MySQL DB snapshot, use the following AWS CLI example.

```
aws rds describe-db-engine-versions --engine mysql --include-all --engine-
version example-engine-version --query "DBEngineVersions[*].ValidUpgradeTarget[*].
{EngineVersion:EngineVersion}" --output text
```

If you don't see results for your snapshot, your engine version might be deprecated. If your engine version is deprecated, we recommend that you upgrade to the newest major version upgrade target or to one of the other available upgrade targets for that version. For more information, see Upgrade options for DB snapshots with unsupported engine versions for RDS for MySQL.

After restoring a DB snapshot upgraded to a new engine version, make sure to test that the upgrade was successful. For more information about a major version upgrade, see the section called “Upgrades of the MySQL DB engine”. To learn how to restore a DB snapshot, see the section called “Restoring to a DB instance”.

> **Note**
>
> You can't upgrade automated DB snapshots that were created during the automated backup process.

You can upgrade a DB snapshot using the AWS Management Console, AWS CLI, or RDS API.

Console

To upgrade a DB snapshot engine version using the AWS Management Console, use the following procedure.

To upgrade a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the snapshot that you want to upgrade.
4. For **Actions**, choose **Upgrade snapshot**. The **Upgrade snapshot** page appears.
5. Choose the **New engine version** to upgrade to.
6. Choose **Save changes** to upgrade the snapshot.

   During the upgrade process, all snapshot actions are disabled for this DB snapshot. Also, the DB snapshot status changes from **Available** to **Upgrading**, and then changes to **Active** upon completion. If the DB snapshot can't be upgraded because of snapshot corruption issues, the status changes to **Unavailable**. You can't recover the snapshot from this state.

   > **Note**
   >
   > If the DB snapshot upgrade fails, the snapshot is rolled back to the original state with the original version.

AWS CLI

To upgrade a DB snapshot to a new database engine version, run the AWS CLI modify-db-snapshot command.

Options

- `--db-snapshot-identifier` – The identifier of the DB snapshot to upgrade. The identifier must be a unique Amazon Resource Name (ARN). For more information, see Amazon Resource Names (ARNs) in Amazon RDS.
- `--engine-version` – The engine version to upgrade the DB snapshot to.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-snapshot \
```

```
    --db-snapshot-identifier my_db_snapshot \
    --engine-version new_version
```

For Windows:

```
aws rds modify-db-snapshot ^
    --db-snapshot-identifier my_db_snapshot ^
    --engine-version new_version
```

Amazon RDS API

To upgrade a DB snapshot to a new database engine version, call the RDS API ModifyDBSnapshot operation.

Parameters

- `DBSnapshotIdentifier` – The identifier of the DB snapshot to upgrade. The identifier must be a unique Amazon Resource Name (ARN). For more information, see Amazon Resource Names (ARNs) in Amazon RDS.
- `EngineVersion` – The engine version to upgrade the DB snapshot to.

Upgrade options for DB snapshots with unsupported engine versions for RDS for MySQL

The following table shows which engine versions you can upgrade to from an unsupported engine version for RDS for MySQL DB snapshots.

> **Note**
>
> You might have to upgrade your DB snapshot more than once to upgrade to your chosen engine version.

DB snapshot engine version	Engine versions available for upgrade
5.5.8	5.5.62, 5.6.51
5.5.12	5.5.62, 5.6.51

DB snapshot engine version	Engine versions available for upgrade
5.5.20	5.5.62, 5.6.51
5.5.23	5.5.62, 5.6.51
5.5.25a	5.5.62, 5.6.51
5.5.27	5.5.62, 5.6.51
5.5.31	5.5.62, 5.6.51
5.5.33	5.5.62, 5.6.51
5.5.37	5.5.62, 5.6.51
5.5.38	5.5.62, 5.6.51
5.5.40	5.5.62, 5.6.51
5.5.40a	5.5.62, 5.6.51
5.5.40b	5.5.62, 5.6.51
5.5.41	5.5.62, 5.6.51
5.5.42	5.5.62, 5.6.51
5.5.59	5.5.62, 5.6.51
5.6.12	5.6.51, 5.7.44
5.6.13	5.6.51, 5.7.44
5.6.17	5.6.51, 5.7.44
5.6.19	5.6.51, 5.7.44
5.6.19a	5.6.51, 5.7.44
5.6.19b	5.6.51, 5.7.44

DB snapshot engine version	Engine versions available for upgrade
5.6.21	5.6.51, 5.7.44
5.6.21b	5.6.51, 5.7.44
5.6.22	5.6.51, 5.7.44
5.6.23	5.6.51, 5.7.44
5.6.27	5.6.51, 5.7.44
5.6.27a	5.6.51, 5.7.44
5.7.10	5.7.44, 5.7.44-rds.20240408, 5.7.44-rds.20240529, 5.7.44-rds.20250103, 5.7.44-rds.20250213, 8.0.32, 8.0.33, 8.0.34, 8.0.35, 8.0.36, 8.0.37, 8.0.39, 8.0.40, 8.0.41
5.7.11	5.7.44, 5.7.44-rds.20240408, 5.7.44-rds.20240529, 5.7.44-rds.20250103, 5.7.44-rds.20250213, 8.0.32, 8.0.33, 8.0.34, 8.0.35, 8.0.36, 8.0.37, 8.0.39, 8.0.40, 8.0.41
5.7.12	5.7.44, 5.7.44-rds.20240408, 5.7.44-rds.20240529, 5.7.44-rds.20250103, 5.7.44-rds.20250213, 8.0.32, 8.0.33, 8.0.34, 8.0.35, 8.0.36, 8.0.37, 8.0.39, 8.0.40, 8.0.41

Importing data into an Amazon RDS for MySQL DB instance

You can use several different techniques to import data into an RDS for MySQL DB instance. The best approach depends on a number of factors:

- Source of the data
- Amount of data
- One-time import or ongoing
- Amount of downtime

If you are also migrating an application with the data, the amount of downtime is important to consider.

The following table lists techniques to importing data into an RDS for MySQL DB instance:

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
Existing MySQL database on premises or on Amazon EC2	Any	One time	Some	Create a backup of your on-premis es database, store it on Amazon S3, and then restore the backup file to a new Amazon RDS DB instance running MySQL.	Restoring a backup into an Amazon RDS for MySQL DB instance
Existing MySQL database on premises or on Amazon EC2	Any	Ongoing	Minimal	Configure replication with an existing MySQL database as the replication source.	Configuri ng binary log file position replicati on with an

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
					external source instance Importing data to an Amazon RDS for MySQL database with reduced downtime

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
Any existing database	Any	One time or ongoing	Minimal	Use AWS Database Migration Service to migrate the database with minimal downtime and, for many database DB engines, continue ongoing replication.	What is AWS Database Migration Service and Using a MySQL-com patible database as a target for AWS DMS in the *AWS Database Migration Service User Guide*
Existing MySQL DB instance	Any	One time or ongoing	Minimal	Create a read replica for ongoing replication. Promote the read replica for one-time creation of a new DB instance.	Working with DB instance read replicas

Source	Amount of data	One time or ongoing	Applicati on downtim	Technique	More informati on
Existing MySQL database	Small	One time	Some	Copy the data directly to your MySQL DB instance using a command-line utility.	Importing data from an external MySQL database to an Amazon RDS for MySQL DB instance
Data not stored in an existing database	Medium	One time	Some	Create flat files and import them using MySQL `LOAD DATA LOCAL INFILE` statements.	Importing data from any source to an Amazon RDS for MySQL DB instance

Note

The `mysql` system database contains authentication and authorization information required to log in to your DB instance and access your data. Dropping, altering, renaming, or truncating tables, data, or other contents of the `mysql` database in your DB instance can result in error and might render the DB instance and your data inaccessible. If this occurs, you can restore the DB instance from a snapshot using the AWS CLI restore-db-instance-

from-db-snapshot command. You can recover the DB instance using the AWS CLI restore-db-instance-to-point-in-time command.

Importing data considerations for MySQL

The following content contains technical information related to loading data into MySQL. This content is aimed at users who are familiar with the MySQL server architecture.

Binary logging

Enabling binary logging reduces data load performance and requires up to four times additional disk space compared to disabled logging. The transaction size used to load the data directly affects system performance and disk space needs—larger transactions require more resources.

Transaction size

Transaction size influences the following aspects of MySQL data loads:

- Resource consumption
- Disk space utilization
- Resume process
- Time to recover
- Input format (flat files or SQL)

This section describes how transaction size affects binary logging and makes the case for disabling binary logging during large data loads. You can enable and disable binary logging by setting the Amazon RDS automated backup retention period. Non-zero values enable binary logging, and zero disables it. For more information, see Backup retention period.

This section also describes the impact of large transactions on InnoDB and why it's important to keep transaction sizes small.

Small transactions

For small transactions, binary logging doubles the number of disk writes required to load the data. This effect can severely degrade performance for other database sessions and increase the time required to load the data. The degradation experienced depends in part on the following factors:

- Upload rate
- Other database activity taking place during the load
- Capacity of your Amazon RDS DB instance

The binary logs also consume disk space roughly equal to the amount of data loaded until the logs are backed up and removed. Amazon RDS minimizes this by frequently backing up and removing binary logs.

Large transactions

For large transactions, binary logging triples IOPS and disk usage for the following reasons:

- The binary log cache stores transaction data temporarily on disk.
- This cache grows with the transaction size, which consumes disk space.
- When the transaction (commit or rollback) completes, the system copies the cache to the binary log.

This process creates three copies of the data:

- The original data
- The cache on disk
- The final binary log entry

Each write operation incurs additional IO, further impacting performance.

Because of this, binary logging requires triple the disk space compared to disabled logging. For example, loading 10 GiB of data as a single transaction creates three copies:

- 10 GiB for the table data
- 10 GiB for the binary log cache
- 10 GiB for the binary log file

The total temporary disk space required is 30 GiB.

Important disk space considerations:

- The cache file persists until either the session ends or a new transaction creates another cache.
- The binary log remains until it's backed up, potentially holding 20 GiB (cache and log) for an extended period.

If you use `LOAD DATA LOCAL INFILE` to load the data, data recovery creates a fourth copy in case the database has to be recovered from a backup made before the load. During recovery, MySQL extracts the data from the binary log into a flat file. MySQL then runs `LOAD DATA LOCAL INFILE`. Building on the preceding example, this recovery requires a total temporary disk space of 40 GiB, or 10 GiB each for table, cache, log, and local file. Without at least 40 GiB of free disk space, recovery fails.

Optimizing large data loads

For large data loads, disable binary logging to reduce overhead and disk space requirements. You can disable binary logging by setting the backup retention period to 0. After loading completes, restore the backup retention period to the appropriate non-zero value. For more information, see Modifying an Amazon RDS DB instance and Backup retention period in the settings table.

> **Note**
>
> If the DB instance is a source DB instance for read replicas, then you can't set the backup retention period to 0.

Before loading the data, we recommend that you create a DB snapshot. For more information, see Managing manual backups.

InnoDB

The following information about undo logging and recovery options supports keeping InnoDB transactions small to optimize database performance.

Understanding InnoDB undo logging

Undo is a logging mechanism that enables transaction rollback and supports multi-version concurrency control (MVCC).

For MySQL 5.7 and lower versions, undo logs are stored in the InnoDB system tablespace (usually ibdata1) and are retained until the purge thread removes them. As a result, large data load

transactions can cause the system tablespace to become quite large and consume disk space that you can't reclaim unless you recreate the database.

For all MySQL versions, the purge thread must wait to remove any undo logs until the oldest active transaction either commits or rolls back. If the database is processing other transactions during the load, their undo logs also accumulate and can't be removed, even if the transactions commit and no other transaction needs the undo logs for MVCC. In this situation, all transactions—including read-only transactions—slow down. This slowdown occurs because all transactions access all rows that any transaction—not just the load transaction—changes. In effect, transactions must scan through undo logs that long-running load transactions prevented from being purged during an undo log cleanup. This affects performance for any operation accessing modified rows.

InnoDB transaction recovery options

Although InnoDB optimizes commit operations, large transaction rollbacks are slow. For faster recovery, perform a point-in-time recovery or restore a DB snapshot. For more information, see Point-in-time recovery and Restoring to a DB instance.

Data import formats

MySQL supports two data import formats: flat files and SQL. Review the information about each format to determine the best option for your needs.

Flat files

For small transactions, load flat files with `LOAD DATA LOCAL INFILE`. This data import format can provide the following benefits over using SQL:

- Less network traffic
- Lower data transmission costs
- Decreased database processing overhead
- Faster processing

`LOAD DATA LOCAL INFILE` loads the entire flat file as one transaction. Keep the size of the individual files small for the following advantages:

- **Resume capability** – You can keep track of which files have been loaded. If a problem arises during the load, you can pick up where you left off. You might need to retransmit some data to Amazon RDS, but with small files, the amount retransmitted is minimal.

- **Parallel data loading** – If you have sufficient IOPS and network bandwidth for a single file load, loading in parallel could save time.
- **Load rate control** – If your data load has a negative impact on other processes, you can control the load rate by increasing the interval between files.

Large transactions reduce the benefits of using `LOAD DATA LOCAL INFILE` to import data. When you can't break a large amount of data into smaller files, consider using SQL.

SQL

SQL has one main advantage over flat files: you can easily keep transaction sizes small. However, SQL can take significantly longer to load than flat files. Also, after a failure, it can be difficult to determine where to resume—you can't restart mysqldump files. If a failure occurs while loading a mysqldump file, you must modify or replace the file before the load can resume. Or, alternatively, after you correct the cause of the failure, you can restore to the point in time before the load and resend the file. For more information, see Point-in-time recovery.

Using Amazon RDS DB snapshots for database checkpoints

If you load data over long durations—such as hours or days—without binary logging, use DB snapshots to provide periodic checkpoints for data safety. Each DB snapshot creates a consistent copy of your database instance that serves as a recovery point during system failures or data corruption events. Because DB snapshots are fast, frequent checkpointing has minimal impact on load performance. You can delete previous DB snapshots without impacting database durability or recovery capabilities. For more information about DB snapshots, see Managing manual backups.

Reducing database load times

The following items are additional tips to reduce load times:

- Create all secondary indexes before loading data into MySQL databases. Unlike other database systems, MySQL rebuilds the entire table when adding or modifying secondary indexes. This process creates a new table with index changes, copies all data, and drops the original table.
- Load data in primary key order. For InnoDB tables, this can reduce load times by 75%–80% and reduce data file size by 50%.
- Disable foreign key constraints by setting `foreign_key_checks` to `0`. This is often required for flat files loaded with `LOAD DATA LOCAL INFILE`. For any load, disabling foreign key

checks accelerates data loading. After loading completes, re-enable constraints by setting `foreign_key_checks` to 1 and verify the data.

- Load data in parallel unless approaching a resource limit. To enable concurrent loading across multiple table segments, use partitioned tables when appropriate.
- To reduce SQL execution overhead, combine multiple `INSERT` statements into single multi-value `INSERT` operations. `mysqldump` implements this optimization automatically.
- Reduce InnoDB log IO operations by setting `innodb_flush_log_at_trx_commit` to `0`. After loading completes, restore `innodb_flush_log_at_trx_commit` to 1.

> **Warning**
>
> Setting `innodb_flush_log_at_trx_commit` to `0` causes InnoDB to flush its logs every second instead of at each commit. This setting increases performance but can risk transaction loss during system failures.

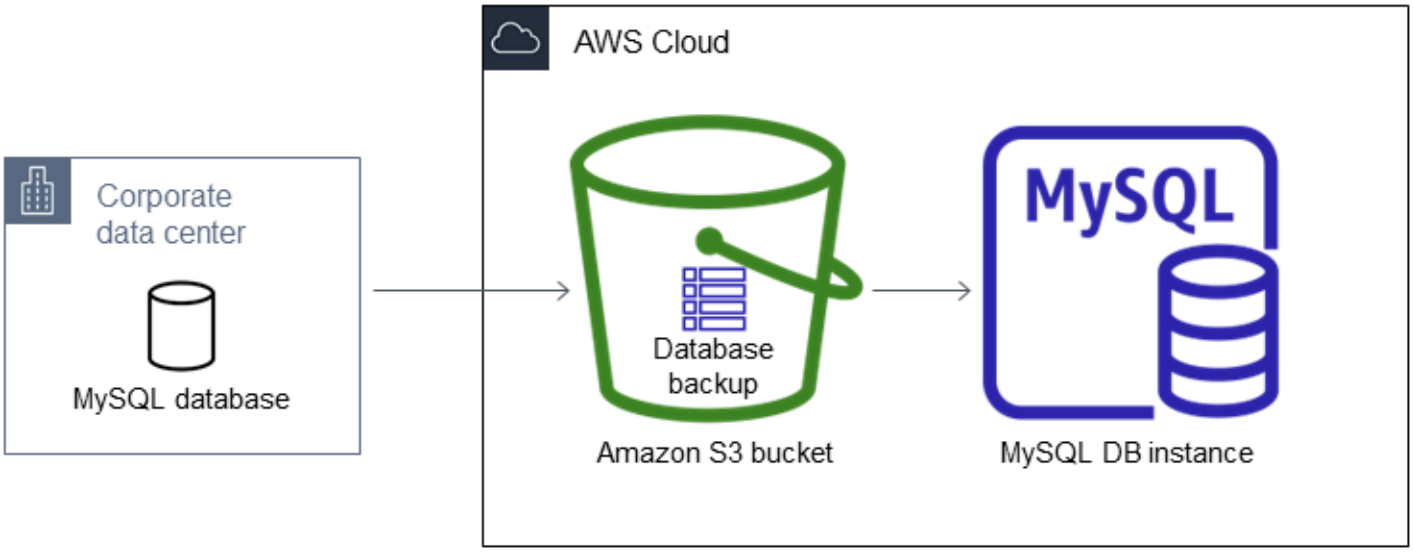
- If you are loading data into a DB instance that doesn't have read replicas, set `sync_binlog` to `0`. After loading completes, restore `sync_binlog parameterto` 1.
- Load data into a Single-AZ DB instance before converting the DB instance to a Multi-AZ deployment. If the DB instance already uses a Multi-AZ deployment, we don't recommend switching to a Single-AZ deployment for data loading. Doing so only provides marginal improvements.

Restoring a backup into an Amazon RDS for MySQL DB instance

Amazon RDS supports importing MySQL databases with backup files. You can create a backup of your database, store the backup file on Amazon S3, and then restore the backup file to a new Amazon RDS DB instance running MySQL. Amazon RDS supports importing backup files from Amazon S3 in all AWS Regions.

The scenario described in this section restores a backup of an on-premises database. As long as the database is accessible, you can use this technique for databases in other locations, such as Amazon EC2 or other cloud services.

The following diagram shows the supported scenario.



If your on-premises database can be offline while you create, copy, and restore backup files, then we recommend that you use backup files to import your database to Amazon RDS. If your database can't be offline, then you can use one of the following methods:

- **Binary logs** – First, import backup files from Amazon S3 and to Amazon RDS, as explained in this topic. Then use binary log (binlog) replication to update your database. For more information, see Configuring binary log file position replication with an external source instance.
- **AWS Database Migration Service** – Use AWS Database Migration Service to migrate your database to Amazon RDS. For more information, see What is AWS Database Migration Service?

Overview of setup to import backup files from Amazon S3 to Amazon RDS

To import backup files from Amazon S3 to Amazon RDS, you need the following components:

- An Amazon S3 bucket to store your backup files.

  If you already have an Amazon S3 bucket, you can use that bucket. If you don't have an Amazon S3 bucket, create a new one. For more information, see Creating a bucket.
- A backup of your on-premises database created by Percona XtraBackup.

  For more information, see Creating your database backup.
- An AWS Identity and Access Management (IAM) role to allow Amazon RDS to access the S3 bucket.

If you already have an IAM role, you can use that role and attach trust and permissions policies to it. For more information, see Creating an IAM role manually.

If you don't have an IAM role, you have two options:

- You can manually create a new IAM role. For more information, see Creating an IAM role manually.
- You can choose for Amazon RDS to create a new IAM role for you. If you want Amazon RDS to create a new IAM role for you, follow the procedure that uses the AWS Management Console in Importing data from Amazon S3 to a new MySQL DB instance section.

Creating your database backup

Use the Percona XtraBackup software to create your backup. We recommend that you use the latest version of Percona XtraBackup. You can install Percona XtraBackup from Software Downloads on the Percona website.

> **Warning**
>
> When creating a database backup, XtraBackup might save credentials in the xtrabackup_info file. Make sure to confirm that the `tool_command` setting in the xtrabackup_info file doesn't contain any sensitive information.

The Percona XtraBackup version that you use depends on the MySQL version that you are backing up.

- **MySQL 8.4** – Use Percona XtraBackup version 8.4.
- **MySQL 8.0** – Use Percona XtraBackup version 8.0.

  > **Note**
  >
  > Percona XtraBackup 8.0.12 and higher versions support migration of all versions of MySQL 8.0. If you are migrating to RDS for MySQL 8.0.32 or higher, you must use Percona XtraBackup 8.0.12 or higher.

- **MySQL 5.7** – Use Percona XtraBackup version 2.4.

You can use Percona XtraBackup to create a full backup of your MySQL database files. Alternatively, if you already use Percona XtraBackup to back up your MySQL database files, you can upload your existing full and incremental backup directories and files.

For more information about backing up your database with Percona XtraBackup, see Percona XtraBackup - Documentation on the Percona website.

Creating a full backup with Percona XtraBackup

To create a full backup of your MySQL database files that Amazon RDS can restore from Amazon S3, use the Percona XtraBackup utility (`xtrabackup`).

For example, the following command creates a backup of a MySQL database and stores the files in the folder `/on-premises/s3-restore/backup` folder.

```
xtrabackup --backup --user=myuser --password=password --target-dir=/on-premises/s3-restore/backup
```

If you want to compress your backup into a single file—which you can split into multiple files later, if needed—you can save your backup in one of the following formats based on your MySQL version:

- **Gzip (.gz)** – For MySQL 5.7 and lower versions
- **tar (.tar)** – For MySQL 5.7 and lower versions
- **Percona xbstream (.xbstream)** – For all MySQL versions

> **Note**
>
> Percona XtraBackup 8.0 and higher only supports Percona xbstream for compression.

MySQL 5.7 and lower versions

The following command creates a backup of your MySQL database split into multiple Gzip files. Replace values with your own information.

```
xtrabackup --backup --user=my_user --password=password --stream=tar \
   --target-dir=/on-premises/s3-restore/backup | gzip - | split -d --bytes=500MB \
```

```
   - /on-premises/s3-restore/backup/backup.tar.gz
```

MySQL 5.7 and lower versions

The following command creates a backup of your MySQL database split into multiple tar files. Replace values with your own information.

```
xtrabackup --backup --user=my_user --password=password --stream=tar \
   --target-dir=/on-premises/s3-restore/backup | split -d --bytes=500MB \
   - /on-premises/s3-restore/backup/backup.tar
```

All MySQL versions

The following command creates a backup of your MySQL database split into multiple xbstream files. Replace values with your own information.

```
xtrabackup --backup --user=myuser --password=password --stream=xbstream \
   --target-dir=/on-premises/s3-restore/backup | split -d --bytes=500MB \
   - /on-premises/s3-restore/backup/backup.xbstream
```

Note

If you see the following error, it might be because you mixed file formats in your command:

```
ERROR:/bin/tar: This does not look like a tar archive
```

Using incremental backups with Percona XtraBackup

If you already use Percona XtraBackup to perform full and incremental backups of your MySQL database files, you don't need to create a full backup and upload the backup files to Amazon S3. Instead, to save time, copy your existing backup directories and files to your Amazon S3 bucket. For more information about creating incremental backups using Percona XtraBackup, see Create an incremental backup on the Percona website.

When copying your existing full and incremental backup files to an Amazon S3 bucket, you must recursively copy the contents of the base directory. Those contents include both the full backup and all incremental backup directories and files. This copy must preserve the directory structure

in the Amazon S3 bucket. Amazon RDS iterates through all files and directories. Amazon RDS uses the `xtrabackup-checkpoints` file that is included with each incremental backup to identify the base directory and to order incremental backups by log sequence number (LSN) range.

Backup considerations for Percona XtraBackup

Amazon RDS consumes your backup files based on the file name. Name your backup files with the appropriate file extension based on the file format. For example, use `.xbstream` for files stored using the Percona xbstream format.

Amazon RDS consumes your backup files in alphabetical order and also in natural number order. To ensure that your backup files are written and named in the proper order, use the `split` option when you issue the `xtrabackup` command.

Amazon RDS doesn't support partial backups created using Percona XtraBackup. You can't use the following options to create a partial backup when you back up the source files for your database:

- `--tables`
- `--tables-exclude`
- `--tables-file`
- `--databases`
- `--databases-exclude`
- `--databases-file`

Creating an IAM role manually

If you don't have an IAM role, you can create a new one manually. However, if you restore the database by using the AWS Management Console, we recommend that you choose to have Amazon RDS create this new IAM role for you. For Amazon RDS to create this role for you, follow the procedure in the Importing data from Amazon S3 to a new MySQL DB instance section.

To create a new IAM role manually for importing your database from Amazon S3, create a role to delegate permissions from Amazon RDS to your Amazon S3 bucket. When you create an IAM role, you attach trust and permissions policies. To import your backup files from Amazon S3, use trust and permissions policies similar to the following examples. For more information about creating the role, see Creating a role to delegate permissions to an AWS service.

The trust and permissions policies require that you provide an Amazon Resource Name (ARN). For more information about ARN formatting, see Amazon Resource Names (ARNs) and AWS service namespaces.

Example trust policy for importing from Amazon S3

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AssumeRoleForBackup",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Example permissions policy for importing from Amazon S3 — IAM user permissions

In the following example, replace *iam_user_id* with your own value.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3AccessRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/S3Access"
        }
    ]
```

```
}
```

Example permissions policy for importing from Amazon S3 — role permissions

In the following example, replace *amzn-s3-demo-bucket* and *prefix* with your own values.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":
    [
        {
        "Effect": "Allow",
        "Action":
            [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        {
        "Effect": "Allow",
        "Action":
            [
                "s3:GetObject"
            ],
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/prefix*"
        },
        {
        "Effect": "Allow",
        "Action":
            [
                "kms:Decrypt"
            ],
        "Resource": [
            "arn:aws:kms:us-east-1:111122223333:key/key_id*"
            ]
        }
    ]
}
```

> **Note**
>
> If you include a file name prefix, include the asterisk (*) after the prefix. If you don't want to specify a prefix, specify only an asterisk.

Importing data from Amazon S3 to a new MySQL DB instance

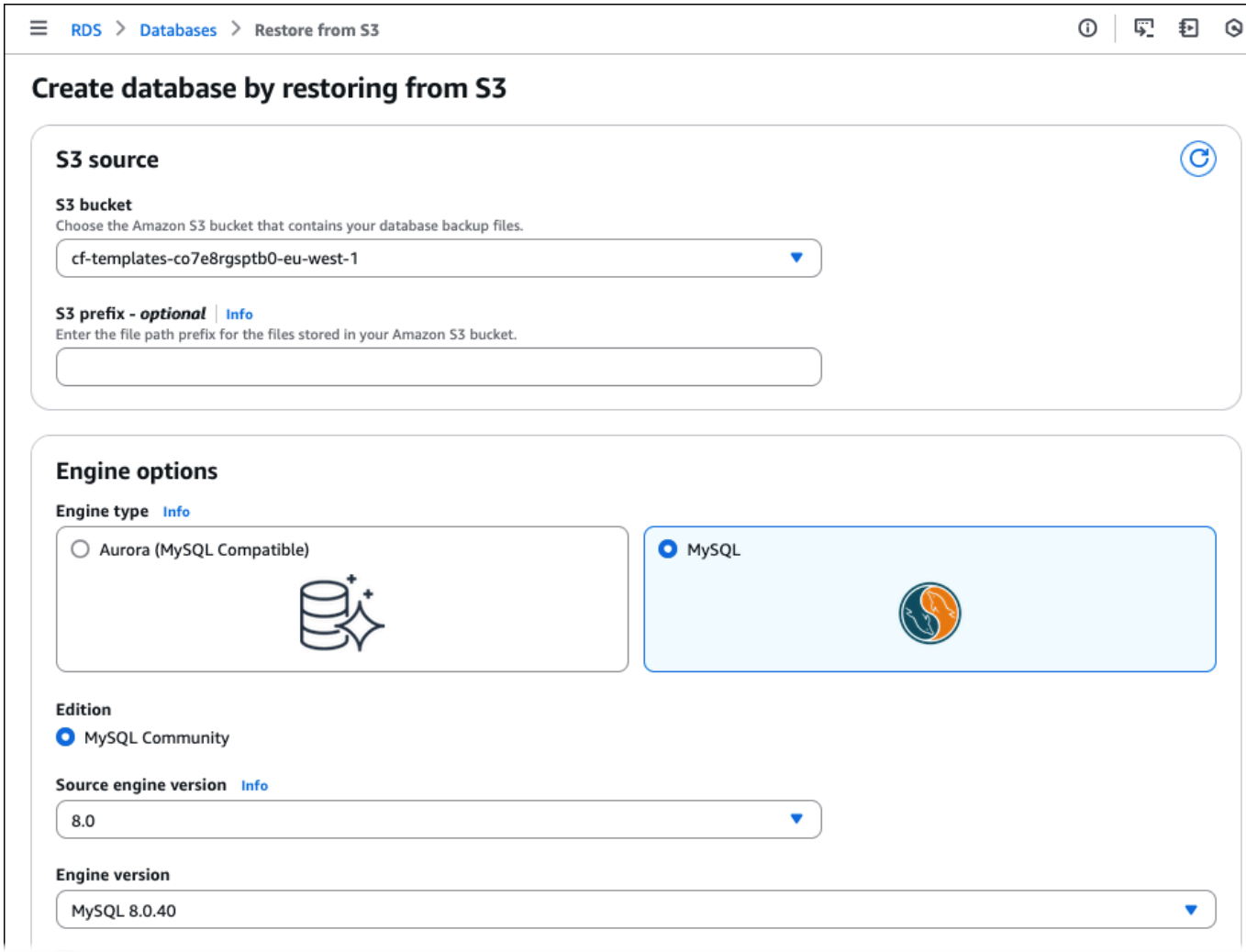
You can import data from Amazon S3 to a new MySQL DB instance using the AWS Management Console, AWS CLI, or RDS API.

Console

To import data from Amazon S3 to a new MySQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region where you want to create your DB instance. Choose the same AWS Region as the Amazon S3 bucket that contains your database backup.
3. In the navigation pane, choose **Databases**.
4. Choose **Restore from S3**.

   The **Create database by restoring from S3** page appears.



5. Under **S3 source**:

   a. Choose the **S3 bucket** that contains the backup.

   b. (Optional) For **S3 prefix**, enter the file path prefix for the files stored in your Amazon S3 bucket.

      If you don't specify a prefix, then Amazon RDS creates your DB instance using all of the files and folders in the root folder of the S3 bucket. If you do specify a prefix, then Amazon RDS creates your DB instance using the files and folders in the S3 bucket where the path for the file begins with the specified prefix.

      For example, you store your backup files on S3 in a subfolder named backups, and you have multiple sets of backup files, each in its own directory (gzip_backup1, gzip_backup2, and so on). In this case, to restore from the files in the gzip_backup1 folder, you specify the prefix backups/gzip_backup1.

6. Under **Engine options**:

   a. For **Engine type**, choose **MySQL**.

   b. For **Source engine version**, choose the MySQL major version of your source database.

   c. For **Engine Version**, choose the default minor version of your MySQL major version in your AWS Region.

      In the AWS Management Console, only the default minor version is available. After you complete the import, you can upgrade your DB instance.

7. For **IAM role**, create or choose IAM role with the required trust policy and permissions policy that allows Amazon RDS to access your Amazon S3 bucket. Perform one of the following actions:

   - (Recommended) Choose **Create a new role**, and enter the **IAM role name**. With this option, Amazon RDS automatically creates the role with the trust policy and permissions policy for you.
   - Choose an existing IAM role. Make sure that this role meets all of the criteria in the section called "Creating an IAM role manually".

8. Specify your DB instance information. For information about each setting, see Settings for DB instances.

   > **Note**
   >
   > Be sure to allocate enough storage for your new DB instance so that the restore operation can succeed.
   > To allow for future growth automatically, under **Additional storage configuration**, choose **Enable storage autoscaling**.

9. Choose additional settings as needed.

10. Choose **Create database**.

AWS CLI

To import data from Amazon S3 to a new MySQL DB instance by using the AWS CLI, run the restore-db-instance-from-s3 command with the following options. For information about each setting, see Settings for DB instances.

> **Note**
>
> Be sure to allocate enough storage for your new DB instance so that the restore operation can succeed.
>
> To enable storage autoscaling and allow for future growth automatically, use the `--max-allocated-storage` option.

- `--allocated-storage`
- `--db-instance-identifier`
- `--db-instance-class`
- `--engine`
- `--master-username`
- `--manage-master-user-password`
- `--s3-bucket-name`
- `--s3-ingestion-role-arn`
- `--s3-prefix`
- `--source-engine`
- `--source-engine-version`

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-s3 \
    --allocated-storage 250 \
    --db-instance-identifier my_identifier \
    --db-instance-class db.m5.large \
    --engine mysql \
    --master-username admin \
    --manage-master-user-password \
    --s3-bucket-name amzn-s3-demo-bucket \
    --s3-ingestion-role-arn arn:aws:iam::account-number:role/rolename \
    --s3-prefix bucket_prefix \
    --source-engine my_sql \
    --source-engine-version 8.0.32 \
```

```
    --max-allocated-storage 1000
```

For Windows:

```
aws rds restore-db-instance-from-s3 ^
    --allocated-storage 250 ^
    --db-instance-identifier my_identifier ^
    --db-instance-class db.m5.large ^
    --engine mysql ^
    --master-username admin ^
    --manage-master-user-password ^
    --s3-bucket-name amzn-s3-demo-bucket ^
    --s3-ingestion-role-arn arn:aws:iam::account-number:role/rolename ^
    --s3-prefix bucket_prefix ^
    --source-engine mysql ^
    --source-engine-version 8.0.32 ^
    --max-allocated-storage 1000
```

RDS API

To import data from Amazon S3 to a new MySQL DB instance by using the Amazon RDS API, call the RestoreDBInstanceFromS3 operation.

Limitations and considerations for importing backup files from Amazon S3 to Amazon RDS

The following limitations and considerations apply to importing backup files from Amazon S3 to an RDS for MySQL DB instance:

- You can only migrate your data to a new DB instance, not to an existing DB instance.
- You must use Percona XtraBackup to back up your data to Amazon S3. For more information, see Creating your database backup.
- The Amazon S3 bucket and the RDS for MySQL DB instance must be in the same AWS Region.
- You can't restore from the following sources:
  - A DB instance snapshot export to Amazon S3. You also can't migrate data from a DB instance snapshot export to your Amazon S3 bucket.
  - An encrypted source database. However, you can encrypt the data being migrated. You can also leave the data unencrypted during the migration process.
  - A MySQL 5.5 or 5.6 database.

- RDS for MySQL doesn't support Percona Server for MySQL as a source database because it can contain `compression_dictionary*` tables in the `mysql schema`.
- RDS for MySQL doesn't support backward migration for either major versions or minor versions. For example, you can't migrate from MySQL version 8.0 to RDS for MySQL 5.7, and you can't migrate from MySQL version 8.0.32 to RDS for MySQL version 8.0.26.
- Amazon RDS doesn't support importing on the db.t2.micro DB instance class from Amazon S3. However, you can restore to a different DB instance class, and then change the DB instance class later. For more information about instance classes, see Hardware specifications for DB instance classes.
- Amazon S3 limits the size of a file uploaded to an Amazon S3 bucket to 5 TB. If a backup file exceeds 5 TB, then you must split the backup file into smaller files.
- Amazon RDS limits the number of files uploaded to an Amazon S3 bucket to 1 million. If the backup data for your database, including all full and incremental backups, exceeds 1 million files, use a Gzip (.gz), tar (.tar.gz), or Percona xbstream (.xbstream) file to store full and incremental backup files in the Amazon S3 bucket. Percona XtraBackup 8.0 only supports Percona xbstream for compression.
- To provide management services for each DB instance, Amazon RDS creates the `rdsadmin` user when it creates the DB instance. Because `rdsamin` is a reserved user in Amazon RDS, the following limitations apply:
  - Amazon RDS doesn't import functions, procedures, views, events, and triggers with the `'rdsadmin'@'localhost'` definer. For more information, see Stored objects with 'rdsamin'@'localhost' as the definer and Master user account privileges.
  - When creating the DB instance, Amazon RDS creates a master user with the maximum supported privileges. When restoring from backup, Amazon RDS automatically removes any unsupported privileges assigned to users being imported.

    To identify users that might be affected by this, see User accounts with unsupported privileges. For more information on supported privileges in RDS for MySQL, see Role-based privilege model for RDS for MySQL.
- Amazon RDS doesn't migrate user-created tables in the `mysql` schema.
- You must configure the `innodb_data_file_path` parameter with only one data file that uses the default data file name `ibdata1:12M:autoextend`. You can migrate databases with two data files, or with a data file with a different name, using this method.

  The following examples are file names that Amazon RDS doesn't allow:

- `innodb_data_file_path=ibdata1:50M`
- `ibdata2:50M:autoextend`
- `innodb_data_file_path=ibdata01:50M:autoextend`

- You can't migrate from a source database that has tables defined outside of the default MySQL data directory.
- The maximum supported size for uncompressed backups using this method is limited to 64 TiB. For compressed backups, this limit is lower to account for uncompression space requirements. In such cases, the maximum supported backup size is `64 TiB - compressed backup size`.

  For information about the maximum supported database size that RDS for MySQL supports, see General Purpose SSD storage and Provisioned IOPS SSD storage.
- Amazon RDS doesn't support the importing of MySQL and other external components and plugins.
- Amazon RDS doesn't restore everything from your database. We recommend that you save the database schema and values for the following items from your source MySQL system database, and then add them to your restored RDS for MySQL DB instance after it has been created:
  - User accounts
  - Functions
  - Stored procedures
  - Time zone information. Time zone information is loaded from the local operating system of your RDS for MySQL DB instance. For more information, see Local time zone for MySQL DB instances.

Stored objects with 'rdsamin'@'localhost' as the definer

Amazon RDS doesn't import functions, procedures, views, events, and triggers with `'rdsadmin'@'localhost'` as the definer.

You can use the following SQL script on your source MySQL database to list the stored objects that have the unsupported definer.

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.

SELECT
    ROUTINE_SCHEMA,
    ROUTINE_NAME
```

```
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.
SELECT
    TABLE_SCHEMA,
    TABLE_NAME
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

User accounts with unsupported privileges

User accounts with privileges that RDS for MySQL doesn't supported are imported without the unsupported privileges. For the list of supported privileges, see Role-based privilege model for RDS for MySQL.

You can run the following SQL query on your source database to list the user accounts that have unsupported privileges.

```
SELECT
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Importing data from an external MySQL database to an Amazon RDS for MySQL DB instance

You can import data from an existing MySQL database to an RDS for MySQL DB instance. You do so by copying the database with mysqldump and piping it directly into the RDS for MySQL DB instance. The `mysqldump` command line utility is commonly used to make backups and transfer data from one MySQL server to another. It's included with MySQL client software.

> **Note**
>
> If you are importing or exporting large amounts of data with a MySQL DB instance, it's more reliable and faster to move data in and out of Amazon RDS by using `xtrabackup` backup files and Amazon S3. For more information, see Restoring a backup into an Amazon RDS for MySQL DB instance.

A typical `mysqldump` command to move data from an external database to an Amazon RDS DB instance looks similar to the following example. Replace values with your own information.

```
mysqldump -u local_user \
    --databases database_name \
    --single-transaction \
    --compress \
    --order-by-primary  \
    --routines=0 \
    --triggers=0 \
    --events=0 \
    -plocal_password | mysql -u RDS_user \
```

```
        --port=port_number \
        --host=host_name \
        -pRDS_password
```

> **Important**
>
> Make sure not to leave a space between the `-p` option and the entered password. As a security best practice, specify credentials other than the prompts shown in this example.

Make sure that you're aware of the following recommendations and considerations:

- Exclude the following schemas from the dump file:
  - `sys`
  - `performance_schema`
  - `information_schema`

  The `mysqldump` utility excludes these schemas by default.
- If you need to migrate users and privileges, consider using a tool that generates the data control language (DCL) for recreating them, such as the pt-show-grants utility.
- To perform the import, make sure the user doing so has access to the DB instance. For more information, see Controlling access with security groups.

The parameters used are as follows:

- `-u` *`local_user`* – Use to specify a user name. In the first usage of this parameter, specify the name of a user account on the local MySQL database that you identify with the `--databases` parameter.
- `--databases` *`database_name`* – Use to specify the name of the database on the local MySQL instance that you want to import into Amazon RDS.
- `--single-transaction` – Use to ensure that all of the data loaded from the local database is consistent with a single point in time. If there are other processes changing the data while `mysqldump` is reading it, using this parameter helps maintain data integrity.
- `--compress` – Use to reduce network bandwidth consumption by compressing the data from the local database before sending it to Amazon RDS.

- `--order-by-primary` – Use to reduce load time by sorting each table's data by its primary key.
- `--routines` – Use if routines such as stored procedures or functions exist in the database that you are copying. Set the parameter to `0`, which excludes the routines during the import process. Then later manually recreate the routines in the Amazon RDS database.
- `--triggers` – Use if triggers exist in the database that you are copying. Set the parameter to `0`, which excludes the triggers during the import process. Then later manually recreate the triggers in the Amazon RDS database.
- `--events` – Use if events exist in the database that you are copying. Set the parameter to `0`, which excludes the events during the import process. Then later manually recreate the events in the Amazon RDS database.
- `-p`*`local_password`* – Use to specify a password. In the first usage of this parameter, specify the password for the user account that you identify with the first `-u` parameter.
- `-u` *`RDS_user`* – Use to specify a user name. In the second usage of this parameter, specify the name of a user account on the default database for the MySQL DB instance that you identify with the `--host` parameter.
- `--port` *`port_number`* – Use to specify the port for your MySQL DB instance. By default, this is 3306 unless you changed the value when creating the DB instance.
- `--host` *`host_name`* – Use to specify the Domain Name System (DNS) name from the Amazon RDS DB instance endpoint, for example, `myinstance.123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the DB instance details in the Amazon RDS console.
- `-p`*`RDS_password`* – Use to specify a password. In the second usage of this parameter, you specify the password for the user account identified by the second `-u` parameter.

Make sure to create any stored procedures, triggers, functions, or events manually in your Amazon RDS database. If you have any of these objects in the database that you are copying, then exclude them when you run `mysqldump`. To do so, include the following parameters with your `mysqldump` command:

- `--routines=0`
- `--triggers=0`
- `--events=0`

Example

The following example copies the `world` sample database on the local host to an RDS for MySQL DB instance. Replace values with your own information.

For Linux, macOS, or Unix:

```
sudo mysqldump -u local_user \
    --databases world \
    --single-transaction \
    --compress \
    --order-by-primary  \
    --routines=0 \
    --triggers=0 \
    --events=0 \
    -plocal_password | mysql -u rds_user \
        --port=3306 \
        --host=my_instance.123456789012.us-east-1.rds.amazonaws.com \
        -pRDS_password
```

For Windows:

Run the following command in a command prompt that has been opened by right-clicking **Command Prompt** on the Windows programs menu and choosing **Run as administrator**. Replace values with your own information.

```
mysqldump -u local_user ^
    --databases world ^
    --single-transaction ^
    --compress ^
    --order-by-primary  ^
    --routines=0 ^
    --triggers=0 ^
    --events=0 ^
    -plocal_password | mysql -u RDS_user ^
        --port=3306 ^
        --host=my_instance.123456789012.us-east-1.rds.amazonaws.com ^
        -pRDS_password
```

> **Note**
>
> As a security best practice, specify credentials other than the prompts shown in the example.

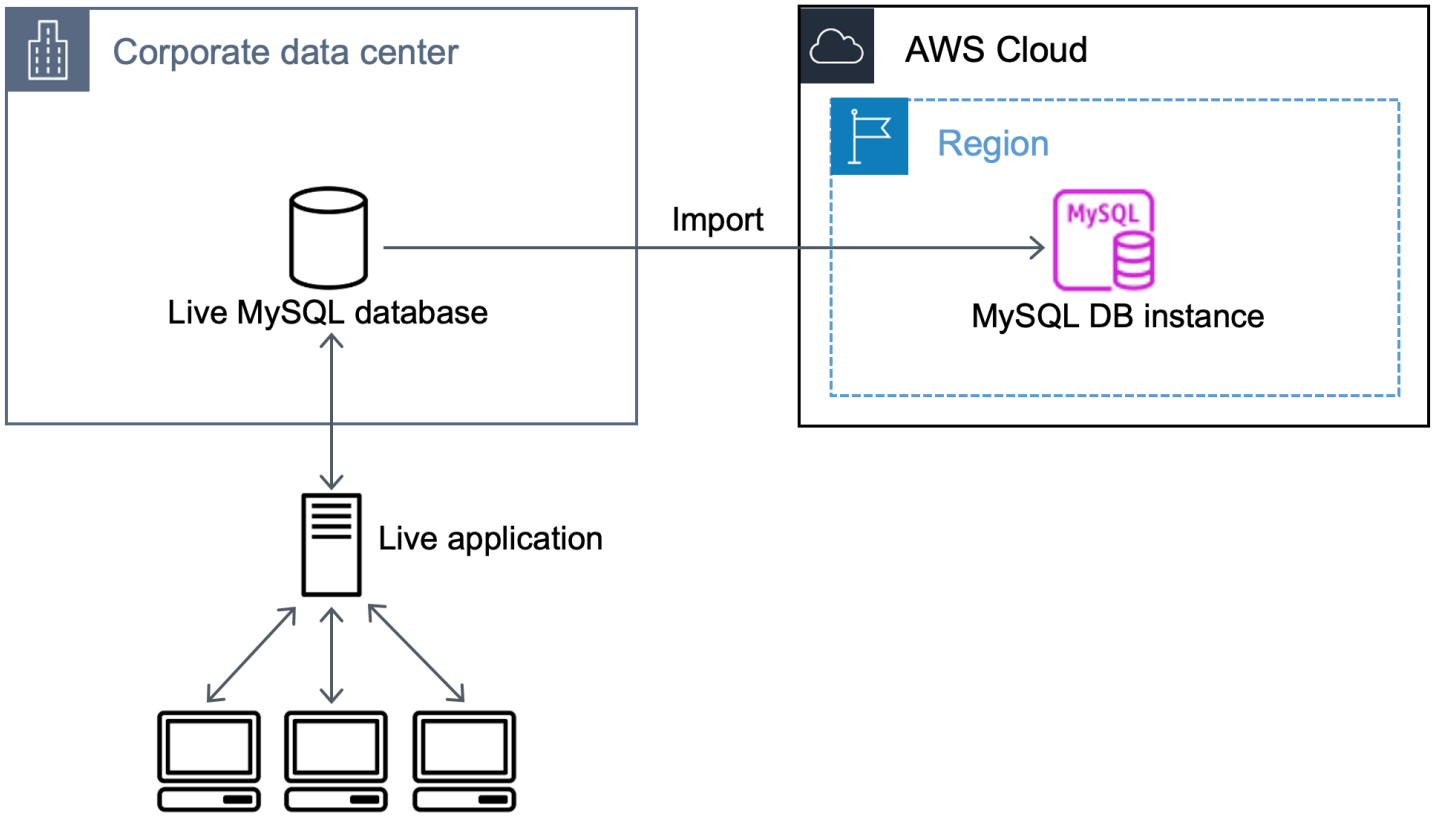
Importing data to an Amazon RDS for MySQL database with reduced downtime

In some cases, you might need to import data from an external MySQL database that supports a live application to an RDS for MySQL DB instance or an RDS for MySQL Multi-AZ DB cluster. Use the following procedure to minimize the impact on availability of applications. This procedure can also help if you are working with a very large database. Using this procedure, you can reduce the cost of the import by reducing the amount of data that is passed across the network to AWS.

In this procedure, you transfer a copy of your database data to an Amazon EC2 instance and import the data into a new Amazon RDS database. You then use replication to bring the Amazon RDS database up-to-date with your live external instance, before redirecting your application to the Amazon RDS database. Configure replication based on binary log coordinates.

> **Note**
>
> If you want to import data into an RDS for MySQL DB instance and your scenario supports it, we recommend moving data in and out of Amazon RDS by using backup files and Amazon S3. For more information, see Restoring a backup into an Amazon RDS for MySQL DB instance.

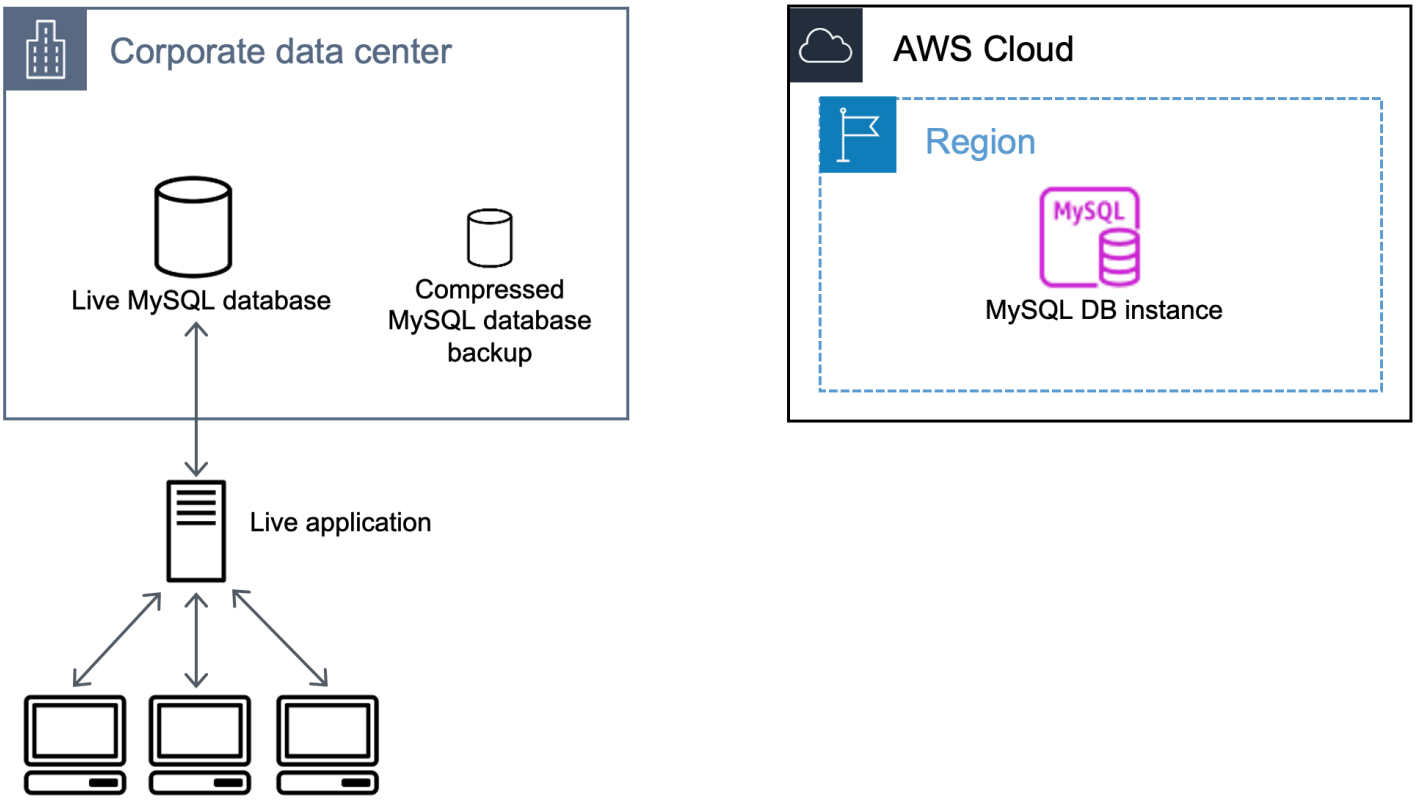
The following diagram shows importing an external MySQL database into a MySQL database on Amazon RDS.



Task 1: Create a copy of your existing database

The first step in the process of migrating a large amount of data to an RDS for MySQL database with minimal downtime is to create a copy of the source data.

The following diagram shows creating a backup of the MySQL database.



You can use the `mysqldump` utility to create a database backup in either SQL or delimited-text format. We recommend that you do a test run with each format in a non-production environment to see which method minimizes the amount of time that `mysqldump` runs.

We also recommend that you weigh `mysqldump` performance against the benefit offered by using the delimited-text format for loading. A backup using delimited-text format creates a tab-separated text file for each table being dumped. To reduce the amount of time required to import your database, you can load these files in parallel using the `LOAD DATA LOCAL INFILE` command. For more information, see Step 5: Load the data in the Importing data from any source procedure.

Before you start the backup operation, make sure to set the replication options on the MySQL database that you are copying to Amazon RDS. The replication options include turning on binary logging and setting a unique server ID. Setting these options causes your server to start logging database transactions and prepares it to be a source replication instance later in this process.

Make sure that you're aware of the following recommendations and considerations:

- Use the `--single-transaction` option with `mysqldump` because it dumps a consistent state of the database. To ensure a valid dump file, don't run data definition language (DDL) statements while `mysqldump` is running. You can schedule a maintenance window for these operations.
- Exclude the following schemas from the dump file:
  - `sys`
  - `performance_schema`
  - `information_schema`

  The `mysqldump` utility excludes these schemas by default.
- If you need to migrate users and privileges, consider using a tool that generates the data control language (DCL) for recreating them, such as the pt-show-grants utility.

To set replication options

1. Edit the `my.cnf` file. This file is usually located under `/etc`.

   ```
   sudo vi /etc/my.cnf
   ```

   Add the `log_bin` and `server_id` options to the `[mysqld]` section. The `log_bin` option provides a file name identifier for binary log files. The `server_id` option provides a unique identifier for the server in source-replica relationships.

   The following example shows the updated `[mysqld]` section of a `my.cnf` file:

   ```
   [mysqld]
   log-bin=mysql-bin
   server-id=1
   ```

   For more information, see Setting the Replication Source Configuration in the MySQL documentation.
2. For replication with a Multi-AZ DB cluster set the `ENFORCE_GTID_CONSISTENCY` and the `GTID_MODE` parameter to `ON`.

   ```
   mysql> SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = ON;
   ```

   ```
   mysql> SET @@GLOBAL.GTID_MODE = ON;
   ```

These settings aren't required for replication with a DB instance.

3. Restart the `mysql` service.

```
sudo service mysqld restart
```

To create a backup copy of your existing database

1. Create a backup of your data using the `mysqldump` utility, specifying either SQL or delimited-text format.

   For MySQL 8.0.25 and lower versions, specify `--master-data=2` to create a backup file that can be used to start replication between servers. For MySQL 8.0.26 and higher versions, specify `--source-data=2` to create a backup file that can be used to start replication between servers. For more information, see the mysqldump — A Database Backup Program in the MySQL documentation.

   To improve performance and ensure data integrity, use the `--order-by-primary` and `--single-transaction` options for `mysqldump`.

   To avoid including the MySQL system database in the backup, don't use the `--all-databases` option with `mysqldump`. For more information, see Creating a Data Snapshot Using mysqldump in the MySQL documentation.

   Use chmod, if necessary, to make sure that the directory where the backup file is being created is writeable.

   > **Important**
   >
   > On Windows, run the command window as an administrator.

   - To produce SQL output, use the following command:

     For Linux, macOS, or Unix:

     ```
     sudo mysqldump \
         --databases database_name \
         --master-data=2  \
     ```

```
    --single-transaction \
    --order-by-primary \
    -r backup.sql \
    -u local_user \
    -ppassword
```

> **Note**
>
> As a security best practice, specify credentials other than the prompts shown in the example.

For Windows:

```
mysqldump ^
    --databases database_name ^
    --master-data=2  ^
    --single-transaction ^
    --order-by-primary ^
    -r backup.sql ^
    -u local_user ^
    -ppassword
```

> **Note**
>
> As a security best practice, specify credentials other than the prompts shown in the example.

- To produce delimited-text output, use the following command:

  For Linux, macOS, or Unix:

```
sudo mysqldump \
    --tab=target_directory \
    --fields-terminated-by ',' \
    --fields-enclosed-by '"' \
    --lines-terminated-by 0x0d0a \
    database_name \
    --master-data=2 \
    --single-transaction \
```

```
    --order-by-primary \
    -ppassword
```

For Windows:

```
mysqldump ^
    --tab=target_directory ^
    --fields-terminated-by "," ^
    --fields-enclosed-by """ ^
    --lines-terminated-by 0x0d0a ^
    database_name ^
    --master-data=2 ^
    --single-transaction ^
    --order-by-primary ^
    -ppassword
```

Note

As a security best practice, specify credentials other than the prompts shown in the example.

Make sure to create any stored procedures, triggers, functions, or events manually in your Amazon RDS database. If you have any of these objects in the database that you are copying, then exclude them when you run `mysqldump`. To do so, include the following arguments with your `mysqldump` command:

- `--routines=0`
- `--triggers=0`
- `--events=0`

For MySQL 8.0.22 and lower versions, when you run `mysqldump` and specify the delimited-text format, a `CHANGE MASTER TO` comment is returned. This comment contains the master log file name and position. For MySQL 8.0.23 and higher versions, when you run `mysqldump` using the delimited-text format, a `CHANGE REPLICATION SOURCE TO` comment is returned. This comment contains the source log file name and position. If the external instance is MySQL 8.0.23 and higher versions, note the values for `MASTER_LOG_FILE` and `MASTER_LOG_POS`. You need these values when setting up replication.

The following output is returned for MySQL 8.0.22 and lower versions:

```
-- Position to start replication or point-in-time recovery from
--
-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031',
 MASTER_LOG_POS=107;
```

The following output is returned for MySQL 8.0.23 and higher versions:

```
-- Position to start replication or point-in-time recovery from
--
-- CHANGE SOURCE TO SOURCE_LOG_FILE='mysql-bin-changelog.000031',
 SOURCE_LOG_POS=107;
```

For MySQL 8.0.22 and lower versions, if you are using SQL format, you can get the master log file name and position in the `CHANGE MASTER TO` comment in the backup file. For MySQL 8.0.23 and higher versions, if you are using SQL format, you can get the source log file name and position in the `CHANGE REPLICATION SOURCE TO` comment in the backup file.

2. Compress the copied data to reduce the amount of network resources needed to copy your data to the Amazon RDS database. Note the size of the backup file. You need this information when determining how large an Amazon EC2 instance to create. When you are done, compress the backup file using GZIP or your preferred compression utility.

   - To compress SQL output, use the following command:

     ```
     gzip backup.sql
     ```

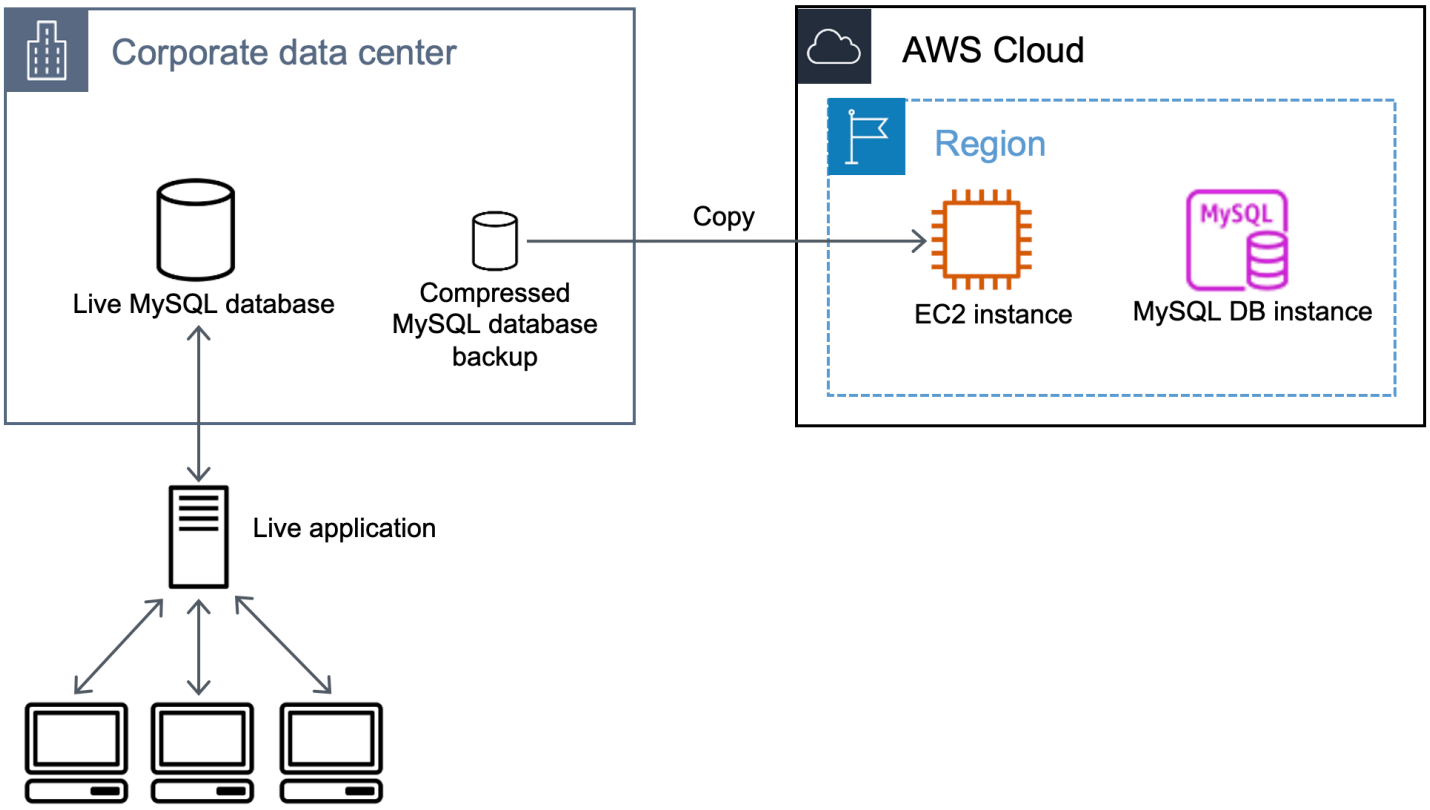
   - To compress delimited-text output, use the following command:

     ```
     tar -zcvf backup.tar.gz target_directory
     ```

Task 2: Create an Amazon EC2 instance and copy the compressed database

Copying your compressed database backup file to an Amazon EC2 instance takes fewer network resources than doing a direct copy of uncompressed data between database instances. After your data is in Amazon EC2, you can copy it from there directly to your MySQL database. For you to save on the cost of network resources, your Amazon EC2 instance must be in the same AWS Region as your Amazon RDS DB instance. Having the Amazon EC2 instance in the same AWS Region as your Amazon RDS database also reduces network latency during the import.

The following diagram shows copying the database backup to an Amazon EC2 instance.



To create an Amazon EC2 instance and copy your data

1. In the AWS Region where you plan to create the Amazon RDS database, create a virtual private cloud (VPC), a VPC security group, and a VPC subnet. Ensure that the inbound rules for your VPC security group allow the IP addresses required for your application to connect to AWS. You can specify a range of IP addresses—for example, `203.0.113.0/24`—or another VPC security group. You can use the Amazon VPC console to create and manage VPCs, subnets, and security groups. For more information, see Getting started with Amazon VPC in the *Amazon Virtual Private Cloud User Guide*.
2. Open the Amazon EC2 console and choose the AWS Region to contain both your Amazon EC2 instance and your Amazon RDS database. Launch an Amazon EC2 instance using the VPC, subnet, and security group that you created in Step 1. Ensure that you select an instance type with enough storage for your database backup file when it is uncompressed. For details on Amazon EC2 instances, see Getting started with Amazon EC2 in the *Amazon Elastic Compute Cloud User Guide*.
3. To connect to your Amazon RDS database from your Amazon EC2 instance, edit your VPC security group. Add an inbound rule specifying the private IP address of your EC2 instance.

You can find the private IP address on the **Details** tab of the **Instance** pane in the EC2 console window. To edit the VPC security group and add an inbound rule, choose **Security Groups** in the EC2 console navigation pane, choose your security group, and then add an inbound rule for MySQL or Aurora specifying the private IP address of your EC2 instance. To learn how to add an inbound rule to a VPC security group, see Security group rules in the *Amazon Virtual Private Cloud User Guide*.

4. Copy your compressed database backup file from your local system to your Amazon EC2 instance. Use `chmod`, if necessary, to make sure that you have write permission for the target directory of the Amazon EC2 instance. You can use `scp` or a Secure Shell (SSH) client to copy the file. The following command is an example `scp` command:

```
scp -r -i key pair.pem backup.sql.gz ec2-user@EC2 DNS:/target_directory/backup.sql.gz
```

> **Important**
>
> When copying sensitive data, be sure to use a secure network transfer protocol.

5. Connect to your Amazon EC2 instance and install the latest updates and the MySQL client tools using the following commands:

```
sudo yum update -y
sudo yum install mysql -y
```

For more information, see Connect to your instance for Linux instances in the *Amazon Elastic Compute Cloud User Guide*.

> **Important**
>
> This example installs the MySQL client on an Amazon Machine Image (AMI) for an Amazon Linux distribution. This example doesn't install the MySQL client on a different distribution, such as Ubuntu or Red Hat Enterprise Linux. For information about installing MySQL, see Installing MySQL in the MySQL documentation.

6. While connected to your Amazon EC2 instance, decompress your database backup file. The following commands are examples.

   - To decompress SQL output, use the following command:

```
gzip backup.sql.gz -d
```

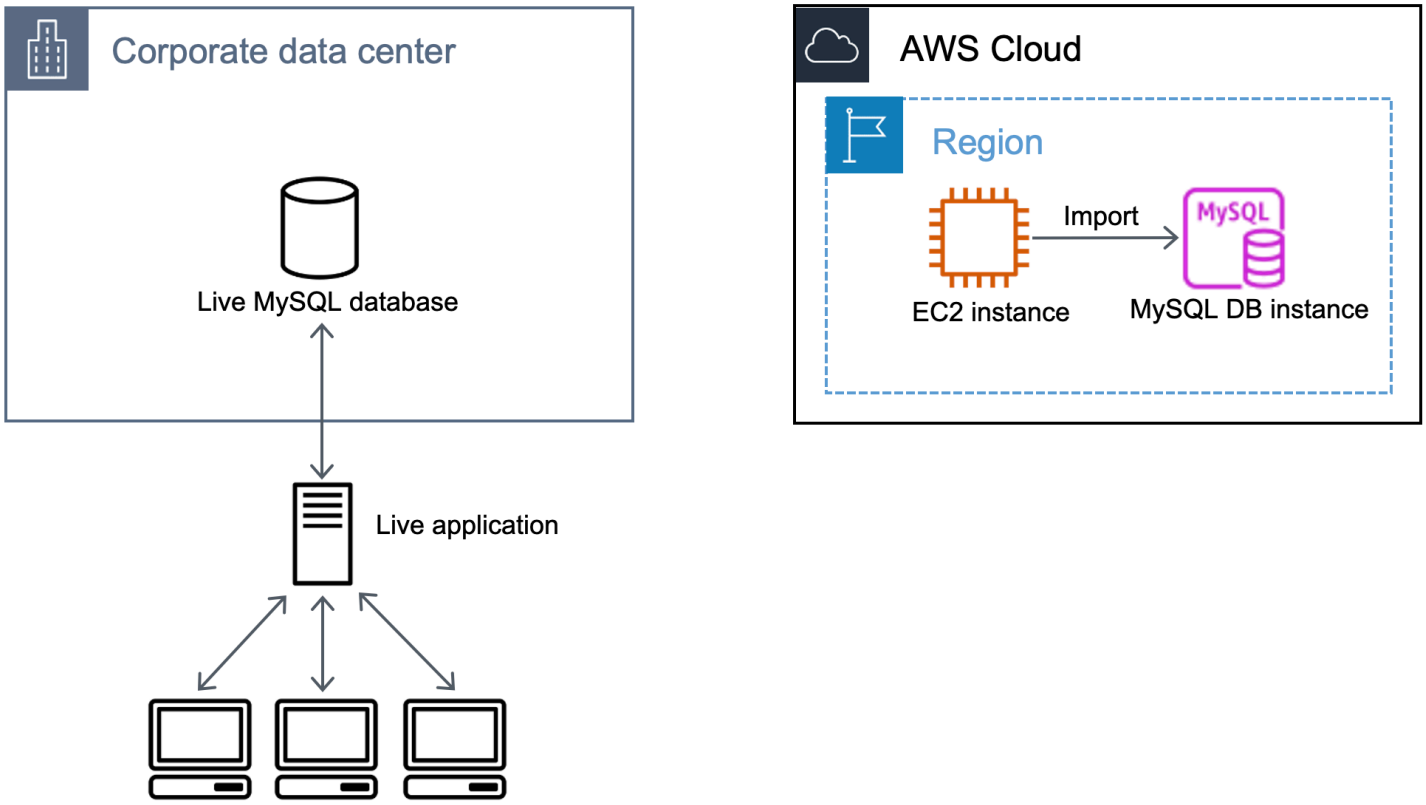
- To decompress delimited-text output, use the following command:

```
tar xzvf backup.tar.gz
```

Task 3: Create a MySQL database and import data from your Amazon EC2 instance

By creating an RDS for MySQL DB instance or an RDS for MySQL Multi-AZ DB cluster in the same AWS Region as your Amazon EC2 instance, you can import the database backup file from Amazon EC2 faster than over the internet.

The following diagram shows importing the backup from an Amazon EC2 instance into a MySQL database.



To create a MySQL database and import your data

1. Determine which DB instance class and what amount of storage space is required to support the expected workload for this Amazon RDS database. As part of this process, decide what is sufficient space and processing capacity for your data load procedures. Also, decide what is required to handle the production workload. You can estimate this based on the size and resources of the source MySQL database. For more information, see DB instance classes.
2. Create a DB instance or Multi-AZ DB cluster in the AWS Region that contains your Amazon EC2 instance.

   To create an RDS for MySQL Multi-AZ DB cluster, follow the instructions in Creating a Multi-AZ DB cluster for Amazon RDS.

   To create an RDS for MySQL DB instance, follow the instructions in Creating an Amazon RDS DB instance and use the following guidelines:

   - Specify a DB engine version that is compatible with your source DB instance.
   - Specify the same virtual private cloud (VPC) and VPC security group as for your Amazon EC2 instance. This approach ensures that your Amazon EC2 instance and your Amazon RDS instance are visible to each other over the network. Make sure your DB instance is publicly accessible. To set up replication with your source database as described in a following section, your DB instance must be publicly accessible.
   - Don't configure multiple Availability Zones, backup retention, or read replicas until after you have imported the database backup. When that import is completed, you can configure Multi-AZ and backup retention for the production instance.
3. Review the default configuration options for the Amazon RDS database. If the default parameter group for the database doesn't have the configuration options that you want, find a different one that does or create a new parameter group. For more information about creating a parameter group, see Parameter groups for Amazon RDS.
4. Connect to the new Amazon RDS database as the master user. Create the users required to support the administrators, applications, and services that need to access the DB instance. The hostname for the Amazon RDS database is the **Endpoint** value for this DB instance without the port number, for example, `mysampledb.123456789012.us-west-2.rds.amazonaws.com`. You can find the endpoint value in the database details in the Amazon RDS console.
5. Connect to your Amazon EC2 instance. For more information, see Connect to your instance for Linux instances in the *Amazon Elastic Compute Cloud User Guide*.

6. Connect to your Amazon RDS database as a remote host from your Amazon EC2 instance using the `mysql` command. The following command is an example:

   ```
   mysql -h host_name -P 3306 -u db_master_user -p
   ```

   The *host_name* is the Amazon RDS database endpoint.

7. At the `mysql` prompt, run the `source` command and pass it the name of your database dump file. This command loads the data into the Amazon RDS DB instance.

   - For SQL format, use the following command:

     ```
     mysql> source backup.sql;
     ```

   - For delimited-text format, first create the database, if it isn't the default database that you created when setting up the Amazon RDS database.

     ```
     mysql> create database database_name;
     mysql> use database_name;
     ```

     Then create the tables.

     ```
     mysql> source table1.sql
     mysql> source table2.sql
     etc...
     ```

     Then import the data.

     ```
     mysql> LOAD DATA LOCAL INFILE 'table1.txt' INTO TABLE table1 FIELDS TERMINATED BY
      ',' ENCLOSED BY '"' LINES TERMINATED BY '0x0d0a';
     mysql> LOAD DATA LOCAL INFILE 'table2.txt' INTO TABLE table2 FIELDS TERMINATED BY
      ',' ENCLOSED BY '"' LINES TERMINATED BY '0x0d0a';
     etc...
     ```

     To improve performance, you can perform these operations in parallel from multiple connections so that all of your tables are created and then loaded at the same time.

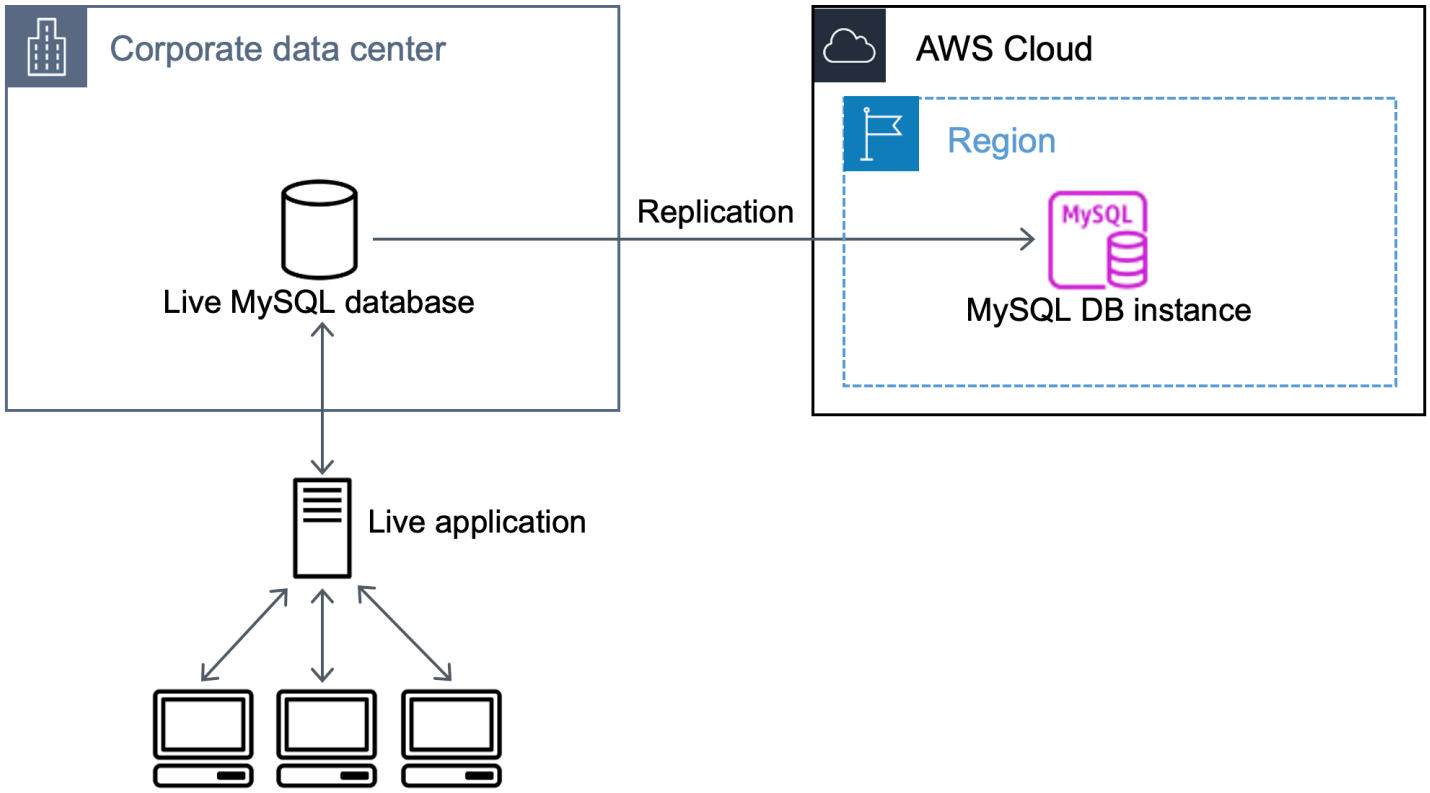
> **Note**
>
> If you used any data-formatting options with `mysqldump` when you initially dumped the table, make sure to use the same options with `LOAD DATA LOCAL INFILE` to ensure proper interpretation of the data file contents.

8. Run a simple `SELECT` query against one or two of the tables in the imported database to verify that the import was successful.

If you no longer need the Amazon EC2 instance used in this procedure, terminate the EC2 instance to reduce your AWS resource usage. To terminate an EC2 instance, see Terminate an instance in the *Amazon Elastic Compute Cloud User Guide*.

Task 4: Replicate data from your external database to your new Amazon RDS database

Your source database was likely updated during the time that it took to copy and transfer the data to the MySQL database. Thus, you can use replication to bring the copied database up-to-date with the source database.



The permissions required to start replication on an Amazon RDS database are restricted and aren't available to your Amazon RDS master user. Because of this, use the appropriate Amazon RDS stored procedure for your major engine version:

- mysql_rds_set_external_master (RDS for MySQL major versions 8.0 and lower)
- mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher)
- mysql.rds_set_external_master_gtid to configure replication and mysql.rds_start_replication to start replication

To start replication

In Task 1, when you set replication options, you turned on binary logging and set a unique server ID for your source database. Now, you can set up your Amazon RDS database as a replica with your live database as the source replication instance.

1. In the Amazon RDS console, add the IP address of the server that hosts the source database to the VPC security group for the Amazon RDS database. For more information on configuring a VPC security group, see Configure security group rules in the *Amazon Virtual Private Cloud User Guide*.

   You might also need to configure your local network to permit connections from the IP address of your Amazon RDS database so that it can communicate with your source instance. To find the IP address of the Amazon RDS database, use the `host` command:

   ```
   host host_name
   ```

   The *`host_name`* is the DNS name from the Amazon RDS database endpoint, for example `myinstance.123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the DB instance details in the Amazon RDS console.

2. Using the client of your choice, connect to the source instance and create a user to be used for replication. This account is used solely for replication and must be restricted to your domain to improve security. The following command is an example:

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

> **Note**
>
> Specify credentials other than the prompts shown here as a security best practice.

3. For the source instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. For example, to grant the `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the '`repl_user`' user for your domain, issue the following command:

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

4. Make the Amazon RDS database the replica. Connect to the Amazon RDS database as the master user and identify the source database as the source replication instance by using the appropriate Amazon RDS stored procedure:

   - mysql_rds_set_external_master (RDS for MySQL major versions 8.0 and lower)
   - mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher)

   If you have a SQL format backup file, use the master log file name and master log position that you determined in Step 4. If you used delimited-text format, use the name and position that you determined when creating the backup files. The following commands are examples:

   MySQL 8.4 and higher versions

```
CALL mysql.rds_set_external_source ('myserver.mydomain.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

   MySQL 8.0 and lower versions

```
CALL mysql.rds_set_external_master ('myserver.mydomain.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

> **Note**
>
> Specify credentials other than the prompts shown here as a security best practice.

5. On the Amazon RDS database, to start replication, run the following command that uses the mysql.rds_start_replication stored procedure:

```
CALL mysql.rds_start_replication;
```

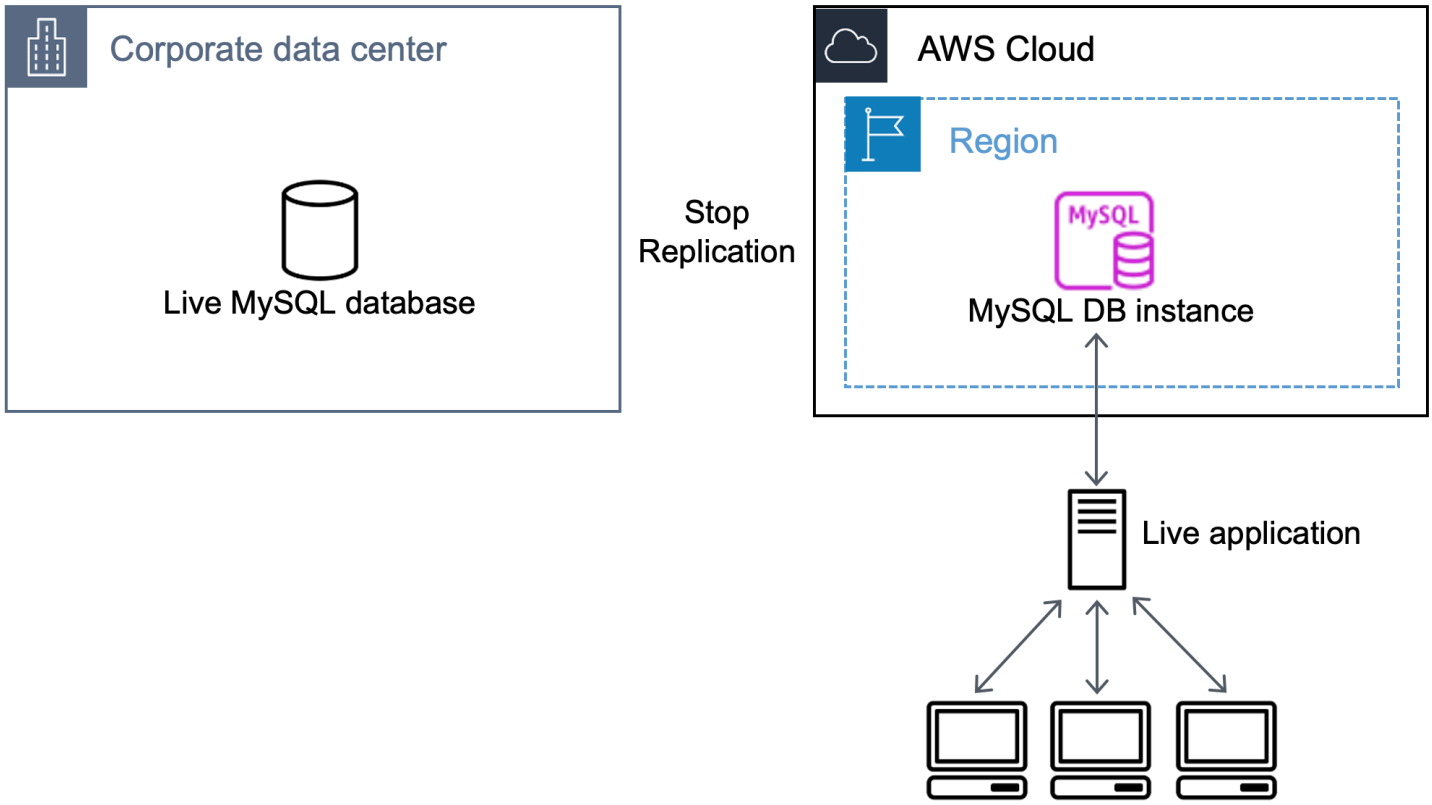
6. On the Amazon RDS database, to determine when the replica is up to date with the source replication instance, run the SHOW REPLICA STATUS command. The results of the `SHOW REPLICA STATUS` command include the `Seconds_Behind_Master` field. When the `Seconds_Behind_Master` field returns 0, then the replica is up to date with the source replication instance.

   > **Note**
   >
   > Previous versions of MySQL used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MySQL version before 8.0.23, then use `SHOW SLAVE STATUS`.

7. After the Amazon RDS database is up to date, turn on automated backups so you can restore that database if needed. You can turn on or modify automated backups for your Amazon RDS database by using the Amazon RDS console. For more information, see Introduction to backups.

Task 5: Redirect your live application to your Amazon RDS instance

After the MySQL database is up to date with the source replication instance, you can now update your live application to use the Amazon RDS instance.



To redirect your live application to your MySQL database and stop replication

1. To add the VPC security group for the Amazon RDS database, add the IP address of the server that hosts the application. For more information on modifying a VPC security group, see Configure security group rules in the *Amazon Virtual Private Cloud User Guide*.
2. Verify that the `Seconds_Behind_Master` field in the SHOW REPLICA STATUS command results is 0, which indicates that the replica is up to date with the source replication instance.

   ```
   SHOW REPLICA STATUS;
   ```

   > **Note**
   >
   > Previous versions of MySQL used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MySQL version before 8.0.23, then use `SHOW SLAVE STATUS`.

3. Close all connections to the source when their transactions complete.
4. Update your application to use the Amazon RDS database. This update typically involves changing the connection settings to identify the hostname and port of the Amazon RDS database, the user account and password to connect with, and the database to use.

5. Connect to the DB instance.

   For a Multi-AZ DB cluster, connect to the writer DB instance.

6. Stop replication for the Amazon RDS instance by running the following command that uses the mysql.rds_stop_replication stored procedure:

```
CALL mysql.rds_stop_replication;
```

7. Reset the replication configuration so this instance is no longer identified as a replica by using the appropriate Amazon RDS stored procedure on your Amazon RDS database:
   - mysql_rds_reset_external_master (RDS for MySQL major versions 8.0 and lower)
   - mysql.rds_reset_external_source (RDS for MySQL major versions 8.4 and higher)

   MySQL 8.4 and higher versions

```
CALL mysql.rds_reset_external_source;
```

   MySQL 8.0 and lower versions

```
CALL mysql.rds_reset_external_master;
```

8. Turn on additional Amazon RDS features such as Multi-AZ support and read replicas. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS and Working with DB instance read replicas.

Importing data from any source to an Amazon RDS for MySQL DB instance

With Amazon RDS, you can migrate existing MySQL data from any source to an RDS for MySQL DB instance. You can transfer data from on-premises databases, other cloud providers, or existing RDS for MySQL DB instances to your target RDS for MySQL DB instance. With this functionality, you can consolidate databases, implement disaster recovery solutions, or transition from self-managed databases. Common scenarios include moving from self-hosted MySQL servers to fully managed Amazon RDS DB instances, consolidating multiple MySQL databases into a single DB instance, or creating test environments with production data. The following sections provide step-by-step instructions for importing your MySQL data using methods such as `mysqldump`, backup files, or replication.

Step 1: Create flat files containing the data to be loaded

Use a common format, such as comma-separated values (CSV), to store the data to be loaded. Each table must have its own file—you can't combine data for multiple tables in the same file. Give each file the same name as the table it corresponds to. The file extension can be anything you like. For example, if the table name is `sales`, the file name could be `sales.csv` or `sales.txt`.

If possible, order the data by the primary key of the table being loaded. Doing this drastically improves load times and minimizes disk storage requirements.

The speed and efficiency of this procedure depends on keeping the size of the files small. If the uncompressed size of any individual file is larger than 1 GiB, split it into multiple files and load each one separately.

On Unix-like systems (including Linux), use the `split` command. For example, the following command splits the `sales.csv` file into multiple files of less than 1 GiB, splitting only at line breaks (-C 1024m). The names of the new files include ascending numerical suffixes. The following command produces files with names such as `sales.part_00` and `sales.part_01`.

```
split -C 1024m -d sales.csv sales.part_
```

Similar utilities are available for other operating systems.

You can store the flat files anywhere. However, when you load the data in Step 5, you must invoke the `mysql` shell from the same location where the files exist, or use the absolute path for the files when you run `LOAD DATA LOCAL INFILE`.

Step 2: Stop any applications from accessing the target DB instance

Before starting a large load, stop all application activity from accessing the target DB instance that you plan to load to. We recommend this particularly if other sessions will be modifying the tables being loaded or tables that they reference. Doing this reduces the risk of constraint violations occurring during the load and improves load performance. It also makes it possible to restore the DB instance to the point just before the load without losing changes made by processes not involved in the load.

Of course, this might not be possible or practical. If you can't stop applications from accessing the DB instance before the load, take steps to ensure the availability and integrity of your data. The specific steps required vary greatly depending upon specific use cases and site requirements.

Step 3: Create a DB snapshot

If you plan to load data into a new DB instance that contains no data, you can skip this step. Otherwise, we recommend that you create DB snapshots of the target Amazon RDS DB instance both before and after the data load. Amazon RDS DB snapshots are complete backups of your DB instance that you can use to restore your DB instance to a known state. When you initiate a DB snapshot, I/O operations to your DB instance are momentarily suspended while your database is backed up.

Creating a DB snapshot immediately before the load makes it possible for you to restore the database to its state before the load, if you need to. A DB snapshot taken immediately after the load protects you from having to load the data again in case of a mishap. You can also use DB snapshots after the load to import data into new database instances.

The following example runs the AWS CLI create-db-snapshot command to create a DB snapshot of the AcmeRDS instance and give the DB snapshot the identifier "preload".

For Linux, macOS, or Unix:

```
aws rds create-db-snapshot \
    --db-instance-identifier AcmeRDS \
    --db-snapshot-identifier preload
```

For Windows:

```
aws rds create-db-snapshot ^
    --db-instance-identifier AcmeRDS ^
    --db-snapshot-identifier preload
```

You can also use the restore from DB snapshot functionality to create test DB instances for dry runs or to undo changes made during the load.

Keep in mind that restoring a database from a DB snapshot creates a new DB instance that, like all DB instances, has a unique identifier and endpoint. To restore the DB instance without changing the endpoint, first delete the DB instance so that you can reuse the endpoint.

For example, to create a DB instance for dry runs or other testing, you give the DB instance its own identifier. In the example, AcmeRDS-2" is the identifier. The example connects to the DB instance using the endpoint associated with AcmeRDS-2. For more information, see restore-db-instance-from-db-snapshot.

For Linux, macOS, or Unix:

```
aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier AcmeRDS-2 \
    --db-snapshot-identifier preload
```

For Windows:

```
aws rds restore-db-instance-from-db-snapshot ^
    --db-instance-identifier AcmeRDS-2 ^
    --db-snapshot-identifier preload
```

To reuse the existing endpoint, first delete the DB instance and then give the restored database the same identifier. For more information, see delete-db-instance.

The following example also takes a final DB snapshot of the DB instance before deleting it. This is optional but recommended.

For Linux, macOS, or Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier AcmeRDS \
    --final-db-snapshot-identifier AcmeRDS-Final

aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier AcmeRDS \
    --db-snapshot-identifier preload
```

For Windows:

```
aws rds delete-db-instance ^
    --db-instance-identifier AcmeRDS ^
    --final-db-snapshot-identifier AcmeRDS-Final

aws rds restore-db-instance-from-db-snapshot ^
    --db-instance-identifier AcmeRDS ^
    --db-snapshot-identifier preload
```

Step 4 (Optional): Turn off Amazon RDS automated backups

> **Warning**
>
> Don't turn off automated backups if you need to perform point-in-time recovery.

Turning off automated backups is a performance optimization and isn't required for data loads. Turning off automated backups erases all existing backups. As a result, after you turn off automated backups, point-in-time recovery isn't possible. Manual DB snapshots aren't affected by turning off automated backups. All existing manual DB snapshots are still available for restore.

Turning off automated backups reduces load time by about 25 percent and reduces the amount of storage space required during the load. If you plan to load data into a new DB instance that contains no data, turning off backups is an easy way to speed up the load and avoid using the additional storage needed for backups. However, in some cases you might plan to load into a DB instance that already contains data. If so, weigh the benefits of turning off backups against the impact of losing the ability to perform point-in-time-recovery.

DB instances have automated backups turned on by default (with a one day retention period). To turn off automated backups, set the backup retention period to zero. After the load, you can turn backups back on by setting the backup retention period to a nonzero value. To turn on or turn off backups, Amazon RDS shuts the DB instance down and then restarts it to turn MySQL logging on or off.

Run the AWS CLI `modify-db-instance` command to set the backup retention to zero and apply the change immediately. Setting the retention period to zero requires a DB instance restart, so wait until the restart has completed before proceeding. For more information, see modify-db-instance.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier AcmeRDS \
    --apply-immediately \
    --backup-retention-period 0
```

For Windows:

```
aws rds modify-db-instance ^
```

```
    --db-instance-identifier AcmeRDS ^
    --apply-immediately ^
    --backup-retention-period 0
```

You can check the status of your DB instance with the AWS CLI describe-db-instances command. The following example displays the DB instance status of the AcmeRDS DB instance:

```
aws rds describe-db-instances --db-instance-identifier AcmeRDS --query "*[].
{DBInstanceStatus:DBInstanceStatus}"
```

When the DB instance status is `available`, you're ready to proceed to the next step.

Step 5: Load the data

To read rows from your flat files into the database tables, use the MySQL `LOAD DATA LOCAL INFILE` statement.

> **Note**
>
> You must invoke the `mysql` shell from the same location where your flat files exist, or use the absolute path for the files when you run `LOAD DATA LOCAL INFILE`.

The following example shows how to load data from a file named `sales.txt` into a table named `Sales` in the database:

```
mysql> LOAD DATA LOCAL INFILE 'sales.txt' INTO TABLE Sales FIELDS TERMINATED BY ' '
 ENCLOSED BY '' ESCAPED BY '\\';
Query OK, 1 row affected (0.01 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

For more information about the `LOAD DATA` statement, see LOAD DATA Statement in the MySQL documentation.

Step 6: Turn back on Amazon RDS automated backups

If you turned off Amazon RDS automated backups in Step 4, after the load is finished, turn automated backups on by setting the backup retention period back to its preload value. As noted in Step 4, Amazon RDS restarts the DB instance, so be prepared for a brief outage.

The following example runs the AWS CLI modify-db-instance command to turn on automated backups for the AcmeRDS DB instance and set the retention period to one day:

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier AcmeRDS \
    --backup-retention-period 1 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier AcmeRDS ^
    --backup-retention-period 1 ^
    --apply-immediately
```

Working with MySQL replication in Amazon RDS

You usually use read replicas to configure replication between Amazon RDS DB instances. For general information about read replicas, see Working with DB instance read replicas. For specific information about working with read replicas on Amazon RDS for MySQL, see Working with MySQL read replicas.

You can use global transaction identifiers (GTIDs) for replication with RDS for MySQL. For more information, see Using GTID-based replication.

You can also set up replication between an RDS for MySQL DB instance and a MariaDB or MySQL instance that is external to Amazon RDS. For information about configuring replication with an external source, see Configuring binary log file position replication with an external source instance.

For any of these replication options, you can use either row-based replication, statement-based, or mixed replication. Row-based replication only replicates the changed rows that result from a SQL statement. Statement-based replication replicates the entire SQL statement. Mixed replication uses statement-based replication when possible, but switches to row-based replication when SQL statements that are unsafe for statement-based replication are run. In most cases, mixed replication is recommended. The binary log format of the DB instance determines whether replication is row-based, statement-based, or mixed. For information about setting the binary log format, see Configuring RDS for MySQL binary logging for Single-AZ databases.

> **Note**
>
> You can configure replication to import databases from a MariaDB or MySQL instance that is external to Amazon RDS, or to export databases to such instances. For more information, see Importing data to an Amazon RDS for MySQL database with reduced downtime and Exporting data from a MySQL DB instance by using replication.

After you restore your DB instance from a snapshot or perform a point-in-time recovery, you can view the last recovered binlog position from the source database in the RDS console. Under **Logs & events**, enter **binlog**. The binlog position appears under **System notes**.

Topics

- Working with MySQL read replicas

- Using GTID-based replication
- Configuring binary log file position replication with an external source instance
- Configuring multi-source-replication for Amazon RDS for MySQL

Working with MySQL read replicas

Following, you can find specific information about working with read replicas on RDS for MySQL. For general information about read replicas and instructions for using them, see Working with DB instance read replicas.

For more information about MySQL read replicas, see the following topics.

- Configuring replication filters with MySQL
- Configuring delayed replication with MySQL
- Updating read replicas with MySQL
- Working with Multi-AZ read replica deployments with MySQL
- Using cascading read replicas with RDS for MySQL
- Monitoring replication lag for MySQL read replicas
- Starting and stopping replication with MySQL read replicas
- Troubleshooting a MySQL read replica problem

Configuring read replicas with MySQL

Before a MySQL DB instance can serve as a replication source, make sure to enable automatic backups on the source DB instance. To do this, set the backup retention period to a value other than 0. This requirement also applies to a read replica that is the source DB instance for another read replica. Automatic backups are supported for read replicas running any version of MySQL. You can configure replication based on binary log coordinates for a MySQL DB instance.

You can configure replication using global transaction identifiers (GTIDS) on the following versions:

- RDS for MySQL version 5.7.44 and higher 5.7 versions
- RDS for MySQL version 8.0.28 and higher 8.0 versions
- RDS for MySQL version 8.4.3 and higher 8.4 versions

For more information, see Using GTID-based replication.

You can create up to 15 read replicas from one DB instance within the same Region. For replication to operate effectively, each read replica should have the same amount of compute and storage resources as the source DB instance. If you scale the source DB instance, also scale the read replicas.

RDS for MySQL supports cascading read replicas. To learn how to configure cascading read replicas, see Using cascading read replicas with RDS for MySQL.

You can run multiple read replica create and delete actions at the same time that reference the same source DB instance. When you perform these actions, stay within the limit of 15 read replicas for each source instance.

A read replica of a MySQL DB instance can't use a lower DB engine version than its source DB instance.

Preparing MySQL DB instances that use MyISAM

If your MySQL DB instance uses a nontransactional engine such as MyISAM, you need to perform the following steps to successfully set up your read replica. These steps are required to make sure that the read replica has a consistent copy of your data. These steps are not required if all of your tables use a transactional engine such as InnoDB.

1. Stop all data manipulation language (DML) and data definition language (DDL) operations on non-transactional tables in the source DB instance and wait for them to complete. SELECT statements can continue running.
2. Flush and lock the tables in the source DB instance.
3. Create the read replica using one of the methods in the following sections.
4. Check the progress of the read replica creation using, for example, the `DescribeDBInstances` API operation. Once the read replica is available, unlock the tables of the source DB instance and resume normal database operations.

Configuring replication filters with MySQL

You can use replication filters to specify which databases and tables are replicated with a read replica. Replication filters can include databases and tables in replication or exclude them from replication.

The following are some use cases for replication filters:

- To reduce the size of a read replica. With replication filtering, you can exclude the databases and tables that aren't needed on the read replica.
- To exclude databases and tables from read replicas for security reasons.
- To replicate different databases and tables for specific use cases at different read replicas. For example, you might use specific read replicas for analytics or sharding.
- For a DB instance that has read replicas in different AWS Regions, to replicate different databases or tables in different AWS Regions.

> **Note**
>
> You can also use replication filters to specify which databases and tables are replicated with a primary MySQL DB instance that is configured as a replica in an inbound replication topology. For more information about this configuration, see Configuring binary log file position replication with an external source instance.

Topics

- Setting replication filtering parameters for RDS for MySQL
- Replication filtering limitations for RDS for MySQL
- Replication filtering examples for RDS for MySQL
- Viewing the replication filters for a read replica

Setting replication filtering parameters for RDS for MySQL

To configure replication filters, set the following replication filtering parameters on the read replica:

- `replicate-do-db` – Replicate changes to the specified databases. When you set this parameter for a read replica, only the databases specified in the parameter are replicated.
- `replicate-ignore-db` – Don't replicate changes to the specified databases. When the `replicate-do-db` parameter is set for a read replica, this parameter isn't evaluated.
- `replicate-do-table` – Replicate changes to the specified tables. When you set this parameter for a read replica, only the tables specified in the parameter are replicated. Also, when the `replicate-do-db` or `replicate-ignore-db` parameter is set, make sure to include the database that includes the specified tables in replication with the read replica.

- `replicate-ignore-table` – Don't replicate changes to the specified tables. When the `replicate-do-table` parameter is set for a read replica, this parameter isn't evaluated.
- `replicate-wild-do-table` – Replicate tables based on the specified database and table name patterns. The `%` and _ wildcard characters are supported. When the `replicate-do-db` or `replicate-ignore-db` parameter is set, make sure to include the database that includes the specified tables in replication with the read replica.
- `replicate-wild-ignore-table` – Don't replicate tables based on the specified database and table name patterns. The `%` and _ wildcard characters are supported. When the `replicate-do-table` or `replicate-wild-do-table` parameter is set for a read replica, this parameter isn't evaluated.

The parameters are evaluated in the order that they are listed. For more information about how these parameters work, see the MySQL documentation:

- For general information, see Replica Server Options and Variables.
- For information about how database replication filtering parameters are evaluated, see Evaluation of Database-Level Replication and Binary Logging Options.
- For information about how table replication filtering parameters are evaluated, see Evaluation of Table-Level Replication Options.

By default, each of these parameters has an empty value. On each read replica, you can use these parameters to set, change, and delete replication filters. When you set one of these parameters, separate each filter from others with a comma.

You can use the `%` and _ wildcard characters in the `replicate-wild-do-table` and `replicate-wild-ignore-table` parameters. The `%` wildcard matches any number of characters, and the _ wildcard matches only one character.

The binary logging format of the source DB instance is important for replication because it determines the record of data changes. The setting of the `binlog_format` parameter determines whether the replication is row-based or statement-based. For more information, see Configuring RDS for MySQL binary logging for Single-AZ databases.

> **Note**
>
> All data definition language (DDL) statements are replicated as statements, regardless of the `binlog_format` setting on the source DB instance.

Replication filtering limitations for RDS for MySQL

The following limitations apply to replication filtering for RDS for MySQL:

- Each replication filtering parameter has a 2,000-character limit.
- Commas aren't supported in replication filters for parameter values. In a list of parameters, commas can only be used as value separators. For example, `ParameterValue='`a,b`'` isn't supported, but `ParameterValue='a,b'` is.
- The MySQL `--binlog-do-db` and `--binlog-ignore-db` options for binary log filtering aren't supported.
- Replication filtering doesn't support XA transactions.

  For more information, see Restrictions on XA Transactions in the MySQL documentation.

Replication filtering examples for RDS for MySQL

To configure replication filtering for a read replica, modify the replication filtering parameters in the parameter group associated with the read replica.

> **Note**
>
> You can't modify a default parameter group. If the read replica is using a default parameter group, create a new parameter group and associate it with the read replica. For more information on DB parameter groups, see Parameter groups for Amazon RDS.

You can set parameters in a parameter group using the AWS Management Console, AWS CLI, or RDS API. For information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS. When you set parameters in a parameter group, all of the DB instances associated with the parameter group use the parameter settings. If you set the replication filtering parameters in a parameter group, make sure that the parameter group is associated only with read replicas. Leave the replication filtering parameters empty for source DB instances.

The following examples set the parameters using the AWS CLI. These examples set `ApplyMethod` to `immediate` so that the parameter changes occur immediately after the CLI command completes. If you want a pending change to be applied after the read replica is rebooted, set `ApplyMethod` to `pending-reboot`.

The following examples set replication filters:

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters
- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example Including databases in replication

The following example includes the `mydb1` and `mydb2` databases in replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example Including tables in replication

The following example includes the `table1` and `table2` tables in database `mydb1` in replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example Including tables in replication using wildcard characters

The following example includes tables with names that begin with `order` and `return` in database `mydb` in replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example Excluding databases from replication

The following example excludes the `mydb5` and `mydb6` databases from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

Example Excluding tables from replication

The following example excludes tables `table1` in database `mydb5` and `table2` in database `mydb6` from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example Excluding tables from replication using wildcard characters

The following example excludes tables with names that begin with `order` and `return` in database `mydb7` from replication.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Viewing the replication filters for a read replica

You can view the replication filters for a read replica in the following ways:

- Check the settings of the replication filtering parameters in the parameter group associated with the read replica.

  For instructions, see Viewing parameter values for a DB parameter group in Amazon RDS.
- In a MySQL client, connect to the read replica and run the `SHOW REPLICA STATUS` statement.

  In the output, the following fields show the replication filters for the read replica:

  - `Replicate_Do_DB`
  - `Replicate_Ignore_DB`
  - `Replicate_Do_Table`
  - `Replicate_Ignore_Table`
  - `Replicate_Wild_Do_Table`
  - `Replicate_Wild_Ignore_Table`

  For more information about these fields, see Checking Replication Status in the MySQL documentation.

Configuring delayed replication with MySQL

You can use delayed replication as a strategy for disaster recovery. With delayed replication, you specify the minimum amount of time, in seconds, to delay replication from the source to the

read replica. In the event of a disaster, such as a table deleted unintentionally, you complete the following steps to recover from the disaster quickly:

- Stop replication to the read replica before the change that caused the disaster is sent to it.

  Use the mysql.rds_stop_replication stored procedure to stop replication.
- Start replication and specify that replication stops automatically at a log file location.

  You specify a location just before the disaster using the mysql.rds_start_replication_until stored procedure.
- Promote the read replica to be the new source DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

Note

- On RDS for MySQL 8.4, delayed replication is supported for MySQL 8.4.3 and higher. On RDS for MySQL 8.0, delayed replication is supported for MySQL 8.0.28 and higher. On RDS for MySQL 5.7, delayed replication is supported for MySQL 5.7.44 and higher.
- Use stored procedures to configure delayed replication. You can't configure delayed replication with the AWS Management Console, the AWS CLI, or the Amazon RDS API.
- You can use replication based on global transaction identifiers (GTIDs) in a delayed replication configuration for the following versions:
  - RDS for MySQL version 5.7.44 and higher 5.7 versions
  - RDS for MySQL version 8.0.28 and higher 8.0 versions
  - RDS for MySQL version 8.4.3 and higher 8.4 versions

  If you use GTID-based replication, use the mysql.rds_start_replication_until_gtid stored procedure instead of the mysql.rds_start_replication_until stored procedure. For more information about GTID-based replication, see Using GTID-based replication.

Topics

- Configuring delayed replication during read replica creation
- Modifying delayed replication for an existing read replica
- Setting a location to stop replication to a read replica

- Promoting a read replica

Configuring delayed replication during read replica creation

To configure delayed replication for any future read replica created from a DB instance, run the mysql.rds_set_configuration stored procedure with the `target delay` parameter.

To configure delayed replication during read replica creation

1. Using a MySQL client, connect to the MySQL DB instance to be the source for read replicas as the master user.
2. Run the mysql.rds_set_configuration stored procedure with the `target delay` parameter.

   For example, run the following stored procedure to specify that replication is delayed by at least one hour (3,600 seconds) for any read replica created from the current DB instance.

   ```
   call mysql.rds_set_configuration('target delay', 3600);
   ```

   > **Note**
   >
   > After running this stored procedure, any read replica you create using the AWS CLI or Amazon RDS API is configured with replication delayed by the specified number of seconds.

Modifying delayed replication for an existing read replica

To modify delayed replication for an existing read replica, run the mysql.rds_set_source_delay stored procedure.

To modify delayed replication for an existing read replica

1. Using a MySQL client, connect to the read replica as the master user.
2. Use the mysql.rds_stop_replication stored procedure to stop replication.
3. Run the mysql.rds_set_source_delay stored procedure.

   For example, run the following stored procedure to specify that replication to the read replica is delayed by at least one hour (3600 seconds).

```
call mysql.rds_set_source_delay(3600);
```

4. Use the mysql.rds_start_replication stored procedure to start replication.

Setting a location to stop replication to a read replica

After stopping replication to the read replica, you can start replication and then stop it at a specified binary log file location using the mysql.rds_start_replication_until stored procedure.

To start replication to a read replica and stop replication at a specific location

1. Using a MySQL client, connect to the source MySQL DB instance as the master user.
2. Run the mysql.rds_start_replication_until stored procedure.

   The following example initiates replication and replicates changes until it reaches location `120` in the `mysql-bin-changelog.000777` binary log file. In a disaster recovery scenario, assume that location `120` is just before the disaster.

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

Replication stops automatically when the stop point is reached. The following RDS event is generated: `Replication has been stopped since the replica reached the stop point specified by the rds_start_replication_until stored procedure.`

Promoting a read replica

After replication is stopped, in a disaster recovery scenario, you can promote a read replica to be the new source DB instance. For information about promoting a read replica, see Promoting a read replica to be a standalone DB instance.

Updating read replicas with MySQL

Read replicas are designed to support read queries, but you might need occasional updates. For example, you might need to add an index to optimize the specific types of queries accessing the replica.

Although you can enable updates by setting the `read_only` parameter to `0` in the DB parameter group for the read replica, we recommend that you don't do so because it can cause problems if the read replica becomes incompatible with the source DB instance. For maintenance operations, we recommend that you use blue/green deployments. For more information, see Using Blue/Green Deployments for database updates.

If you disable read-only on a read replica, change the value of the `read_only` parameter back to `1` as soon as possible.

Working with Multi-AZ read replica deployments with MySQL

You can create a read replica from either single-AZ or Multi-AZ DB instance deployments. You use Multi-AZ deployments to improve the durability and availability of critical data, but you can't use the Multi-AZ secondary to serve read-only queries. Instead, you can create read replicas from high-traffic Multi-AZ DB instances to offload read-only queries. If the source instance of a Multi-AZ deployment fails over to the secondary, any associated read replicas automatically switch to use the secondary (now primary) as their replication source. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

You can create a read replica as a Multi-AZ DB instance. Amazon RDS creates a standby of your replica in another Availability Zone for failover support for the replica. Creating your read replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ DB instance.

Using cascading read replicas with RDS for MySQL

RDS for MySQL supports cascading read replicas. With *cascading read replicas*, you can scale reads without adding overhead to your source RDS for MySQL DB instance.

With cascading read replicas, your RDS for MySQL DB instance sends data to the first read replica in the chain. That read replica then sends data to the second replica in the chain, and so on. The end result is that all read replicas in the chain have the changes from the RDS for MySQL DB instance, but without the overhead solely on the source DB instance.

You can create a series of up to three read replicas in a chain from a source RDS for MySQL DB instance. For example, suppose that you have an RDS for MySQL DB instance, `mysql-main`. You can do the following:

- Starting with `mysql-main`, create the first read replica in the chain, `read-replica-1`.
- Next, from `read-replica-1`, create the next read replica in the chain, `read-replica-2`.
- Finally, from `read-replica-2`, create the third read replica in the chain, `read-replica-3`.

You can't create another read replica beyond this third cascading read replica in the series for `mysql-main`. A complete series of instances from an RDS for MySQL source DB instance through to the end of a series of cascading read replicas can consist of at most four DB instances.

For cascading read replicas to work, each source RDS for MySQL DB instance must have automated backups turned on. To turn on automatic backups on a read replica, first create the read replica, and then modify the read replica to turn on automatic backups. For more information, see Creating a read replica.

As with any read replica, you can promote a read replica that's part of a cascade. Promoting a read replica from within a chain of read replicas removes that replica from the chain. For example, suppose that you want to move some of the workload from your `mysql-main` DB instance to a new instance for use by the accounting department only. Assuming the chain of three read replicas from the example, you decide to promote `read-replica-2`. The chain is affected as follows:

- Promoting `read-replica-2` removes it from the replication chain.
  - It is now a full read/write DB instance.
  - It continues replicating to `read-replica-3`, just as it was doing before promotion.
- Your `mysql-main` continues replicating to `read-replica-1`.

For more information about promoting read replicas, see Promoting a read replica to be a standalone DB instance.

Monitoring replication lag for MySQL read replicas

For MySQL read replicas, you can monitor replication lag in Amazon CloudWatch by viewing the Amazon RDS `ReplicaLag` metric. The `ReplicaLag` metric reports the value of the `Seconds_Behind_Master` field of the `SHOW REPLICA STATUS` command.

Common causes for replication lag for MySQL are the following:

- A network outage.
- Writing to tables that have different indexes on a read replica. If the `read_only` parameter is set to `0` on the read replica, replication can break if the read replica becomes incompatible with the source DB instance. After you've performed maintenance tasks on the read replica, we recommend that you set the `read_only` parameter back to 1.
- Using a nontransactional storage engine such as MyISAM. Replication is only supported for the InnoDB storage engine on MySQL.

When the `ReplicaLag` metric reaches 0, the replica has caught up to the source DB instance. If the `ReplicaLag` metric returns -1, then replication is currently not active. `ReplicaLag` = -1 is equivalent to `Seconds_Behind_Master` = `NULL`.

Starting and stopping replication with MySQL read replicas

You can stop and restart the replication process on an Amazon RDS DB instance by calling the system stored procedures mysql.rds_stop_replication and mysql.rds_start_replication. You can do this when replicating between two Amazon RDS instances for long-running operations such as creating large indexes. You also need to stop and start replication when importing or exporting databases. For more information, see Importing data to an Amazon RDS for MySQL database with reduced downtime and Exporting data from a MySQL DB instance by using replication.

If replication is stopped for more than 30 consecutive days, either manually or due to a replication error, Amazon RDS terminates replication between the source DB instance and all read replicas. It does so to prevent increased storage requirements on the source DB instance and long failover times. The read replica DB instance is still available. However, replication can't be resumed because the binary logs required by the read replica are deleted from the source DB instance after replication is terminated. You can create a new read replica for the source DB instance to reestablish replication.

Troubleshooting a MySQL read replica problem

For MySQL DB instances, in some cases read replicas present replication errors or data inconsistencies (or both) between the read replica and its source DB instance. This problem occurs when some binary log (binlog) events or InnoDB redo logs aren't flushed during a failure of the read replica or the source DB instance. In these cases, manually delete and recreate the read replicas. You can reduce the chance of this happening by setting the following parameter values: `sync_binlog=1` and `innodb_flush_log_at_trx_commit=1`. These settings might reduce performance, so test their impact before implementing the changes in a production environment.

Warning

In the parameter group associated with the source DB instance, we recommend keeping these parameter values: `sync_binlog=1` and `innodb_flush_log_at_trx_commit=1`. These parameters are dynamic. If you don't want to use these settings, we recommend temporarily setting those values before executing any operation on the source DB instance that might cause it to restart. These operations include, but are not limited to, rebooting, rebooting with failover, upgrading the database version, and changing the DB instance

class or its storage. The same recommendation applies to creating new read replicas for the source DB instance.
Failure to follow this guidance increases the risk of read replicas presenting replication errors or data inconsistencies (or both) between the read replica and its source DB instance.

The replication technologies for MySQL are asynchronous. Because they are asynchronous, occasional `BinLogDiskUsage` increases on the source DB instance and `ReplicaLag` on the read replica are to be expected. For example, a high volume of write operations to the source DB instance can occur in parallel. In contrast, write operations to the read replica are serialized using a single I/O thread, which can lead to a lag between the source instance and read replica. For more information about read-only replicas in the MySQL documentation, see Replication implementation details.

You can do several things to reduce the lag between updates to a source DB instance and the subsequent updates to the read replica, such as the following:

- Sizing a read replica to have a storage size and DB instance class comparable to the source DB instance.
- Ensuring that parameter settings in the DB parameter groups used by the source DB instance and the read replica are compatible. For more information and an example, see the discussion of the `max_allowed_packet` parameter later in this section.

Amazon RDS monitors the replication status of your read replicas and updates the `Replication State` field of the read replica instance to `Error` if replication stops for any reason. An example might be if DML queries run on your read replica conflict with the updates made on the source DB instance.

You can review the details of the associated error thrown by the MySQL engine by viewing the `Replication Error` field. Events that indicate the status of the read replica are also generated, including RDS-EVENT-0045, RDS-EVENT-0046, and RDS-EVENT-0047. For more information about events and subscribing to events, see Working with Amazon RDS event notification. If a MySQL error message is returned, review the error number in the MySQL error message documentation.

One common issue that can cause replication errors is when the value for the `max_allowed_packet` parameter for a read replica is less than the `max_allowed_packet` parameter for the source DB instance. The `max_allowed_packet` parameter is a custom parameter that you can set in a DB parameter group. You use `max_allowed_packet` to

specify the maximum size of DML code that can be run on the database. In some cases, the `max_allowed_packet` value in the DB parameter group associated with a read replica is smaller than the `max_allowed_packet` value in the DB parameter group associated with the source DB instance. In these cases, the replication process can throw the error `Packet bigger than 'max_allowed_packet' bytes` and stop replication. To fix the error, have the source DB instance and read replica use DB parameter groups with the same `max_allowed_packet` parameter values.

Other common situations that can cause replication errors include the following:

- Writing to tables on a read replica. In some cases, you might create indexes on a read replica that are different from the indexes on the source DB instance. If you do, set the `read_only` parameter to `0` to create the indexes. If you write to tables on the read replica, it might break replication if the read replica becomes incompatible with the source DB instance. After you perform maintenance tasks on the read replica, we recommend that you set the `read_only` parameter back to 1.
- Using a non-transactional storage engine such as MyISAM. Read replicas require a transactional storage engine. Replication is only supported for the InnoDB storage engine on MySQL.
- Using unsafe nondeterministic queries such as `SYSDATE()`. For more information, see Determination of safe and unsafe statements in binary logging.

If you decide that you can safely skip an error, you can follow the steps described in the section Skipping the current replication error for RDS for MySQL. Otherwise, you can first delete the read replica. Then you create an instance using the same DB instance identifier so that the endpoint remains the same as that of your old read replica. If a replication error is fixed, the `Replication State` changes to *replicating*.

Using GTID-based replication

The following content explains how to use global transaction identifiers (GTIDs) with binary log (binlog) replication among Amazon RDS for MySQL DB instances.

If you use binlog replication and aren't familiar with GTID-based replication with MySQL, see Replication with global transaction identifiers in the MySQL documentation.

GTID-based replication is supported for the following versions:

- All RDS for MySQL 8.4 versions

- All RDS for MySQL 8.0 versions
- All RDS for MySQL 5.7 versions

All MySQL DB instances in a replication configuration must meet this version requirement.

Topics

- Overview of global transaction identifiers (GTIDs)
- Parameters for GTID-based replication
- Enabling GTID-based replication for new read replicas for RDS for MySQL
- Enabling GTID-based replication for existing read replicas for RDS for MySQL
- Disabling GTID-based replication for a MySQL DB instance with read replicas

Overview of global transaction identifiers (GTIDs)

Global transaction identifiers (GTIDs) are unique identifiers generated for committed MySQL transactions. You can use GTIDs to make binlog replication simpler and easier to troubleshoot.

MySQL uses two different types of transactions for binlog replication:

- *GTID transactions* – Transactions that are identified by a GTID.
- *Anonymous transactions* – Transactions that don't have a GTID assigned.

In a replication configuration, GTIDs are unique across all DB instances. GTIDs simplify replication configuration because when you use them, you don't have to refer to log file positions. GTIDs also make it easier to track replicated transactions and determine whether the source instance and replicas are consistent.

You can use GTID-based replication to replicate data with RDS for MySQL read replicas. You can configure GTID-based replication when you are creating new read replicas, or you can convert existing read replicas to use GTID-based replication.

You can also use GTID-based replication in a delayed replication configuration with RDS for MySQL. For more information, see Configuring delayed replication with MySQL.

Parameters for GTID-based replication

Use the following parameters to configure GTID-based replication.

Parameter	Valid values	Description
gtid_mode	OFF, OFF_PERMISSIVE , ON_PERMISSIVE , ON	OFF specifies that new transactions are anonymous transactions (that is, don't have GTIDs), and a transaction must be anonymous to be replicated. OFF_PERMISSIVE specifies that new transactions are anonymous transactions, but all transactions can be replicated. ON_PERMISSIVE specifies that new transactions are GTID transactions, but all transactions can be replicated. ON specifies that new transactions are GTID transactions, and a transaction must be a GTID transaction to be replicated.
enforce_gtid_consistency	OFF, ON, WARN	OFF allows transactions to violate GTID consistency. ON prevents transactions from violating GTID consistency. WARN allows transactions to violate GTID consistency but generates a warning when a violation occurs.

Note

In the AWS Management Console, the gtid_mode parameter appears as gtid-mode.

For GTID-based replication, use these settings for the parameter group for your DB instance or read replica:

- ON and ON_PERMISSIVE apply only to outgoing replication from an RDS DB instance. Both of these values cause your RDS DB instance to use GTIDs for transactions that are replicated. ON requires that the target database also use GTID-based replication. ON_PERMISSIVE makes GTID-based replication optional on the target database.
- OFF_PERMISSIVE, if set, means that your RDS DB instances can accept incoming replication from a source database. They can do this regardless of whether the source database uses GTID-based replication.
- OFF, if set, means that your RDS DB instance only accepts incoming replication from source databases that don't use GTID-based replication.

For more information about parameter groups, see Parameter groups for Amazon RDS.

Enabling GTID-based replication for new read replicas for RDS for MySQL

When GTID-based replication is enabled for an RDS for MySQL DB instance, GTID-based replication is configured automatically for read replicas of the DB instance.

To enable GTID-based replication for new read replicas

1. Make sure that the parameter group associated with the DB instance has the following parameter settings:

   - gtid_mode – ON or ON_PERMISSIVE
   - enforce_gtid_consistency – ON

   For more information about setting configuration parameters using parameter groups, see Parameter groups for Amazon RDS.
2. If you changed the parameter group of the DB instance, reboot the DB instance. For more information on how to do so, see Rebooting a DB instance.
3. Create one or more read replicas of the DB instance. For more information on how to do so, see Creating a read replica.

Amazon RDS attempts to establish GTID-based replication between the MySQL DB instance and the read replicas using the MASTER_AUTO_POSITION. If the attempt fails, Amazon RDS uses log file positions for replication with the read replicas. For more information about the MASTER_AUTO_POSITION, see GTID auto-positioning in the MySQL documentation.

Enabling GTID-based replication for existing read replicas for RDS for MySQL

For an existing MySQL DB instance with read replicas that doesn't use GTID-based replication, you can configure GTID-based replication between the DB instance and the read replicas.

To enable GTID-based replication for existing read replicas

1. If the DB instance or any read replica is using an 8.0 version of RDS for MySQL version lower than 8.0.26, upgrade the DB instance or read replica to 8.0.26 or a higher MySQL 8.0 version. All RDS for MySQL 8.4 versions and 5.7 versions support GTID-based replication.

   For more information, see Upgrades of the RDS for MySQL DB engine.
2. (Optional) Reset the GTID parameters and test the behavior of the DB instance and read replicas:

   a. Make sure that the parameter group associated with the DB instance and each read replica has the `enforce_gtid_consistency` parameter set to `WARN`.

      For more information about setting configuration parameters using parameter groups, see Parameter groups for Amazon RDS.

   b. If you changed the parameter group of the DB instance, reboot the DB instance. If you changed the parameter group for a read replica, reboot the read replica.

      For more information, see Rebooting a DB instance.

   c. Run your DB instance and read replicas with your normal workload and monitor the log files.

      If you see warnings about GTID-incompatible transactions, adjust your application so that it only uses GTID-compatible features. Make sure that the DB instance is not generating any warnings about GTID-incompatible transactions before proceeding to the next step.
3. Reset the GTID parameters for GTID-based replication that allows anonymous transactions until the read replicas have processed all of them.

   a. Make sure that the parameter group associated with the DB instance and each read replica has the following parameter settings:

      - `gtid_mode` – `ON_PERMISSIVE`
      - `enforce_gtid_consistency` – `ON`

b. If you changed the parameter group of the DB instance, reboot the DB instance. If you changed the parameter group for a read replica, reboot the read replica.

4. Wait for all of your anonymous transactions to be replicated. To check that these are replicated, do the following:

   a. Run the following statement on your source DB instance.

      MySQL 8.4

      ```
      SHOW BINARY LOG STATUS;
      ```

      MySQL 5.7 and 8.0

      ```
      SHOW MASTER STATUS;
      ```

      Note the values in the `File` and `Position` columns.

   b. On each read replica, use the file and position information from its source instance in the previous step to run the following query.

      ```
      SELECT MASTER_POS_WAIT('file', position);
      ```

      For example, if the file name is `mysql-bin-changelog.000031` and the position is `107`, run the following statement.

      ```
      SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
      ```

      If the read replica is past the specified position, the query returns immediately. Otherwise, the function waits. Proceed to the next step when the query returns for all read replicas.

5. Reset the GTID parameters for GTID-based replication only.

   a. Make sure that the parameter group associated with the DB instance and each read replica has the following parameter settings:

      - `gtid_mode` – `ON`
      - `enforce_gtid_consistency` – `ON`

   b. Reboot the DB instance and each read replica.

6. On each read replica, run the following procedure.

   MySQL 8.4 and higher major versions

   ```
   CALL mysql.rds_set_source_auto_position(1);
   ```

   MySQL 8.0 and lower major versions

   ```
   CALL mysql.rds_set_master_auto_position(1);
   ```

Disabling GTID-based replication for a MySQL DB instance with read replicas

You can disable GTID-based replication for a MySQL DB instance with read replicas.

To disable GTID-based replication for a MySQL DB instance with read replicas

1. On each read replica, run the following procedure:

   MySQL 8.4 and higher major versions

   ```
   CALL mysql.rds_set_source_auto_position(0);
   ```

   MySQL 8.0 and lower major versions

   ```
   CALL mysql.rds_set_master_auto_position(0);
   ```

2. Reset the `gtid_mode` to `ON_PERMISSIVE`.

   a. Make sure that the parameter group associated with the MySQL DB instance and each read replica has `gtid_mode` set to `ON_PERMISSIVE`.

      For more information about setting configuration parameters using parameter groups, see Parameter groups for Amazon RDS.

   b. Reboot the MySQL DB instance and each read replica. For more information about rebooting, see Rebooting a DB instance.

3. Reset the `gtid_mode` to `OFF_PERMISSIVE`.

   a. Make sure that the parameter group associated with the MySQL DB instance and each read replica has `gtid_mode` set to `OFF_PERMISSIVE`.

b. Reboot the MySQL DB instance and each read replica.

4. Wait for all of the GTID transactions to be applied on all of the read replicas. To check that these are applied, do the following steps:

   a. On the MySQL DB instance, run the following command:

      MySQL 8.4

      ```
      SHOW BINARY LOG STATUS
      ```

      MySQL 5.7 and 8.0

      ```
      SHOW MASTER STATUS
      ```

      Your output should be similar to the following output.

      ```
      File                        Position
      ------------------------------------
      mysql-bin-changelog.000031      107
      ------------------------------------
      ```

      Note the file and position in your output.

   b. On each read replica, use the file and position information from its source instance in the previous step to run the following query:

      MySQL 8.4 and MySQL 8.0.26 and higher MySQL 8.0 versions

      ```
      SELECT SOURCE_POS_WAIT('file', position);
      ```

      MySQL 5.7

      ```
      SELECT MASTER_POS_WAIT('file', position);
      ```

      For example, if the file name is `mysql-bin-changelog.000031` and the position is `107`, run the following statement:

      MySQL 8.4 and MySQL 8.0.26 and higher MySQL 8.0 versions

```
SELECT SOURCE_POS_WAIT('mysql-bin-changelog.000031', 107);
```

MySQL 5.7

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

5. Reset the GTID parameters to disable GTID-based replication.

   a. Make sure that the parameter group associated with the MySQL DB instance and each read replica has the following parameter settings:

      - `gtid_mode` – `OFF`
      - `enforce_gtid_consistency` – `OFF`

   b. Reboot the MySQL DB instance and each read replica.

Configuring binary log file position replication with an external source instance

You can set up replication between an RDS for MySQL or MariaDB DB instance and a MySQL or MariaDB instance that is external to Amazon RDS using binary log file replication.

Topics

- Before you begin
- Configuring binary log file position replication with an external source instance

Before you begin

You can configure replication using the binary log file position of replicated transactions.

The permissions required to start replication on an Amazon RDS DB instance are restricted and not available to your Amazon RDS master user. Because of this, make sure that you use the Amazon RDS mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) or mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher), and mysql.rds_start_replication commands to set up replication between your live database and your Amazon RDS database.

To set the binary logging format for a MySQL or MariaDB database, update the `binlog_format` parameter. If your DB instance uses the default DB instance parameter group, create a new DB parameter group to modify the `binlog_format` parameter. In MariaDB and MySQL 8.0 and lower versions, `binlog_format` defaults to `MIXED`. However, you can also set `binlog_format` to `ROW` or `STATEMENT` if you need a specific binary log (binlog) format. Reboot your DB instance for the change to take effect. In MySQL 8.4 and higher versions, `binlog_format` defaults to `ROW`.

For information about setting the `binlog_format` parameter, see Configuring RDS for MySQL binary logging for Single-AZ databases. For information about the implications of different MySQL replication types, see Advantages and disadvantages of statement-based and row-based replication in the MySQL documentation.

Configuring binary log file position replication with an external source instance

Follow these guidelines when you set up an external source instance and a replica on Amazon RDS:

- Monitor failover events for the Amazon RDS DB instance that is your replica. If a failover occurs, then the DB instance that is your replica might be recreated on a new host with a different network address. For information on how to monitor failover events, see Working with Amazon RDS event notification.
- Maintain the binlogs on your source instance until you have verified that they have been applied to the replica. This maintenance makes sure that you can restore your source instance in the event of a failure.
- Turn on automated backups on your Amazon RDS DB instance. Turning on automated backups makes sure that you can restore your replica to a particular point in time if you need to re-synchronize your source instance and replica. For information on backups and point-in-time restore, see Backing up, restoring, and exporting data.

To configure binary log file replication with an external source instance

1. Make the source MySQL or MariaDB instance read-only.

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   mysql> SET GLOBAL read_only = ON;
   ```

2. Run the `SHOW MASTER STATUS` command on the source MySQL or MariaDB instance to determine the binlog location.

   You receive output similar to the following example.

```
File                        Position
-------------------------------------
 mysql-bin-changelog.000031      107
-------------------------------------
```

3. Copy the database from the external instance to the Amazon RDS DB instance using `mysqldump`. For very large databases, you might want to use the procedure in [Importing data to an Amazon RDS for MySQL database with reduced downtime].

   For Linux, macOS, or Unix:

```
mysqldump --databases database_name \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -plocal_password | mysql \
        --host=hostname \
        --port=3306 \
        -u RDS_user_name \
        -pRDS_password
```

   For Windows:

```
mysqldump --databases database_name ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -plocal_password | mysql ^
        --host=hostname ^
        --port=3306 ^
        -u RDS_user_name ^
        -pRDS_password
```

   Note

   Make sure that there isn't a space between the `-p` option and the entered password.

To specify the host name, user name, port, and password to connect to your Amazon RDS DB instance, use the `--host`, `--user (-u)`, `--port` and `-p` options in the `mysql` command. The host name is the Domain Name Service (DNS) name from the Amazon RDS DB instance endpoint, for example `myinstance.123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the instance details in the AWS Management Console.

4. Make the source MySQL or MariaDB instance writeable again.

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

For more information on making backups for use with replication, see the MySQL documentation.

5. In the AWS Management Console, add the IP address of the server that hosts the external database to the virtual private cloud (VPC) security group for the Amazon RDS DB instance. For more information on modifying a VPC security group, see Security groups for your VPC in the *Amazon Virtual Private Cloud User Guide*.

The IP address can change when the following conditions are met:

- You are using a public IP address for communication between the external source instance and the DB instance.
- The external source instance was stopped and restarted.

If these conditions are met, verify the IP address before adding it.

You might also need to configure your local network to permit connections from the IP address of your Amazon RDS DB instance. You do this so that your local network can communicate with your external MySQL or MariaDB instance. To find the IP address of the Amazon RDS DB instance, use the `host` command.

```
host db_instance_endpoint
```

The host name is the DNS name from the Amazon RDS DB instance endpoint.

6. Using the client of your choice, connect to the external instance and create a user to use for replication. Use this account solely for replication and restrict it to your domain to improve security. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

7. For the external instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. For example, to grant the `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the '`repl_user`' user for your domain, issue the following command.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

8. Make the Amazon RDS DB instance the replica. To do so, first connect to the Amazon RDS DB instance as the master user. Then identify the external MySQL or MariaDB database as the source instance by using the mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher) or mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) command. Use the master log file name and master log position that you determined in step 2. The following commands are examples.

MySQL 8.4

```
CALL mysql.rds_set_external_source ('mysourceserver.mydomain.com', 3306,
 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

MariaDB and MySQL 8.0 and 5.7

```
CALL mysql.rds_set_external_master ('mymasterserver.mydomain.com', 3306,
 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1);
```

> **Note**
>
> On RDS for MySQL, you can choose to use delayed replication by running the mysql.rds_set_external_source_with_delay (RDS for MySQL major versions 8.4

and higher) or mysql.rds_set_external_master_with_delay (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) stored procedure instead. On RDS for MySQL, one reason to use delayed replication is to turn on disaster recovery with the mysql.rds_start_replication_until stored procedure. Currently, RDS for MariaDB supports delayed replication but doesn't support the `mysql.rds_start_replication_until` procedure.

9. On the Amazon RDS DB instance, issue the mysql.rds_start_replication command to start replication.

```
CALL mysql.rds_start_replication;
```

Configuring multi-source-replication for Amazon RDS for MySQL

With multi-source replication, you can set up an Amazon RDS for MySQL DB instance as a replica that receives binary log events from more than one RDS for MySQL source DB instance. Multi-source replication is supported for RDS for MySQL DB instances running the following engine versions:

- All MySQL 8.4 versions
- 8.0.35 and higher minor versions
- 5.7.44 and higher minor versions

For information about MySQL multi-source replication, see MySQL Multi-Source Replication in the MySQL documentation. The MySQL documentation contains detailed information about this feature, while this topic describes how to configure and manage the multi-source replication channels on your RDS for MySQL DB instances.

Use cases for multi-source replication

The following cases are good candidates for using multi-source replication on RDS for MySQL:

- Applications that need to merge or combine multiple shards on separate DB instances into a single shard.
- Applications that need to generate reports from data consolidated from multiple sources.

- Requirements to create consolidated long-term backups of data that's distributed among multiple RDS for MySQL DB instances.

Prerequisites for multi-source replication

Before you configure multi-source replication, complete the following prerequisites.

- Make sure that each source RDS for MySQL DB instance has automatic backups enabled. Enabling automatic backups enables binary logging. To learn how to enable automatic backups, see the section called “Enabling automated backups”.
- To avoid replication errors, we recommended that you block write operations to the source DB instances. You can do so by setting the `read-only` parameter to `ON` in a custom parameter group attached to the RDS for MySQL source DB instance. You can use the AWS Management Console or the AWS CLI to create a new custom parameter group or to modify an existing one. For more information, see the section called “Creating a DB parameter group” and the section called “Modifying parameters in a DB parameter group”.
- For each source DB instance, add the IP address of the instance to the Amazon virtual private cloud (VPC) security group for the multi-source DB instance. To identify the IP address of a source DB instance, you can run the command `dig` *`RDS Endpoint`*. Run the command from an Amazon EC2 instance in the same VPC as the destination multi-source DB instance.
- For each source DB instance, use a client to connect to the DB instance and create a database user with the required privileges for replication, as in the following example.

```
CREATE USER 'repl_user' IDENTIFIED BY 'password';
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user';
```

> **Note**
>
> Starting with MySQL 8.4, the `REPLICATION SLAVE` privilege has been deprecated and replaced with `REPLICATION REPLICA`. For MySQL 8.4 and later versions, use the following syntax instead:
>
> ```
> CREATE USER 'repl_user' IDENTIFIED BY 'password';
> GRANT REPLICATION CLIENT, REPLICATION REPLICA ON *.* TO 'repl_user';
> ```

Configuring multi-source replication channels on RDS for MySQL DB instances

Configuring multi-source replication channels is similar to configuring single source replication. For multi-source replication, you first turn on binary logging on the source instance. Then, you import data from the sources to the multi-source replica. Then, you start replication from each source by using the binary log coordinates or by using GTID auto-positioning.

To configure an RDS for MySQL DB instance as a multi-source replica of two or more RDS for MySQL DB instances, perform the following steps.

Topics

- Step 1: Import data from the source DB instances to the multi-source replica
- Step 2: Start replication from the source DB instances to the multi-source replica

Step 1: Import data from the source DB instances to the multi-source replica

Perform the following steps on each source DB instance.

Before you import the data from a source to the multi-source replica, determine the current binary log file and position by running the `SHOW MASTER STATUS` command. Take note of these details for use in the next step. In this example output, the file is `mysql-bin-changelog.000031` and the position is `107`.

> **Note**
>
> Starting with MySQL 8.4, the `SHOW MASTER STATUS` command has been deprecated and replaced with `SHOW BINARY LOG STATUS`. For MySQL 8.4 and later versions, use `SHOW BINARY LOG STATUS` instead.

```
File                        Position
-----------------------------------
mysql-bin-changelog.000031      107
-----------------------------------
```

Now copy the database from the source DB instance to the multi-source replica by using `mysqldump`, as in the following example.

```
mysqldump --databases database_name \
```

```
 --single-transaction \
 --compress \
 --order-by-primary \
 -u RDS_user_name \
 -p RDS_password \
 --host=RDS Endpoint | mysql \
 --host=RDS Endpoint \
 --port=3306 \
 -u RDS_user_name \
-p RDS_password
```

After copying the database, you can set the read-only parameter to OFF on the source DB instance.

Step 2: Start replication from the source DB instances to the multi-source replica

For each source DB instance, use the administrative user credentials to connect to the instance, and run the following two stored procedures. These stored procedures configure replication on a channel and start replication. This example uses the binlog file name and position from the example output in the previous step.

```
CALL mysql.rds_set_external_source_for_channel('mysourcehost.example.com', 3306,
 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 1, 'channel_1');
CALL mysql.rds_start_replication_for_channel('channel_1');
```

For more information about using these stored procedures and others to set up and manage your replication channels, see the section called “Managing multi-source replication”.

Using filters with multi-source replication

You can use replication filters to specify which databases and tables are replicated with in multi-source replica. Replication filters can include databases and tables in replication or exclude them from replication. For more information on replication filters, see the section called “Configuring replication filters”.

With multi-source replication, you can configure replication filters globally or at the channel level. Channel-level filtering is available only with supported DB instances running version 8.0 or version 8.4. The following examples show how to configure filters globally or at the channel level.

Note the following requirements and behavior with filtering in multi-source replication:

- Back quotes (` `) around the channel names are required.

- If you change replication filters in the parameter group, the multi-source replica's `sql_thread` for all channels with updates are restarted to apply the changes dynamically. If an update involves a global filter, then all replication channels in the running state are restarted.
- All global filters are applied before any channel-specific filters.
- If a filter is applied globally and at the channel level, then only the channel-level filter is applied. For example, if the filters are `replicate_ignore_db="db1,`channel_22`:db2"`, then `replicate_ignore_db` set to `db1` is applied to all channels except for `channel_22`, and only `channel_22` ignores changes from `db2`.

Example 1: Setting a global filter

In the following example, the `temp_data` database is excluded from replication in every channel.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-ignore-
db,ParameterValue='temp_data',ApplyMethod=immediate"
```

Example 2: Setting a channel-level filter

In the following example, changes from the `sample22` database are only included in channel `channel_22`. Similarly, changes from the `sample99` database are only included in channel `channel_99`.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
--db-parameter-group-name myparametergroup \
--parameters "ParameterName=replicate-do-db,ParameterValue='\`channel_22\`:sample22,
\`channel_99\`:sample99',ApplyMethod=immediate"
```

Monitoring multi-source replication channels

You can monitor individual channels in a multi-source replica by using the following methods:

- To monitor the status of all channels or a specific channel, connect to the multi-source replica and run the `SHOW REPLICA STATUS` or `SHOW REPLICA STATUS FOR CHANNEL`

'*channel_name*' command. For more information, see Checking Replication Status in the MySQL documentation.

- To receive notification when a replication channel is started, stopped, or removed, use RDS event notification. For more information, see the section called "Working with Amazon RDS event notification".
- To monitor the lag for a specific channel, check the `ReplicationChannelLag` metric for it. Data points for this metric have a period of 60 seconds (1 minute) are available for 15 days. To locate the replication channel lag for a channel, use the instance identifier and the replication channel name. To receive notification when this lag exceeds a particular threshold, you can set up a CloudWatch alarm. For more information, see the section called "Monitoring RDS with CloudWatch".

Considerations and best practices for multi-source replication

Before you use multi-source replication on RDS for MySQL, review the following considerations and best practices:

- Make sure that a DB instance configured as a multi-source replica has sufficient resources such as throughput, memory, CPU, and IOPS to handle the workload from multiple source instances.
- Regularly monitor resource utilization on your multi-source replica and adjust the storage or instance configuration to handle the workload without straining resources.
- You can configure multi-threaded replication on a multi-source replica by setting the system variable `replica_parallel_workers` to a value greater than `0`. In this case, the number of threads allocated to each channel is the value of this variable, plus one coordinator thread to manage the applier threads.
- Configure replication filters appropriately to avoid conflicts. To replicate an entire database to another database on a replica, you can use the `--replicate-rewrite-db` option. For example, you can replicate all tables in database A to database B on a replica instance. This approach can be helpful when all source instances are using the same schema naming convention. For information about the `--replicate-rewrite-db` option, see Replica Server Options and Variables in the MySQL documentation.
- To avoid replication errors, avoid writing to the replica. We recommended that you enable the `read_only` parameter on multi-source replicas to block write operations. Doing so helps to eliminate replication issues caused by conflicting write operations.

- To increase the performance of read operations such as sorts and high-load joins that are executed on the multi-source replica, consider using RDS Optimized Reads. This feature can help with queries that depend on large temporary tables or sort files. For more information, see the section called “Improving query performance with RDS Optimized Reads”.
- To minimize replication lag and improve the performance of a multi-source replica, consider enabling optimized writes. For more information, see the section called “Improving write performance with RDS Optimized Writes for MySQL”.
- Perform management operations (such as changing configuration) on one channel at a time, and avoid performing changes to multiple channels from multiple connections. These practices can lead to conflicts in replication operations. For example, simultaneously executing `rds_skip_repl_error_for_channel` and `rds_start_replication_for_channel` procedures from multiple connections can cause skipping of events on a different channel than intended.
- You can enable backups on a multi-source replication instance and export data from that instance to an Amazon S3 bucket to store it for long-term purposes. However, it's important to also configure backups with appropriate retention on the individual source instances. For information about exporting snapshot data to Amazon S3, see the section called “Exporting DB snapshot data to Amazon S3”.
- To distribute the read workload on a multi-source replica, you can create read replicas from a multi-source replica. You can locate these read replicas in different AWS Regions based on your application's requirements. For more information about read replicas, see the section called “MySQL read replicas”.

Limitations for multi-source replication on RDS for MySQL

The following limitations apply to multi-source replication on RDS for MySQL:

- Currently, RDS for MySQL supports configuring a maximum of 15 channels for a multi-source replica.
- A read replica instance can't be configured as a multi-source replica.
- To configure multi-source replication on RDS for MySQL running engine version 5.7, Performance Schema must be enabled on the replica instance. Enabling Performance Schema is optional on RDS for MySQL running engine version 8.0 or 8.4.

- For RDS for MySQL running engine version 5.7, replication filters apply to all replication channels. For RDS for MySQL running engine version 8.0 or 8.4, you can configure filters that apply to all replication channels or to individual channels.
- Restoring an RDS snapshot or performing a Point-in-time-Restore (PITR) doesn't restore multi-source replica channel configurations.
- When you create a read replica of a multi-source replica, it only replicates data from the multi-source instance. It doesn't restore any channel configuration.
- MySQL doesn't support setting up a different number of parallel workers for each channel. Every channel gets the same number of parallel workers based on the `replica_parallel_workers` value.

The following additional limitations apply if your multi-source replication target is a Multi-AZ DB cluster:

- A channel must be configured for a source RDS for MySQL instance before any writes to that instance occur.
- Each source RDS for MySQL instance must have GTID-based replication enabled.
- A failover event on the DB cluster removes the multi-source replication configuration. Restoring that configuration requires repeating the configuration steps.

Configuring active-active clusters for RDS for MySQL

An active-active cluster in Amazon RDS is a database configuration where multiple nodes actively handle read and write operations, distributing the workload across instances to improve availability and scalability. Each node in the cluster is synchronized to maintain data consistency, enabling high availability and faster failover in case of node failure

You can set up an active-active cluster for RDS for MySQL by using the MySQL Group Replication plugin. The Group Replication plugin is supported for RDS for MySQL DB instances running the following versions:

- All MySQL 8.4 versions
- MySQL 8.0.35 and higher minor versions

For information about MySQL Group Replication, see Group Replication in the MySQL documentation. The MySQL documentation contains detailed information about this feature, while this topic describes how to configure and manage the plugin on your RDS for MySQL DB instances.

> **Note**
>
> For the sake of brevity, all mentions of "active-active" cluster in this topic refer to active-active clusters using the MySQL Group Replication plugin.

Use cases for active-active clusters

The following cases are good candidates for using active-active clusters:

- Applications that need all of the DB instances in the cluster to support write operations. The Group Replication plugin keeps the data consistent on each DB instance in the active-active cluster. For more information about how this works, see Group Replication in the MySQL documentation.
- Applications that require continuous availability of the database. With an active-active cluster, the data is retained on the all of the DB instances in the cluster. If one DB instance fails, the application can reroute traffic to another DB instance in the cluster.
- Applications that might need to split read and write operations among different DB instances in the cluster for load balancing purposes. With an active-active cluster, your applications can

send read traffic to specific DB instances and write traffic to others. You can also switch which DB instances to send reads or writes to at any time.

Topics

- Limitations and considerations for active-active clusters
- Preparing for a cross-VPC active-active cluster
- Required parameter settings for active-active clusters
- Converting an existing DB instance to an active-active cluster
- Setting up an active-active cluster with new DB instances
- Adding a DB instance to an active-active cluster
- Monitoring active-active clusters
- Stopping Group Replication on a DB instance in an active-active cluster
- Renaming a DB instance in an active-active cluster
- Removing a DB instance from an active-active cluster

Limitations and considerations for active-active clusters

Active-active clusters in Amazon RDS offer enhanced availability and scalability by distributing workloads across multiple instances. However, there are important limitations and considerations to keep in mind when using this architecture.

The following sections outline key factors such as replication delays, conflict resolution, resource allocation, and failover behavior. Understanding these considerations can help ensure optimal performance and reliability in active-active cluster deployments.

Topics

- Limitations for RDS for MySQL active-active clusters
- Considerations and best practices for RDS for MySQL active-active clusters

Limitations for RDS for MySQL active-active clusters

The following limitations apply to active-active clusters for RDS for MySQL:

- The master user name can't be `rdsgrprepladmin` for DB instances in an active-active cluster. This user name is reserved for Group Replication connections.

- For DB instances with read replicas in active-active clusters, a prolonged replication status other than `Replicating` can cause log files to exceed storage limits. For information about the status of read replicas, see Monitoring read replication.
- Blue/green deployments aren't supported for DB instances in an active-active cluster. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.
- Kerberos authentication isn't supported for DB instances in an active-active cluster. For more information, see Using Kerberos authentication for Amazon RDS for MySQL.
- The DB instances in a Multi-AZ DB cluster can't be added to an active-active cluster. However, the DB instances in a Multi-AZ DB instance deployment can be added to an active-active cluster. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.
- Tables that don't have a primary key aren't replicated in an active-active cluster because writes are rejected by the Group Replication plugin.
- Non-InnoDB tables aren't replicated in an active-active cluster.
- Active-active clusters don't support concurrent DML and DDL statements on different DB instances in the cluster.
- You can't configure an active-active cluster to use single-primary mode for the group's replication mode. For this configuration, we recommend using a Multi-AZ DB cluster instead. For more information, see Multi-AZ DB cluster deployments for Amazon RDS.
- Multi-source replication isn't supported for DB instances in an active-active cluster.
- A cross-Region active-active cluster can't enforce certificate authority (CA) verification for Group Replication connections.

Considerations and best practices for RDS for MySQL active-active clusters

Before you use RDS for MySQL active-active clusters, review the following considerations and best practices:

- Active-active clusters can't have more than nine DB instances.
- With the Group Replication plugin, you can control the transaction consistency guarantees of the active-active cluster. For more information, see Transaction Consistency Guarantees in the MySQL documentation.
- Conflicts are possible when different DB instances update the same row in an active-active cluster. For information about conflicts and conflict resolution, see Group Replication in the MySQL documentation.

- For fault tolerance, include at least three DB instances in your active-active cluster. It is possible to configure an active-active cluster with only one or two DB instances, but the cluster won't be fault tolerant. For information about fault tolerance, see Fault-tolerance in the MySQL documentation.
- When a DB instance joins an existing active-active cluster and is running the same engine version as the lowest engine version in the cluster, the DB instance joins in read-write mode.
- When a DB instance joins an existing active-active cluster and is running a higher engine version than the lowest engine version in the cluster, the DB instance must remain in read-only mode.
- If you enable Group Replication for a DB instance by setting its `rds.group_replication_enabled` parameter to `1` in the DB parameter group, but replication hasn't started or has failed to start, the DB instance is placed in super-read-only mode to prevent data inconsistencies. For information about super-read-only mode, see the MySQL documentation.
- You can upgrade a DB instance in an active-active cluster, but the DB instance is read-only until all of the other DB instances in the active-active cluster are upgraded to same engine version or a higher engine version. When you upgrade a DB instance, the DB instance automatically joins the same active-active cluster when the upgrade completes. To avoid an unintended switch to read-only mode for a DB instance, disable automatic minor version upgrades for it. For information about upgrading a MySQL DB instance, see Upgrades of the RDS for MySQL DB engine.
- You can add a DB instance in a Multi-AZ DB instance deployment to an existing active-active cluster. You can also convert a Single-AZ DB instance in an active-active cluster to a Multi-AZ DB instance deployment. If a primary DB instance in a Multi-AZ deployment fails, that primary instance fails over to the standby instance. The new primary DB instance automatically joins the same cluster after failover completes. For more information about Multi-AZ DB instance deployments, see Multi-AZ DB instance deployments for Amazon RDS.
- We recommend that the DB instances in an active-active cluster have different time ranges for their maintenance windows. This practice avoids multiple DB instances in the cluster going offline for maintenance at the same time. For more information, see Amazon RDS maintenance window.
- Active-active clusters can use SSL for connections between DB instances. To configure SSL connections, set the `group_replication_recovery_use_ssl` and `group_replication_ssl_mode` parameters. The values for these parameters must match for all DB instances in the active-active cluster.

Currently, active-active clusters don't support certificate authority (CA) verification for connections between AWS Regions. So, the group_replication_ssl_mode parameter must be set to DISABLED (the default) or REQUIRED for cross-Region clusters.

- An RDS for MySQL active-active cluster runs in multi-primary mode. The default value of the group_replication_enforce_update_everywhere_checks is ON and the parameter is static. When this parameter is set to ON, applications can't insert into a table that has cascading foreign key constraints.
- An RDS for MySQL active-active cluster uses the MySQL communication stack for connection security instead of XCOM. For more information, see Communication Stack for Connection Security Management in the MySQL documentation.
- When a DB parameter group is associated with a DB instance in an active-active cluster, we recommend only associating this DB parameter group with other DB instances that are in the cluster.
- Active-active clusters only support RDS for MySQL DB instances. These DB instances must be running supported versions of the DB engine.
- When a DB instance in an active-active cluster has an unexpected failure, RDS starts recovery of the DB instance automatically. If the DB instance doesn't recover, we recommend replacing it with a new DB instance by performing a point-in-time recovery with a healthy DB instance in the cluster. For instructions, see Adding a DB instance to an active-active cluster using point-in-time recovery.
- You can delete a DB instance in an active-active cluster without affecting the other DB instances in the cluster. For information about deleting a DB instance, see Deleting a DB instance.
- When a DB instance unintentionally leaves an active-active cluster, by default the group_replication_exit_state_action parameter changes to OFFLINE_MODE. In this state, the DB instance is inaccessible and you must reboot the DB instance to bring it back online and to rejoin the cluster. You can change this behavior by modifying the group_replication_exit_state_action parameter in a custom parameter group. By setting the parameter to READ_ONLY, when the DB instance unintentionally leaves a cluster, it enters a super read-only state rather than going offline.

Preparing for a cross-VPC active-active cluster

You can configure an active-active cluster with Amazon RDS for MySQL DB instances in more than one VPC. The VPCs can be in the same AWS Region or different AWS Regions.

> **Note**
>
> Sending traffic between multiple AWS Regions might incur additional costs. For more information, see Overview of Data Transfer Costs for Common Architectures.

If you are configuring an active-active cluster in a single VPC, you can skip these steps and move on to Setting up an active-active cluster with new DB instances.

To prepare for an active-active cluster with DB instances in more than one VPC

1. Make sure the IPv4 address ranges in the CIDR blocks meet the following requirements:

   - The IPv4 address ranges in the CIDR blocks of the VPCs can't overlap.
   - All of the IPv4 address ranges in the CIDR blocks either must be lower than `128.0.0.0/subnet_mask` or higher than 128.0.0.0/`subnet_mask`.

   The following ranges illustrate these requirements:

   - `10.1.0.0/16` in one VPC and `10.2.0.0/16` in the other VPC is supported.
   - `172.1.0.0/16` in one VPC and `172.2.0.0/16` in the other VPC is supported.
   - `10.1.0.0/16` in one VPC and `10.1.0.0/16` in the other VPC *is not* supported because the ranges overlap.
   - `10.1.0.0/16` in one VPC and `172.1.0.0/16` in the other VPC *is not* supported because one is below `128.0.0.0/subnet_mask` and the other is above `128.0.0.0/subnet_mask`.

   For information about CIDR blocks, see VPC CIDR blocks in the *Amazon VPC User Guide*.
2. In each VPC, make sure DNS resolution and DNS hostnames are both enabled.

   For instructions, see View and update DNS attributes for your VPC in the *Amazon VPC User Guide*.
3. Configure the VPCs so that you can route traffic between them in one of the following ways:

   - Create a VPC peering connection between the VPCs.

     For instructions, see Create a VPC peering connection in the *Amazon VPC Peering Guide*. In each VPC, make sure there are inbound rules for your security groups that reference

security groups in the peered VPC. Doing so allows traffic to flow to and from instances that are associated with the referenced security group in the peered VPC. For instructions, see Update your security groups to reference peer security groups in the *Amazon VPC Peering Guide*.

- Create a transit gateway between the VPCs.

  For instructions, see Getting started with transit gateways in *Amazon VPC Transit Gateways*. In each VPC, make sure there are inbound rules for your security groups that allow traffic from the other VPC, such as inbound rules that specify the CIDR of the other VPC. Doing so allows traffic to flow to and from instances that are associated with the referenced security group in the active-active cluster. For more information, see Control traffic to your AWS resources using security groups in the *Amazon VPC User Guide*.

Required parameter settings for active-active clusters

Configuring parameters for active-active clusters in Amazon RDS for MySQL is essential for maintaining consistent performance and operational stability. This table details the key parameters that control replication, conflict resolution, and workload distribution. Correct configuration ensures efficient synchronization between nodes, minimizes replication lag, and optimizes resource utilization in distributed or high-traffic environments.

Parameter	Description	Required setting
`binlog_format`	Sets the binary logging format. The default value for RDS for MySQL 8.0 versions and lower is `MIXED`. The default value for RDS for MySQL 8.4 versions is `ROW`. For more information, see the MySQL documentation.	`ROW`
`enforce_gtid_consistency`	Enforces GTID consistency for statement execution. The default value for RDS for MySQL is `OFF`. For more	`ON`

Parameter	Description	Required setting
	information, see the MySQL documentation.	
group_replication_group_name	Sets the Group Replication name to a UUID. The UUID format is 11111111-2222-3333-4444-555555555555. You can generate a MySQL UUID by connecting to a MySQL DB instance and running SELECT UUID(). The value must be the same for all of the DB instances in the active-active cluster. For more information, see the MySQL documentation.	A MySQL UUID
gtid-mode	Controls GTID-based logging. The default value for RDS for MySQL is OFF_PERMISSIVE. For more information, see the MySQL documentation.	ON

Parameter	Description	Required setting
`rds.custom_dns_resolution`	Specifies whether to allow DNS resolution from the Amazon DNS server in your VPC. DNS resolution must be enabled when Group Replication is enabled with the `rds.group_replication_enabled` parameter. DNS resolution can't be enabled when Group Replication is disabled with the `rds.group_replication_enabled` parameter. For more information, see Amazon DNS server in the *Amazon VPC User Guide*.	1
`rds.group_replication_enabled`	Specifies whether Group Replication is enabled for a DB instance. Group Replication must be enabled on a DB instance in an active-active cluster.	1
`replica_preserve_commit_order` (RDS for MySQL 8.4 and higher versions) or `slave_preserve_commit_order` (RDS for MySQL 8.0 versions)	Controls the order that transactions are committed on a replica. The default value for RDS for MySQL is `ON`. For more information, see the MySQL documentation.	`ON`

Converting an existing DB instance to an active-active cluster

The DB engine version of the DB instance you want to migrate to an active-active cluster must be one of the following versions:

- All MySQL 8.4 versions
- MySQL 8.0.35 and higher minor versions

If you need to upgrade the engine version, see Upgrades of the RDS for MySQL DB engine.

If you are setting up an active-active cluster with DB instances in more than one VPC, make sure you complete the prerequisites in Preparing for a cross-VPC active-active cluster.

Complete the following steps to migrate an existing DB instance to an active-active cluster for RDS for MySQL.

Topics

- Step 1: Set the active-active cluster parameters in one or more custom parameter groups
- Step 2: Associate the DB instance with a DB parameter group that has the required Group Replication parameters set
- Step 3: Create the active-active cluster
- Step 4: Create additional RDS for MySQL DB instances for the active-active cluster
- Step 5: Initialize the group on the DB instance you are converting
- Step 6: Start replication on the other DB instances in the active-active cluster
- Step 7: (Recommended) Check the status of the active-active cluster

Step 1: Set the active-active cluster parameters in one or more custom parameter groups

The RDS for MySQL DB instances in an active-active cluster must be associated with a custom parameter group that has the correct setting for required parameters. For information about the parameters and the required setting for each one, see Required parameter settings for active-active clusters.

You can set these parameters in new parameter groups or in existing parameter groups. However, to avoid accidentally affecting DB instances that aren't part of the active-active cluster, we strongly

recommend that you create a new custom parameter group. The DB instances in an active-active cluster can be associated with the same DB parameter group or with different DB parameter groups.

You can use the AWS Management Console or the AWS CLI to create a new custom parameter group. For more information, see Creating a DB parameter group in Amazon RDS. The following example runs the create-db-parameter-group AWS CLI command to create a custom DB parameter group named *myactivepg* for RDS for MySQL 8.0:

For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
  --db-parameter-group-name myactivepg \
  --db-parameter-group-family mysql8.0 \
  --description "Parameter group for active-active clusters"
```

For Windows:

```
aws rds create-db-parameter-group ^
  --db-parameter-group-name myactivepg ^
  --db-parameter-group-family mysql8.0 ^
  --description "Parameter group for active-active clusters"
```

You can also use the AWS Management Console or the AWS CLI to set the parameters in the custom parameter group. For more information, see Modifying parameters in a DB parameter group in Amazon RDS.

The following example runs the modify-db-parameter-group AWS CLI command to set the parameters for RDS for MySQL 8.0. To use this example with RDS for MySQL 8.4, change `slave_preserve_commit_order` to `replica_preserve_commit_order`.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myactivepg \
  --parameters
 "ParameterName='rds.group_replication_enabled',ParameterValue='1',ApplyMethod=pending-
reboot" \

 "ParameterName='rds.custom_dns_resolution',ParameterValue='1',ApplyMethod=pending-
reboot" \
```

```
 "ParameterName='enforce_gtid_consistency',ParameterValue='ON',ApplyMethod=pending-
reboot" \
               "ParameterName='gtid-mode',ParameterValue='ON',ApplyMethod=pending-
reboot" \

 "ParameterName='binlog_format',ParameterValue='ROW',ApplyMethod=immediate" \

 "ParameterName='slave_preserve_commit_order',ParameterValue='ON',ApplyMethod=immediate"
 \

 "ParameterName='group_replication_group_name',ParameterValue='11111111-2222-3333-4444-55555555
reboot"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myactivepg ^
  --parameters
 "ParameterName='rds.group_replication_enabled',ParameterValue='1',ApplyMethod=pending-
reboot" ^

 "ParameterName='rds.custom_dns_resolution',ParameterValue='1',ApplyMethod=pending-
reboot" ^

 "ParameterName='enforce_gtid_consistency',ParameterValue='ON',ApplyMethod=pending-
reboot" ^
               "ParameterName='gtid-mode',ParameterValue='ON',ApplyMethod=pending-
reboot" ^

 "ParameterName='binlog_format',ParameterValue='ROW',ApplyMethod=immediate" ^

 "ParameterName='slave_preserve_commit_order',ParameterValue='ON',ApplyMethod=immediate"
 ^

 "ParameterName='group_replication_group_name',ParameterValue='11111111-2222-3333-4444-55555555
reboot"
```

Step 2: Associate the DB instance with a DB parameter group that has the required Group Replication parameters set

Associate the DB instance with a parameter group you created or modified in the previous step. For instructions, see Associating a DB parameter group with a DB instance in Amazon RDS.

Reboot the DB instance for the new parameter settings to take effect. For instructions, see Rebooting a DB instance.

Step 3: Create the active-active cluster

In the DB parameter group associated with the DB instance, set the `group_replication_group_seeds` parameter to the endpoint of the DB instance you are converting.

You can use the AWS Management Console or the AWS CLI to set the parameter. You don't need to reboot the DB instance after setting this parameter. For more information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

The following example runs the modify-db-parameter-group AWS CLI command to set the parameters:

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myactivepg \
  --parameters
 "ParameterName='group_replication_group_seeds',ParameterValue='myactivedb1.123456789012.us-east-1.rds.amazonaws.com:3306',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myactivepg ^
  --parameters
 "ParameterName='group_replication_group_seeds',ParameterValue='myactivedb1.123456789012.us-east-1.rds.amazonaws.com:3306',ApplyMethod=immediate"
```

Step 4: Create additional RDS for MySQL DB instances for the active-active cluster

To create additional DB instances for the active-active cluster, perform point-in-time recovery on the DB instance you are converting. For instructions, see Adding a DB instance to an active-active cluster using point-in-time recovery.

An active-active cluster can have up to nine DB instances. Perform point-in-time recovery on the DB instance until you have the number of DB instances you want for the cluster. When you perform point-in-recovery, make sure you associate the DB instance you are adding with a DB parameter group that has `rds.group_replication_enabled` set to 1. Otherwise, Group Replication won't start on the newly added DB instance.

Step 5: Initialize the group on the DB instance you are converting

Initialize the group and start replication:

1. Connect to that DB instance you are converting in a SQL client. For more information about connecting to an RDS for MySQL DB instance, see Connecting to your MySQL DB instance.
2. In the SQL client, run the following stored procedures and replace *`group_replication_user_password`* with the password for the `rdsgrprepladmin` user. The `rdsgrprepladmin` user is reserved for Group Replication connections in an active-active cluster. The password for this user must be the same on all of the DB instances in an active-active cluster.

```
call mysql.rds_set_configuration('binlog retention hours', 168); -- 7 days binlog
call mysql.rds_group_replication_create_user('group_replication_user_password');
call
 mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
call mysql.rds_group_replication_start(1);
```

   This example sets the `binlog retention hours` value to 168, which means that binary log files are retained for seven days on the DB instance. You can adjust this value to meet your requirements.

   This example specifies 1 in the `mysql.rds_group_replication_start` stored procedure to initialize a new group with the current DB instance.

   For more information about the stored procedures called in the example, see Managing active-active clusters.

Step 6: Start replication on the other DB instances in the active-active cluster

For each of the DB instances in the active-active cluster, use a SQL client to connect to the instance, and run the following stored procedures. Replace *group_replication_user_password* with the password for the `rdsgrprepladmin` user.

```
call mysql.rds_set_configuration('binlog retention hours', 168); -- 7 days binlog
call mysql.rds_group_replication_create_user('group_replication_user_password');
call
 mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
call mysql.rds_group_replication_start(0);
```

This example sets the `binlog retention hours` value to 168, which means that binary log files are retained for seven days on each DB instance. You can adjust this value to meet your requirements.

This example specifies `0` in the `mysql.rds_group_replication_start` stored procedure to join the current DB instance to an existing group.

> **Tip**
>
> Make sure you run these stored procedures on all of the other DB instances in the active-active cluster.

Step 7: (Recommended) Check the status of the active-active cluster

To make sure each member of the cluster is configured correctly, check the status of the cluster by connecting to a DB instance in the active-active cluster, and running the following SQL command:

```
SELECT * FROM performance_schema.replication_group_members;
```

Your output should show `ONLINE` for the `MEMBER_STATE` of each DB instance, as in the following sample output:

```
+---------------------------+--------------------------------------
+---------------+-------------+--------------+-------------+---------------
+----------------------------+
```

```
| CHANNEL_NAME               | MEMBER_ID                            | MEMBER_HOST    |
 MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION | MEMBER_COMMUNICATION_STACK
 |
+---------------------------+--------------------------------------
+---------------+-------------+--------------+-------------+---------------
+----------------------------+
| group_replication_applier | 9854d4a2-5d7f-11ee-b8ec-0ec88c43c251 | ip-10-15-3-137 |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
| group_replication_applier | 9e2e9c28-5d7f-11ee-8039-0e5d58f05fef | ip-10-15-3-225 |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
| group_replication_applier | a6ba332d-5d7f-11ee-a025-0a5c6971197d | ip-10-15-1-83  |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
+---------------------------+--------------------------------------
+---------------+-------------+--------------+-------------+---------------
+----------------------------+
3 rows in set (0.00 sec)
```

For information about the possible MEMBER_STATE values, see Group Replication Server States in the MySQL documentation.

Setting up an active-active cluster with new DB instances

Complete the following steps to set up an active-active cluster using new Amazon RDS for MySQL DB instances.

If you are setting up an active-active cluster with DB instances in more than one VPC, make sure you complete the prerequisites in Preparing for a cross-VPC active-active cluster.

Topics

- Step 1: Set the active-active cluster parameters in one or more custom parameter groups
- Step 2: Create new RDS for MySQL DB instances for the active-active cluster
- Step 3: Specify the DB instances in the active-active cluster
- Step 4: Initialize the group on a DB instance and start replication
- Step 5: Start replication on the other DB instances in the active-active cluster
- Step 6: (Recommended) Check the status of the active-active cluster
- Step 7: (Optional) Import data into a DB instance in the active-active cluster

Step 1: Set the active-active cluster parameters in one or more custom parameter groups

The RDS for MySQL DB instances in an active-active cluster must be associated with a custom parameter group that has the correct setting for required parameters. For information about the parameters and the required setting for each one, see Required parameter settings for active-active clusters.

You can set these parameters in new parameter groups or in existing parameter groups. However, to avoid accidentally affecting DB instances that aren't part of the active-active cluster, we strongly recommend that you create a new custom parameter group. The DB instances in an active-active cluster can be associated with the same DB parameter group or with different DB parameter groups.

You can use the AWS Management Console or the AWS CLI to create a new custom parameter group. For more information, see Creating a DB parameter group in Amazon RDS. The following example runs the create-db-parameter-group AWS CLI command to create a custom DB parameter group named *myactivepg* for RDS for MySQL 8.0:

For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
  --db-parameter-group-name myactivepg \
  --db-parameter-group-family mysql8.0 \
  --description "Parameter group for active-active clusters"
```

For Windows:

```
aws rds create-db-parameter-group ^
  --db-parameter-group-name myactivepg ^
  --db-parameter-group-family mysql8.0 ^
  --description "Parameter group for active-active clusters"
```

You can also use the AWS Management Console or the AWS CLI to set the parameters in the custom parameter group. For more information, see Modifying parameters in a DB parameter group in Amazon RDS.

The following example runs the modify-db-parameter-group AWS CLI command to set the parameters for RDS for MySQL 8.0. To use this example with RDS for MySQL 8.4, change `slave_preserve_commit_order` to `replica_preserve_commit_order`.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myactivepg \
  --parameters
 "ParameterName='rds.group_replication_enabled',ParameterValue='1',ApplyMethod=pending-
reboot" \

 "ParameterName='rds.custom_dns_resolution',ParameterValue='1',ApplyMethod=pending-
reboot" \

 "ParameterName='enforce_gtid_consistency',ParameterValue='ON',ApplyMethod=pending-
reboot" \
               "ParameterName='gtid-mode',ParameterValue='ON',ApplyMethod=pending-
reboot" \

 "ParameterName='binlog_format',ParameterValue='ROW',ApplyMethod=immediate" \

 "ParameterName='slave_preserve_commit_order',ParameterValue='ON',ApplyMethod=immediate"
 \

 "ParameterName='group_replication_group_name',ParameterValue='11111111-2222-3333-4444-55555555
reboot"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myactivepg ^
  --parameters
 "ParameterName='rds.group_replication_enabled',ParameterValue='1',ApplyMethod=pending-
reboot" ^

 "ParameterName='rds.custom_dns_resolution',ParameterValue='1',ApplyMethod=pending-
reboot" ^

 "ParameterName='enforce_gtid_consistency',ParameterValue='ON',ApplyMethod=pending-
reboot" ^
               "ParameterName='gtid-mode',ParameterValue='ON',ApplyMethod=pending-
reboot" ^

 "ParameterName='binlog_format',ParameterValue='ROW',ApplyMethod=immediate" ^
```

```
 "ParameterName='slave_preserve_commit_order',ParameterValue='ON',ApplyMethod=immediate"
 ^

 "ParameterName='group_replication_group_name',ParameterValue='11111111-2222-3333-4444-55555555
reboot"
```

Step 2: Create new RDS for MySQL DB instances for the active-active cluster

Active-active clusters are supported for the following versions of RDS for MySQL DB instances:

- All MySQL version 8.4 versions
- MySQL version 8.0.35 and higher minor versions

You can create up to nine new DB instances for the cluster.

You can use the AWS Management Console or the AWS CLI to create new DB instances. For more information about creating a DB instance, see Creating an Amazon RDS DB instance. When you create the DB instance, associate it with a DB parameter group that you created or modified in the previous step.

Step 3: Specify the DB instances in the active-active cluster

In the DB parameter group associated with each DB instance, set the `group_replication_group_seeds` parameter to the endpoints of the DB instances you want to include in the cluster.

You can use the AWS Management Console or the AWS CLI to set the parameter. You don't need to reboot the DB instance after setting this parameter. For more information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

The following example runs the modify-db-parameter-group AWS CLI command to set the parameters:

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name myactivepg \
  --parameters
 "ParameterName='group_replication_group_seeds',ParameterValue='myactivedb1.123456789012.us-
```

```
east-1.rds.amazonaws.com:3306,myactivedb2.123456789012.us-
east-1.rds.amazonaws.com:3306,myactivedb3.123456789012.us-
east-1.rds.amazonaws.com:3306',ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
  --db-parameter-group-name myactivepg ^
  --parameters
 "ParameterName='group_replication_group_seeds',ParameterValue='myactivedb1.123456789012.us-
east-1.rds.amazonaws.com:3306,myactivedb2.123456789012.us-
east-1.rds.amazonaws.com:3306,myactivedb3.123456789012.us-
east-1.rds.amazonaws.com:3306',ApplyMethod=immediate"
```

Tip

Make sure you set the `group_replication_group_seeds` parameter in each DB parameter group that is associated with a DB instance in the active-active cluster.

Step 4: Initialize the group on a DB instance and start replication

You can choose any new DB to initialize the group and start replication. To do so, complete the following steps:

1. Choose a DB instance in the active-active cluster, and connect to that DB instance in a SQL client. For more information about connecting to an RDS for MySQL DB instance, see Connecting to your MySQL DB instance.
2. In the SQL client, run the following stored procedures and replace *group_replication_user_password* with the password for the `rdsgrprepladmin` user. The `rdsgrprepladmin` user is reserved for Group Replication connections in an active-active cluster. The password for this user must be the same on all of the DB instances in an active-active cluster.

   ```
   call mysql.rds_set_configuration('binlog retention hours', 168); -- 7 days binlog
   call mysql.rds_group_replication_create_user('group_replication_user_password');
   call
    mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
   call mysql.rds_group_replication_start(1);
   ```

This example sets the `binlog retention hours` value to 168, which means that binary log files are retained for seven days on the DB instance. You can adjust this value to meet your requirements.

This example specifies 1 in the `mysql.rds_group_replication_start` stored procedure to initialize a new group with the current DB instance.

For more information about the stored procedures called in the example, see Managing active-active clusters.

Step 5: Start replication on the other DB instances in the active-active cluster

For each of the DB instances in the active-active cluster, use a SQL client to connect to the instance, and run the following stored procedures. Replace *`group_replication_user_password`* with the password for the `rdsgrprepladmin` user.

```
call mysql.rds_set_configuration('binlog retention hours', 168); -- 7 days binlog
call mysql.rds_group_replication_create_user('group_replication_user_password');
call
 mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
call mysql.rds_group_replication_start(0);
```

This example sets the `binlog retention hours` value to 168, which means that binary log files are retained for seven days on each DB instance. You can adjust this value to meet your requirements.

This example specifies `0` in the `mysql.rds_group_replication_start` stored procedure to join the current DB instance to an existing group.

> **Tip**
>
> Make sure you run these stored procedures on all of the other DB instances in the active-active cluster.

Step 6: (Recommended) Check the status of the active-active cluster

To make sure each member of the cluster is configured correctly, check the status of the cluster by connecting to a DB instance in the active-active cluster, and running the following SQL command:

```
SELECT * FROM performance_schema.replication_group_members;
```

Your output should show `ONLINE` for the `MEMBER_STATE` of each DB instance, as in the following sample output:

```
+---------------------------+--------------------------------------
+---------------+-------------+-------------+-------------+---------------
+-----------------------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST    |
 MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION | MEMBER_COMMUNICATION_STACK
 |
+---------------------------+--------------------------------------
+---------------+-------------+-------------+-------------+---------------
+-----------------------------+
| group_replication_applier | 9854d4a2-5d7f-11ee-b8ec-0ec88c43c251 | ip-10-15-3-137 |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
| group_replication_applier | 9e2e9c28-5d7f-11ee-8039-0e5d58f05fef | ip-10-15-3-225 |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
| group_replication_applier | a6ba332d-5d7f-11ee-a025-0a5c6971197d | ip-10-15-1-83  |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
+---------------------------+--------------------------------------
+---------------+-------------+-------------+-------------+---------------
+-----------------------------+
3 rows in set (0.00 sec)
```

For information about the possible `MEMBER_STATE` values, see Group Replication Server States in the MySQL documentation.

Step 7: (Optional) Import data into a DB instance in the active-active cluster

You can import data from a MySQL database into a DB instance in the active-active cluster. After the data is imported, Group Replication replicates it to the other DB instances in the cluster.

For information about importing data, see Importing data to an Amazon RDS for MySQL database with reduced downtime.

Adding a DB instance to an active-active cluster

You can add a DB instance to an Amazon RDS for MySQL active-active cluster by restoring a DB snapshot or by restoring a DB instance to a point in time. An active-active cluster can include up to nine DB instances.

When you recover a DB instance to a point in time, it usually includes more recent transactions than a DB instance that was restored from a DB snapshot. When the DB instance has more recent transactions, fewer transactions need to be applied when you start replication. So, using point-in-time recovery to add a DB instance to a cluster is usually faster than restoring from a DB snapshot.

Topics

- Adding a DB instance to an active-active cluster using point-in-time recovery
- Adding a DB instance to an active-active cluster using a DB snapshot

Adding a DB instance to an active-active cluster using point-in-time recovery

You can add a DB instance to an active-active cluster by performing point-in-time recovery on a DB instance in the cluster.

For information about recovering a DB instance to a point in time in a different AWS Region, see Replicating automated backups to another AWS Region.

To add a DB instance to an active-active cluster using point-in-time recovery

1. Create a new DB instance by performing point-in-time recovery on a DB instance in the active-active cluster.

   You can perform point-in-time recovery on any DB instance in the cluster to create the new DB instance. For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

   > **Important**
   >
   > During point-in-time-recovery, associate the new DB instance with a DB parameter group that has the active-active cluster parameters set. Otherwise, Group Replication won't start on the new DB instance. For information about the parameters and the required setting for each one, see Required parameter settings for active-active clusters.

> **Tip**
>
> If you take a snapshot of the DB instance before you start point-in-time recovery, you might be able to reduce the amount of time required to apply transactions on the new DB instance.

2. Add the DB instance to the `group_replication_group_seeds` parameter in each DB parameter group associated with a DB instance in the active-active cluster, including the DB parameter group that you associated with the new DB instance.

   For more information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

3. In a SQL client, connect to the new DB instance, and call the mysql.rds_group_replication_set_recovery_channel stored procedure. Replace *`group_replication_user_password`* with the password for the `rdsgrprepladmin` user.

```
call
 mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
```

4. Using the SQL client, call the mysql.rds_group_replication_start stored procedure to start replication:

```
call mysql.rds_group_replication_start(0);
```

Adding a DB instance to an active-active cluster using a DB snapshot

You can add a DB instance to an active-active cluster by creating a DB snapshot of a DB instance in the cluster and then restoring the DB snapshot.

For information about copying a snapshot to a different AWS Region, see the section called “Cross-Region copying”.

To add a DB instance to an active-active cluster using a DB snapshot

1. Create a DB snapshot of a DB instance in the active-active cluster.

You can create a DB snapshot of any DB instance in the cluster. For instructions, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

2. Restore a DB instance from the DB snapshot.

   During the snapshot restore operation, associate the new DB instance with a DB parameter group that has the active-active cluster parameters set. For information about the parameters and the required setting for each one, see Required parameter settings for active-active clusters.

   For information about restoring a DB instance from a DB snapshot, see Restoring to a DB instance.

3. Add the DB instance to the `group_replication_group_seeds` parameter in each DB parameter group associated with a DB instance in the active-active cluster, including the DB parameter group that you associated with the new DB instance.

   For more information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

4. In a SQL client, connect to the new DB instance, and call the mysql.rds_group_replication_set_recovery_channel stored procedure. Replace *`group_replication_user_password`* with the password for the `rdsgrprepladmin` user.

   ```
   call
    mysql.rds_group_replication_set_recovery_channel('group_replication_user_password');
   ```

5. Using the SQL client, call the mysql.rds_group_replication_start stored procedure to start replication:

   ```
   call mysql.rds_group_replication_start(0);
   ```

Monitoring active-active clusters

Monitoring active-active clusters in Amazon RDS for MySQL is crucial for tracking performance, replication integrity, and node synchronization. You can monitor your active-active cluster by connecting to a DB instance in the cluster, and running the following SQL command:

```
SELECT * FROM performance_schema.replication_group_members;
```

Your output should show ONLINE for the MEMBER_STATE of each DB instance, as in the following sample output:

```
+---------------------------+--------------------------------------
+---------------+-------------+--------------+-------------+---------------
+----------------------------+
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST    |
 MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION | MEMBER_COMMUNICATION_STACK
 |
+---------------------------+--------------------------------------
+---------------+-------------+--------------+-------------+---------------
+----------------------------+
| group_replication_applier | 9854d4a2-5d7f-11ee-b8ec-0ec88c43c251 | ip-10-15-3-137 |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
| group_replication_applier | 9e2e9c28-5d7f-11ee-8039-0e5d58f05fef | ip-10-15-3-225 |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
| group_replication_applier | a6ba332d-5d7f-11ee-a025-0a5c6971197d | ip-10-15-1-83  |
      3306 | ONLINE       | PRIMARY     | 8.0.35         | MySQL                       |
+---------------------------+--------------------------------------
+---------------+-------------+--------------+-------------+---------------
+----------------------------+
3 rows in set (0.00 sec)
```

For information about the possible MEMBER_STATE values, see Group Replication Server States in the MySQL documentation.

Stopping Group Replication on a DB instance in an active-active cluster

You can stop Group Replication on a DB instance in an active-active cluster. When you stop Group Replication, the DB instance is placed in super-read-only mode until replication is restarted or that DB instance is removed from the active-active cluster. For information about super-read-only mode, see the MySQL documentation.

To stop Group Replication temporarily for an active-active cluster

1. Connect to a DB instance in the active-active cluster using a SQL client.

   For more information about connecting to an RDS for MySQL DB instance, see Connecting to your MySQL DB instance.

2. In the SQL client, call the mysql.rds_group_replication_stop stored procedure:

```
call mysql.rds_group_replication_stop();
```

Renaming a DB instance in an active-active cluster

You can change the name of a DB instance in an active-active cluster. To rename more than one DB instance in an active-active cluster, do so one DB instance at a time. So, rename one DB instance and rejoin it to the cluster before you rename the next DB instance.

To rename a DB instance in an active-active cluster

1. Connect to the DB instance in a SQL client, and call the mysql.rds_group_replication_stop stored procedure:

   ```
   call mysql.rds_group_replication_stop();
   ```

2. Rename the DB instance by following the instructions in Renaming a DB instance.
3. Modify the `group_replication_group_seeds` parameter in each DB parameter group associated with a DB instance in the active-active cluster.

   In the parameter setting, replace the old DB instance endpoint with the new DB instance endpoint. For more information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.
4. Connect to the DB instance in a SQL client, and call the mysql.rds_group_replication_start stored procedure:

   ```
   call mysql.rds_group_replication_start(0);
   ```

Removing a DB instance from an active-active cluster

When you remove a DB instance from an active-active cluster, it reverts to a standalone DB instance.

To remove a DB instance from an active-active cluster

1. Connect to the DB instance in a SQL client, and call the mysql.rds_group_replication_stop stored procedure:

```
call mysql.rds_group_replication_stop();
```

2. Modify the `group_replication_group_seeds` parameter for the DB instances that will remain in the active-active cluster.

   In the `group_replication_group_seeds` parameter, delete the DB instance that you are removing from the active-active cluster. For more information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.
3. Modify the parameters of the DB instance you are removing from the active-active cluster so that it is no longer part of the cluster.

   You can either associate the DB instance with a different parameter group, or modify the parameters in the DB parameter group associated with the DB instance. The parameters to modify include `group_replication_group_name`, `rds.group_replication_enabled`, and `group_replication_group_seeds`. For more information about active-active cluster parameters, see Required parameter settings for active-active clusters.

   If you modify the parameters in a DB parameter group, make sure the DB parameter group isn't associated with other DB instances in the active-active cluster.
4. Reboot the DB instance you removed from the active-active cluster for the new parameter settings to take effect.

   For instructions, see Rebooting a DB instance.

Exporting data from a MySQL DB instance by using replication

To export data from an RDS for MySQL DB instance to a MySQL instance running external to Amazon RDS, you can use replication. In this scenario, the MySQL DB instance is the *source MySQL DB instance*, and the MySQL instance running external to Amazon RDS is the *external MySQL database*.

The external MySQL database can run either on-premises in your data center, or on an Amazon EC2 instance. The external MySQL database must run the same version as the source MySQL DB instance, or a later version.

Replication to an external MySQL database is only supported during the time it takes to export a database from the source MySQL DB instance. The replication should be terminated when the data has been exported and applications can start accessing the external MySQL instance.

The following list shows the steps to take. Each step is discussed in more detail in later sections.

1. Prepare an external MySQL DB instance.
2. Prepare the source MySQL DB instance for replication.
3. Use the mysqldump utility to transfer the database from the source MySQL DB instance to the external MySQL database.
4. Start replication to the external MySQL database.
5. After the export completes, stop replication.

Prepare an external MySQL database

Perform the following steps to prepare the external MySQL database.

To prepare the external MySQL database

1. Install the external MySQL database.
2. Connect to the external MySQL database as the master user. Then create the users required to support the administrators, applications, and services that access the database.
3. Follow the directions in the MySQL documentation to prepare the external MySQL database as a replica. For more information, see Setting the Replica Configuration in the MySQL documentation.

4. Configure an egress rule for the external MySQL database to operate as a read replica during the export. The egress rule allows the external MySQL database to connect to the source MySQL DB instance during replication. Specify an egress rule that allows Transmission Control Protocol (TCP) connections to the port and IP address of the source MySQL DB instance.

   Specify the appropriate egress rules for your environment:

   - If the external MySQL database is running in an Amazon EC2 instance in a virtual private cloud (VPC) based on the Amazon VPC service, specify the egress rules in a VPC security group. For more information, see Controlling access with security groups.
   - If the external MySQL database is installed on-premises, specify the egress rules in a firewall.
5. If the external MySQL database is running in a VPC, configure rules for the VPC access control list (ACL) rules in addition to the security group egress rule:

   - Configure an ACL ingress rule allowing TCP traffic to ports 1024–65535 from the IP address of the source MySQL DB instance.
   - Configure an ACL egress rule allowing outbound TCP traffic to the port and IP address of the source MySQL DB instance.

   For more information about Amazon VPC network ACLs, see Network ACLs in *Amazon VPC User Guide.*
6. (Optional) Set the `max_allowed_packet` parameter to the maximum size to avoid replication errors. We recommend this setting.

Prepare the source MySQL DB instance

Perform the following steps to prepare the source MySQL DB instance as the replication source.

To prepare the source MySQL DB instance

1. Ensure that your client computer has enough disk space available to save the binary logs while setting up replication.
2. Connect to the source MySQL DB instance, and create a replication account by following the directions in Creating a User for Replication in the MySQL documentation.
3. Configure ingress rules on the system running the source MySQL DB instance to allow the external MySQL database to connect during replication. Specify an ingress rule that allows

TCP connections to the port used by the source MySQL DB instance from the IP address of the external MySQL database.

4. Specify the egress rules:

   - If the source MySQL DB instance is running in a VPC, specify the ingress rules in a VPC security group. For more information, see Controlling access with security groups.

5. If source MySQL DB instance is running in a VPC, configure VPC ACL rules in addition to the security group ingress rule:

   - Configure an ACL ingress rule to allow TCP connections to the port used by the Amazon RDS instance from the IP address of the external MySQL database.
   - Configure an ACL egress rule to allow TCP connections from ports 1024–65535 to the IP address of the external MySQL database.

   For more information about Amazon VPC network ACLs, see Network ACLs in the *Amazon VPC User Guide.*

6. Ensure that the backup retention period is set long enough that no binary logs are purged during the export. If any of the logs are purged before the export has completed, you must restart replication from the beginning. For more information about setting the backup retention period, see Introduction to backups.

7. Use the `mysql.rds_set_configuration` stored procedure to set the binary log retention period long enough that the binary logs aren't purged during the export. For more information, see Accessing MySQL binary logs.

8. Create an Amazon RDS read replica from the source MySQL DB instance to further ensure that the binary logs of the source MySQL DB instance are not purged. For more information, see Creating a read replica.

9. After the Amazon RDS read replica has been created, call the `mysql.rds_stop_replication` stored procedure to stop the replication process. The source MySQL DB instance no longer purges its binary log files, so they are available for the replication process.

10. (Optional) Set both the `max_allowed_packet` parameter and the `slave_max_allowed_packet` parameter to the maximum size to avoid replication errors. The maximum size for both parameters is 1 GB. We recommend this setting for both parameters. For information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Copy the database

Perform the following steps to copy the database.

To copy the database

1. Connect to the RDS read replica of the source MySQL DB instance, and run the MySQL `SHOW REPLICA STATUS\G` statement. Note the values for the following:

   - `Master_Host`
   - `Master_Port`
   - `Master_Log_File`
   - `Exec_Master_Log_Pos`

   > **Note**
   >
   > Previous versions of MySQL used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MySQL version before 8.0.23, then use `SHOW SLAVE STATUS`.

2. Use the mysqldump utility to create a snapshot, which copies the data from Amazon RDS to your local client computer. Ensure that your client computer has enough space to hold the `mysqldump` files from the databases to be replicated. This process can take several hours for very large databases. Follow the directions in Creating a Data Snapshot Using mysqldump in the MySQL documentation.

   The following example runs `mysqldump` on a client and writes the dump to a file.

   For Linux, macOS, or Unix:

   ```
   mysqldump -h source_MySQL_DB_instance_endpoint \
       -u user \
       -ppassword \
       --port=3306 \
       --single-transaction \
       --routines \
       --triggers \
       --databases  database database2 > path/rds-dump.sql
   ```

For Windows:

```
mysqldump -h source_MySQL_DB_instance_endpoint ^
    -u user ^
    -ppassword ^
    --port=3306 ^
    --single-transaction ^
    --routines ^
    --triggers ^
    --databases  database database2 > path\rds-dump.sql
```

You can load the backup file into the external MySQL database. For more information, see Reloading SQL-Format Backups in the MySQL documentation. You can run another utility to load the data into the external MySQL database.

Complete the export

Perform the following steps to complete the export.

To complete the export

1. Use the MySQL `CHANGE MASTER` statement to configure the external MySQL database. Specify the ID and password of the user granted `REPLICATION SLAVE` permissions. Specify the `Master_Host`, `Master_Port`, `Relay_Master_Log_File`, and `Exec_Master_Log_Pos` values that you got from the MySQL `SHOW REPLICA STATUS\G` statement that you ran on the RDS read replica. For more information, see CHANGE MASTER TO Statement in the MySQL documentation.

   > **Note**
   >
   > Previous versions of MySQL used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MySQL version before 8.0.23, then use `SHOW SLAVE STATUS`.

2. Use the MySQL `START REPLICA` command to initiate replication from the source MySQL DB instance to the external MySQL database.

Doing this starts replication from the source MySQL DB instance and exports all source changes that have occurred after you stopped replication from the Amazon RDS read replica.

> **Note**
>
> Previous versions of MySQL used `START SLAVE` instead of `START REPLICA`. If you are using a MySQL version before 8.0.23, then use `START SLAVE`.

3. Run the MySQL `SHOW REPLICA STATUS\G` command on the external MySQL database to verify that it is operating as a read replica. For more information about interpreting the results, see SHOW SLAVE | REPLICA STATUS Statement in the MySQL documentation.
4. After replication on the external MySQL database has caught up with the source MySQL DB instance, use the MySQL `STOP REPLICA` command to stop replication from the source MySQL DB instance.

> **Note**
>
> Previous versions of MySQL used `STOP SLAVE` instead of `STOP REPLICA`. If you are using a MySQL version before 8.0.23, then use `STOP SLAVE`.

5. On the Amazon RDS read replica, call the `mysql.rds_start_replication` stored procedure. Doing this allows Amazon RDS to start purging the binary log files from the source MySQL DB instance.

Options for MySQL DB instances

Following, you can find a description of options, or additional features, that are available for Amazon RDS instances running the MySQL DB engine. To enable these options, you can add them to a custom option group, and then associate the option group with your DB instance. For more information about working with option groups, see Working with option groups.

Amazon RDS supports the following options for MySQL:

Option	Option ID	Engine versions
MariaDB Audit Plugin support for MySQL	`MARIADB_AUDIT_PLUGIN`	All MySQL 8.4 versions MySQL 8.0.28 and higher 8.0 versions All MySQL 5.7 versions
MySQL memcached support	`MEMCACHED`	All MySQL 5.7 and 8.0 versions

MariaDB Audit Plugin support for MySQL

Amazon RDS offers an audit plugin for MySQL database instances based on the open source MariaDB Audit Plugin. For more information, see the Audit Plugin for MySQL Server GitHub repository.

> **Note**
>
> The audit plugin for MySQL is based on the MariaDB Audit Plugin. Throughout this article, we refer to it as MariaDB Audit Plugin.

The MariaDB Audit Plugin records database activity, including users logging on to the database and queries run against the database. The record of database activity is stored in a log file.

Audit Plugin option settings

Amazon RDS supports the following settings for the MariaDB Audit Plugin option.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_FILE_PATH	/rdsdbdata/log/audit/	/rdsdbdata/log/audit/	The location of the log file. The log file contains the record of the activity specified in SERVER_AUDIT_EVENTS . For more information, see Viewing and listing database log files and MySQL database log files.
SERVER_AUDIT_FILE_ROTATE_SIZE	1–1000000000	1000000	The size in bytes that when reached, causes the file to rotate. For more information, see Overview of RDS for MySQL database logs.
SERVER_AUDIT_FILE_ROTATIONS	0–100	9	The number of log rotations to save when server_audit_output_type=file . If set to 0, then the log file never rotates. For more information, see Overview of RDS for

Option setting	Valid values	Default value	Description
			MySQL database logs and Downloading a database log file.
SERVER_AUDIT_EVENTS	CONNECT, QUERY, QUERY_DDL , QUERY_DML , QUERY_DML_NO_SELECT , QUERY_DCL	CONNECT, QUERY	The types of activity to record in the log. Installing the MariaDB Audit Plugin is itself logged. • CONNECT: Log successful and unsuccessful connections to the database, and disconnections from the database. • QUERY: Log the text of all queries run against the database. • QUERY_DDL : Similar to the QUERY event, but returns only data definition language (DDL) queries (CREATE, ALTER, and so on). • QUERY_DML : Similar to the QUERY event, but returns only data manipulation language (DML) queries (INSERT, UPDATE, and so on, and also SELECT). • QUERY_DML_NO_SELECT : Similar to the QUERY_DML event, but doesn't log SELECT queries. • QUERY_DCL : Similar to the QUERY event, but returns only data control language (DCL) queries (GRANT, REVOKE, and so on). For MySQL, TABLE is not supported.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_INCL_USERS	Multiple comma-separated values	None	Include only activity from the specified users. By default, activity is recorded for all users. SERVER_AUDIT_INCL_USERS and SERVER_AUDIT_EXCL_USERS are mutually exclusive. If you add values to SERVER_AUDIT_INCL_USERS, make sure no values are added to SERVER_AUDIT_EXCL_USERS.
SERVER_AUDIT_EXCL_USERS	Multiple comma-separated values	None	Exclude activity from the specified users. By default, activity is recorded for all users. SERVER_AUDIT_INCL_USERS and SERVER_AUDIT_EXCL_USERS are mutually exclusive. If you add values to SERVER_AUDIT_EXCL_USERS, make sure no values are added to SERVER_AUDIT_INCL_USERS. The rdsadmin user queries the database every second to check the health of the database. Depending on your other settings, this activity can possibly cause the size of your log file to grow very large, very quickly. If you don't need to record this activity, add the rdsadmin user to the SERVER_AUDIT_EXCL_USERS list. **Note** CONNECT activity is always recorded for all users, even if the user is specified for this option setting.

Option setting	Valid values	Default value	Description
SERVER_AUDIT_LOGGING	ON	ON	Logging is active. The only valid value is ON. Amazon RDS does not support deactivating logging. If you want to deactivate logging, remove the MariaDB Audit Plugin. For more information, see Removing the MariaDB Audit Plugin.
SERVER_AUDIT_QUERY_LOG_LIMIT	0–2147483647	1024	The limit on the length of the query string in a record.

Adding the MariaDB Audit Plugin

The general process for adding the MariaDB Audit Plugin to a DB instance is the following:

- Create a new option group, or copy or modify an existing option group
- Add the option to the option group
- Associate the option group with the DB instance

After you add the MariaDB Audit Plugin, you don't need to restart your DB instance. As soon as the option group is active, auditing begins immediately.

> **Important**
>
> Adding the MariaDB Audit Plugin to a DB instance might cause an outage. We recommend adding the MariaDB Audit Plugin during a maintenance window or during a time of low database workload.

To add the MariaDB Audit Plugin

1. Determine the option group you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step.

Otherwise, create a custom DB option group. Choose **mysql** for **Engine**, and choose **5.7**, **8.0**, or **8.4** for **Major engine version**. For more information, see Creating an option group.

2. Add the **MARIADB_AUDIT_PLUGIN** option to the option group, and configure the option settings. For more information about adding options, see Adding an option to an option group. For more information about each setting, see Audit Plugin option settings.
3. Apply the option group to a new or existing DB instance.

   - For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

Audit log format

Log files are represented as comma-separated variable (CSV) files in UTF-8 format.

> **Tip**
>
> Log file entries are not in sequential order. To order the entries, use the timestamp value. To see the latest events, you might have to review all log files. For more flexibility in sorting and searching the log data, turn on the setting to upload the audit logs to CloudWatch and view them using the CloudWatch interface.
> To view audit data with more types of fields and with output in JSON format, you can also use the Database Activity Streams feature. For more information, see Monitoring Amazon RDS with Database Activity Streams.

The audit log files include the following comma-delimited information in rows, in the specified order:

Field	Description
timestamp	The `YYYYMMDD` followed by the `HH:MI:SS` (24-hour clock) for the logged event.
serverhost	The name of the instance that the event is logged for.

Field	Description
username	The connected user name of the user.
host	The host that the user connected from.
connectionid	The connection ID number for the logged operation.
queryid	The query ID number, which can be used for finding the relational table events and related queries. For `TABLE` events, multiple lines are added.
operation	The recorded action type. Possible values are: `CONNECT`, `QUERY`, `READ`, `WRITE`, `CREATE`, `ALTER`, `RENAME`, and `DROP`.
database	The active database, as set by the `USE` command.
object	For `QUERY` events, this value indicates the query that the database performed. For `TABLE` events, it indicates the table name.
retcode	The return code of the logged operation.
connection_type	The security state of the connection to the server. Possible values are: • `Undefined` • `TCP/IP` • `Socket` • `Named pipe` • `SSL/TLS` • `Shared memory`

Viewing and downloading the MariaDB Audit Plugin log

After you enable the MariaDB Audit Plugin, you access the results in the log files the same way you access any other text-based log files. The audit log files are located at `/rdsdbdata/log/audit/`. For information about viewing the log file in the console, see Viewing and listing database log files. For information about downloading the log file, see Downloading a database log file.

Modifying MariaDB Audit Plugin settings

After you enable the MariaDB Audit Plugin, you can modify the settings. For more information about how to modify option settings, see Modifying an option setting. For more information about each setting, see Audit Plugin option settings.

Removing the MariaDB Audit Plugin

Amazon RDS doesn't support turning off logging in the MariaDB Audit Plugin. However, you can remove the plugin from a DB instance. When you remove the MariaDB Audit Plugin, the DB instance is restarted automatically to stop auditing.

To remove the MariaDB Audit Plugin from a DB instance, do one of the following:

- Remove the MariaDB Audit Plugin option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group
- Modify the DB instance and specify a different option group that doesn't include the plugin. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

MySQL memcached support

Amazon RDS supports using the memcached interface to InnoDB tables that was introduced in MySQL 5.6. The memcached API enables applications to use InnoDB tables in a manner similar to NoSQL key-value data stores.

> **Note**
>
> The memcached interface is no longer available in MySQL 8.4. When you upgrade your DB instances to MySQL 8.4, you must disable memcached in existing option groups.

The memcached interface is a simple, key-based cache. Applications use memcached to insert, manipulate, and retrieve key-value data pairs from the cache. MySQL 5.6 introduced a plugin that implements a daemon service that exposes data from InnoDB tables through the memcached protocol. For more information about the MySQL memcached plugin, see InnoDB integration with memcached.

To enable memcached support for an RDS for MySQL DB instance

1. Determine the security group to use for controlling access to the memcached interface. If the set of applications already using the SQL interface are the same set that will access the memcached interface, you can use the existing VPC security group used by the SQL interface. If a different set of applications will access the memcached interface, define a new VPC or DB security group. For more information about managing security groups, see Controlling access with security groups
2. Create a custom DB option group, selecting MySQL as the engine type and version. For more information about creating an option group, see Creating an option group.
3. Add the MEMCACHED option to the option group. Specify the port that the memcached interface will use, and the security group to use in controlling access to the interface. For more information about adding options, see Adding an option to an option group.
4. Modify the option settings to configure the memcached parameters, if necessary. For more information about how to modify option settings, see Modifying an option setting.
5. Apply the option group to an instance. Amazon RDS enables memcached support for that instance when the option group is applied:

- You enable memcached support for a new instance by specifying the custom option group when you launch the instance. For more information about launching a MySQL instance, see Creating an Amazon RDS DB instance.
- You enable memcached support for an existing instance by specifying the custom option group when you modify the instance. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

6. Specify which columns in your MySQL tables can be accessed through the memcached interface. The memcached plug-in creates a catalog table named containers in a dedicated database named innodb_memcache. You insert a row into the containers table to map an InnoDB table for access through memcached. You specify a column in the InnoDB table that is used to store the memcached key values, and one or more columns that are used to store the data values associated with the key. You also specify a name that a memcached application uses to refer to that set of columns. For details on inserting rows in the containers table, see InnoDB memcached plugin internals. For an example of mapping an InnoDB table and accessing it through memcached, see Writing applications for the InnoDB memcached plugin.
7. If the applications accessing the memcached interface are on different computers or EC2 instances than the applications using the SQL interface, add the connection information for those computers to the VPC security group associated with the MySQL instance. For more information about managing security groups, see Controlling access with security groups.

You turn off the memcached support for an instance by modifying the instance and specifying the default option group for your MySQL version. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

MySQL memcached security considerations

The memcached protocol does not support user authentication. For more information about MySQL memcached security considerations, see Security Considerations for the InnoDB memcached Plugin in the MySQL documentation.

You can take the following actions to help increase the security of the memcached interface:

- Specify a different port than the default of 11211 when adding the MEMCACHED option to the option group.

- Ensure that you associate the memcached interface with a VPC security group that limits access to known, trusted client addresses and EC2 instances. For more information about managing security groups, see Controlling access with security groups.

MySQL memcached connection information

To access the memcached interface, an application must specify both the DNS name of the Amazon RDS instance and the memcached port number. For example, if an instance has a DNS name of my-cache-instance.cg034hpkmmjt.region.rds.amazonaws.com and the memcached interface is using port 11212, the connection information specified in PHP would be:

```
<?php

$cache = new Memcache;
$cache->connect('my-cache-instance.cg034hpkmmjt.region.rds.amazonaws.com',11212);
?>
```

To find the DNS name and memcached port of a MySQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the top right corner of the AWS Management Console, select the region that contains the DB instance.
3. In the navigation pane, choose **Databases**.
4. Choose the MySQL DB instance name to display its details.
5. In the **Connect** section, note the value of the **Endpoint** field. The DNS name is the same as the endpoint. Also, note that the port in the **Connect** section is not used to access the memcached interface.
6. In the **Details** section, note the name listed in the **Option Group** field.
7. In the navigation pane, choose **Option groups**.
8. Choose the name of the option group used by the MySQL DB instance to show the option group details. In the **Options** section, note the value of the **Port** setting for the **MEMCACHED** option.

MySQL memcached option settings

Amazon RDS exposes the MySQL memcached parameters as option settings in the Amazon RDS MEMCACHED option.

MySQL memcached parameters

- DAEMON_MEMCACHED_R_BATCH_SIZE – an integer that specifies how many memcached read operations (get) to perform before doing a COMMIT to start a new transaction. The allowed values are 1 to 4294967295; the default is 1. The option does not take effect until the instance is restarted.
- DAEMON_MEMCACHED_W_BATCH_SIZE – an integer that specifies how many memcached write operations, such as add, set, or incr, to perform before doing a COMMIT to start a new transaction. The allowed values are 1 to 4294967295; the default is 1. The option does not take effect until the instance is restarted.
- INNODB_API_BK_COMMIT_INTERVAL – an integer that specifies how often to auto-commit idle connections that use the InnoDB memcached interface. The allowed values are 1 to 1073741824; the default is 5. The option takes effect immediately, without requiring that you restart the instance.
- INNODB_API_DISABLE_ROWLOCK – a Boolean that disables (1 (true)) or enables (0 (false)) the use of row locks when using the InnoDB memcached interface. The default is 0 (false). The option does not take effect until the instance is restarted.
- INNODB_API_ENABLE_MDL – a Boolean that when set to 0 (false) locks the table used by the InnoDB memcached plugin, so that it cannot be dropped or altered by DDL through the SQL interface. The default is 0 (false). The option does not take effect until the instance is restarted.
- INNODB_API_TRX_LEVEL – an integer that specifies the transaction isolation level for queries processed by the memcached interface. The allowed values are 0 to 3. The default is 0. The option does not take effect until the instance is restarted.

Amazon RDS configures these MySQL memcached parameters, and they cannot be modified: DAEMON_MEMCACHED_LIB_NAME, DAEMON_MEMCACHED_LIB_PATH, and INNODB_API_ENABLE_BINLOG. The parameters that MySQL administrators set by using daemon_memcached_options are available as individual MEMCACHED option settings in Amazon RDS.

MySQL daemon_memcached_options parameters

- BINDING_PROTOCOL – a string that specifies the binding protocol to use. The allowed values are auto, ascii, or binary. The default is auto, which means the server automatically negotiates the protocol with the client. The option does not take effect until the instance is restarted.
- BACKLOG_QUEUE_LIMIT – an integer that specifies how many network connections can be waiting to be processed by memcached. Increasing this limit may reduce errors received by a client that is not able to connect to the memcached instance, but does not improve the performance of the server. The allowed values are 1 to 2048; the default is 1024. The option does not take effect until the instance is restarted.
- CAS_DISABLED – a Boolean that enables (1 (true)) or disables (0 (false)) the use of compare and swap (CAS), which reduces the per-item size by 8 bytes. The default is 0 (false). The option does not take effect until the instance is restarted.
- CHUNK_SIZE – an integer that specifies the minimum chunk size, in bytes, to allocate for the smallest item's key, value, and flags. The allowed values are 1 to 48. The default is 48 and you can significantly improve memory efficiency with a lower value. The option does not take effect until the instance is restarted.
- CHUNK_SIZE_GROWTH_FACTOR – a float that controls the size of new chunks. The size of a new chunk is the size of the previous chunk times CHUNK_SIZE_GROWTH_FACTOR. The allowed values are 1 to 2; the default is 1.25. The option does not take effect until the instance is restarted.
- ERROR_ON_MEMORY_EXHAUSTED – a Boolean that when set to 1 (true) specifies that memcached will return an error rather than evicting items when there is no more memory to store items. If set to 0 (false), memcached will evict items if there is no more memory. The default is 0 (false). The option does not take effect until the instance is restarted.
- MAX_SIMULTANEOUS_CONNECTIONS – an integer that specifies the maximum number of concurrent connections. Setting this value to anything under 10 prevents MySQL from starting. The allowed values are 10 to 1024; the default is 1024. The option does not take effect until the instance is restarted.
- VERBOSITY – a string that specifies the level of information logged in the MySQL error log by the memcached service. The default is v. The option does not take effect until the instance is restarted. The allowed values are:
  - v – Logs errors and warnings while running the main event loop.
  - vv – In addition to the information logged by v, also logs each client command and the response.

- vvv – In addition to the information logged by vv, also logs internal state transitions.

Amazon RDS configures these MySQL DAEMON_MEMCACHED_OPTIONS parameters, they cannot be modified: DAEMON_PROCESS, LARGE_MEMORY_PAGES, MAXIMUM_CORE_FILE_LIMIT, MAX_ITEM_SIZE, LOCK_DOWN_PAGE_MEMORY, MASK, IDFILE, REQUESTS_PER_EVENT, SOCKET, and USER.

Parameters for MySQL

By default, a MySQL DB instance uses a DB parameter group that is specific to a MySQL database. This parameter group contains parameters for the MySQL database engine. For information about working with parameter groups and setting parameters, see Parameter groups for Amazon RDS.

RDS for MySQL parameters are set to the default values of the storage engine that you have selected. For more information about MySQL parameters, see the MySQL documentation. For more information about MySQL storage engines, see Supported storage engines for RDS for MySQL.

You can view the parameters available for a specific RDS for MySQL version using the RDS console or the AWS CLI. For information about viewing the parameters in a MySQL parameter group in the RDS console, see Viewing parameter values for a DB parameter group in Amazon RDS.

Using the AWS CLI, you can view the parameters for an RDS for MySQL version by running the `describe-engine-default-parameters` command. Specify one of the following values for the `--db-parameter-group-family` option:

- `mysql8.4`
- `mysql8.0`
- `mysql5.7`

For example, to view the parameters for RDS for MySQL version 8.0, run the following command.

```
aws rds describe-engine-default-parameters --db-parameter-group-family mysql8.0
```

Your output looks similar to the following.

```
{
    "EngineDefaults": {
        "Parameters": [
            {
                "ParameterName": "activate_all_roles_on_login",
                "ParameterValue": "0",
                "Description": "Automatically set all granted roles as active after the
 user has authenticated successfully.",
                "Source": "engine-default",
                "ApplyType": "dynamic",
                "DataType": "boolean",
```

```
                "AllowedValues": "0,1",
                "IsModifiable": true
            },
            {
                "ParameterName": "allow-suspicious-udfs",
                "Description": "Controls whether user-defined functions that have only
 an xxx symbol for the main function can be loaded",
                "Source": "engine-default",
                "ApplyType": "static",
                "DataType": "boolean",
                "AllowedValues": "0,1",
                "IsModifiable": false
            },
            {
                "ParameterName": "auto_generate_certs",
                "Description": "Controls whether the server autogenerates SSL key and
 certificate files in the data directory, if they do not already exist.",
                "Source": "engine-default",
                "ApplyType": "static",
                "DataType": "boolean",
                "AllowedValues": "0,1",
                "IsModifiable": false
            },
        ...
```

To list only the modifiable parameters for RDS for MySQL version 8.0, run the following command.

For Linux, macOS, or Unix:

```
aws rds describe-engine-default-parameters --db-parameter-group-family mysql8.0 \
   --query 'EngineDefaults.Parameters[?IsModifiable==`true`]'
```

For Windows:

```
aws rds describe-engine-default-parameters --db-parameter-group-family mysql8.0 ^
   --query "EngineDefaults.Parameters[?IsModifiable==`true`]"
```

Common DBA tasks for MySQL DB instances

In the following content, you can find descriptions of the Amazon RDS-specific implementations of some common DBA tasks for DB instances running the MySQL database engine. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. Also, it restricts access to certain system procedures and tables that require advanced privileges.

For information about working with MySQL log files on Amazon RDS, see MySQL database log files.

Understanding predefined users

Amazon RDS automatically creates several predefined users with new RDS for MySQL DB instances. Predefined users and their privileges can't be changed. You can't drop, rename, or modify privileges for these predefined users. Attempting to do so results in an error.

- **rdsadmin** – A user that's created to handle many of the management tasks that the administrator with `superuser` privileges would perform on a standalone MySQL database. This user is used internally by RDS for MySQL for many management tasks.
- **rdsrepladmin** – A user that's used internally by Amazon RDS to support replication activities on RDS for MySQL DB instances and clusters.

For information about other common DBA tasks, see the following topics.

Topics

- Role-based privilege model for RDS for MySQL
- Dynamic privileges for RDS for MySQL
- Ending a session or query for RDS for MySQL
- Skipping the current replication error for RDS for MySQL
- Working with InnoDB tablespaces to improve crash recovery times for RDS for MySQL
- Managing the Global Status History for RDS for MySQL
- Configuring buffer pool size and redo log capacity in MySQL 8.4

Role-based privilege model for RDS for MySQL

Starting with RDS for MySQL version 8.0.36, you can't modify the tables in the `mysql` database directly. In particular, you can't create database users by performing data manipulation language

(DML) operations on the `grant` tables. Instead, you use MySQL account-management statements such as `CREATE USER`, `GRANT`, and `REVOKE` to grant role-based privileges to users. You also can't create other kinds of objects such as stored procedures in the `mysql` database. You can still query the `mysql` tables. If you use binary log replication, changes made directly to the `mysql` tables on the source DB instance aren't replicated to the target cluster.

In some cases, your application might use shortcuts to create users or other objects by inserting into the `mysql` tables. If so, change your application code to use the corresponding statements such as `CREATE USER`.

To export metadata for database users during the migration from an external MySQL database, use one of the following methods:

- Use MySQL Shell's instance dump utility with a filter to exclude users, roles, and grants. The following example shows you the command syntax to use. Make sure that `outputUrl` is empty.

  ```
  mysqlsh user@host -- util.dumpInstance(outputUrl,{excludeSchemas:['mysql'],users:
   true})
  ```

  For more information, see Instance Dump Utility, Schema Dump Utility, and Table Dump Utility in the MySQL Reference Manual.
- Use the `mysqlpump` client utility. This example includes all tables except for tables in the `mysql` system database. It also includes `CREATE USER` and `GRANT` statements to reproduce all MySQL users in the migrated database.

  ```
  mysqlpump --exclude-databases=mysql --users
  ```

  The `mysqlpump` client utility is no longer available with MySQL 8.4. Instead, use `mysqldump`.

To simplify managing permissions for many users or applications, you can use the `CREATE ROLE` statement to create a role that has a set of permissions. Then you can use the `GRANT` and `SET ROLE` statements and the `current_role` function to assign roles to users or applications, switch the current role, and check which roles are in effect. For more information on the role-based permission system in MySQL 8.0, see Using Roles in the MySQL Reference Manual.

> **Important**
>
> We strongly recommend that you do not use the master user directly in your applications. Instead, adhere to the best practice of using a database user created with the minimal privileges required for your application.

Starting with version 8.0.36, RDS for MySQL includes a special role that has all of the following privileges. This role is named `rds_superuser_role`. The primary administrative user for each DB instance already has this role granted. The `rds_superuser_role` role includes the following privileges for all database objects:

- `ALTER`
- `APPLICATION_PASSWORD_ADMIN`
- `ALTER ROUTINE`
- `CREATE`
- `CREATE ROLE`
- `CREATE ROUTINE`
- `CREATE TEMPORARY TABLES`
- `CREATE USER`
- `CREATE VIEW`
- `DELETE`
- `DROP`
- `DROP ROLE`
- `EVENT`
- `EXECUTE`
- `INDEX`
- `INSERT`
- `LOCK TABLES`
- `PROCESS`
- `REFERENCES`
- `RELOAD`
- `REPLICATION CLIENT`

- REPLICATION SLAVE
- ROLE_ADMIN
- SET_USER_ID
- SELECT
- SHOW DATABASES
- SHOW VIEW
- TRIGGER
- UPDATE
- XA_RECOVER_ADMIN

The role definition also includes WITH GRANT OPTION so that an administrative user can grant that role to other users. In particular, the administrator must grant any privileges needed to perform binary log replication with the MySQL cluster as the target.

Tip

To see the full details of the permissions, use the following statement.

```
SHOW GRANTS FOR rds_superuser_role@'%';
```

When you grant access by using roles in RDS for MySQL version 8.0.36 and higher, you also activate the role by using the SET ROLE *role_name* or SET ROLE ALL statement. The following example shows how. Substitute the appropriate role name for CUSTOM_ROLE.

```
# Grant role to user
mysql> GRANT CUSTOM_ROLE TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the CUSTOM_ROLE role has not
 been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+--------------------------+
| CURRENT_ROLE()           |
+--------------------------+
| `rds_superuser_role`@`%` |
+--------------------------+
```

```
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-------------------------------------------------+
| CURRENT_ROLE()                                  |
+-------------------------------------------------+
| `CUSTOM_ROLE`@`%`,`rds_superuser_role`@`%` |
+-------------------------------------------------+
```

Dynamic privileges for RDS for MySQL

Dynamic privileges are MySQL privileges that you can explicitly grant by using the GRANT statement. Depending on your version of RDS for MySQL, RDS allows you to grant only specific dynamic privileges. RDS disallows some of these privileges because they can interfere with the specific database operations, such as replication and backup.

The following table shows which of these privileges you can grant for different MySQL versions. If you are upgrading from a MySQL version lower than 8.0.36 to version 8.0.36 or higher, you might have to update your application code if granting a particular privilege is no longer allowed.

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
ALLOW_NONEXISTENT_DEFINER	Not available	Not available	Disallowed
APPLICATION_PASSWORD_ADMIN	Allowed	Allowed	Allowed
AUDIT_ABORT_EXEMPT	Allowed	Disallowed	Disallowed

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
AUDIT_ADMIN	Disallowed	Disallowed	Disallowed
AUTHENTICATION_POLICY_ADMIN	Allowed	Disallowed	Disallowed
BACKUP_ADMIN	Allowed	Disallowed	Disallowed
BINLOG_ADMIN	Allowed	Disallowed	Disallowed
BINLOG_ENCRYPTION_ADMIN	Disallowed	Disallowed	Disallowed
CLONE_ADMIN	Disallowed	Disallowed	Disallowed
CONNECTION_ADMIN	Allowed	Disallowed	Disallowed
ENCRYPTION_KEY_ADMIN	Disallowed	Disallowed	Disallowed
FIREWALL_ADMIN	Disallowed	Disallowed	Disallowed
FIREWALL_EXEMPT	Allowed	Disallowed	Disallowed
FIREWALL_USER	Disallowed	Disallowed	Disallowed
FLUSH_OPTIMIZER_COSTS	Allowed	Allowed	Allowed
FLUSH_PRIVILEGES	Not available	Not available	Allowed
FLUSH_STATUS	Allowed	Allowed	Allowed
FLUSH_TABLES	Allowed	Allowed	Allowed
FLUSH_USER_RESOURCES	Allowed	Allowed	Allowed

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
GROUP_REPLICATION_ADMIN	Disallowed	Disallowed	Disallowed
GROUP_REPLICATION_STREAM	Disallowed	Disallowed	Disallowed
INNODB_REDO_LOG_ARCHIVE	Disallowed	Disallowed	Disallowed
INNODB_REDO_LOG_ENABLE	Disallowed	Disallowed	Disallowed
MASKING_DICTIONARIES_ADMIN	Disallowed	Disallowed	Disallowed
NDB_STORED_USER	Disallowed	Disallowed	Disallowed
OPTIMIZE_LOCAL_TABLE	Not available	Not available	Disallowed
PASSWORDLESS_USER_ADMIN	Disallowed	Disallowed	Disallowed
PERSIST_RO_VARIABLES_ADMIN	Disallowed	Disallowed	Disallowed
REPLICATION_APPLIER	Allowed	Disallowed	Disallowed
REPLICATION_SLAVE_ADMIN	Disallowed	Disallowed	Disallowed
RESOURCE_GROUP_ADMIN	Allowed	Disallowed	Disallowed

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
RESOURCE_GROUP_USER	Allowed	Disallowed	Disallowed
ROLE_ADMIN	Allowed	Allowed	Allowed
SENSITIVE_VARIABLES_OBSERVER	Allowed	Allowed	Allowed
SERVICE_CONNECTION_ADMIN	Allowed	Disallowed	Disallowed
SESSION_VARIABLES_ADMIN	Allowed	Allowed	Allowed
SET_ANY_DEFINER	Not available	Not available	Allowed
SET_USER_ID	Allowed	Allowed	Not available
SHOW_ROUTINE	Allowed	Allowed	Allowed
SKIP_QUERY_REWRITE	Disallowed	Disallowed	Disallowed
SYSTEM_USER	Disallowed	Disallowed	Disallowed
SYSTEM_VARIABLES_ADMIN	Disallowed	Disallowed	Disallowed
TABLE_ENCRYPTION_ADMIN	Disallowed	Disallowed	Disallowed
TELEMETRY_LOG_ADMIN	Allowed	Disallowed	Disallowed
TP_CONNECTION_ADMIN	Disallowed	Disallowed	Disallowed

Privilege	MySQL 8.0.35 and lower	MySQL 8.0.36 and higher minor versions	MySQL 8.4.3 and higher
TRANSACTION_GTID_TAG	Not available	Not available	Disallowed
VERSION_TOKEN_ADMIN	Disallowed	Disallowed	Disallowed
XA_RECOVER_ADMIN	Allowed	Allowed	Allowed

Ending a session or query for RDS for MySQL

You can end user sessions or queries on DB instances by using the `rds_kill` and `rds_kill_query` commands. First connect to your MySQL DB instance, then issue the appropriate command as shown following. For more information, see Connecting to your MySQL DB instance.

```
CALL mysql.rds_kill(thread-ID)
CALL mysql.rds_kill_query(thread-ID)
```

For example, to end the session that is running on thread 99, you would type the following:

```
CALL mysql.rds_kill(99);
```

To end the query that is running on thread 99, you would type the following:

```
CALL mysql.rds_kill_query(99);
```

Skipping the current replication error for RDS for MySQL

You can skip an error on your read replica if the error is causing your read replica to stop responding and the error doesn't affect the integrity of your data.

> **Note**
>
> First verify that the error in question can be safely skipped. In a MySQL utility, connect to the read replica and run the following MySQL command.

```
SHOW REPLICA STATUS\G
```

For information about the values returned, see the MySQL documentation.
Previous versions of and MySQL used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MySQL version before 8.0.23, then use `SHOW SLAVE STATUS`.

You can skip an error on your read replica in the following ways.

Topics

- Calling the mysql.rds_skip_repl_error procedure
- Setting the slave_skip_errors parameter

Calling the mysql.rds_skip_repl_error procedure

Amazon RDS provides a stored procedure that you can call to skip an error on your read replicas. First connect to your read replica, then issue the appropriate commands as shown following. For more information, see Connecting to your MySQL DB instance.

To skip the error, issue the following command.

```
CALL mysql.rds_skip_repl_error;
```

This command has no effect if you run it on the source DB instance, or on a read replica that hasn't encountered a replication error.

For more information, such as the versions of MySQL that support `mysql.rds_skip_repl_error`, see mysql.rds_skip_repl_error.

Important

If you attempt to call `mysql.rds_skip_repl_error` and encounter the following error: `ERROR 1305 (42000): PROCEDURE mysql.rds_skip_repl_error does not exist`, then upgrade your MySQL DB instance to the latest minor version or one of the minimum minor versions listed in mysql.rds_skip_repl_error.

Setting the slave_skip_errors parameter

To skip one or more errors, you can set the `slave_skip_errors` static parameter on the read replica. You can set this parameter to skip one or more specific replication error codes. Currently, you can set this parameter only for RDS for MySQL 5.7 DB instances. After you change the setting for this parameter, make sure to reboot your DB instance for the new setting to take effect. For information about setting this parameter, see the MySQL documentation.

We recommend setting this parameter in a separate DB parameter group. You can associate this DB parameter group only with the read replicas that need to skip errors. Following this best practice reduces the potential impact on other DB instances and read replicas.

> **Important**
>
> Setting a nondefault value for this parameter can lead to replication inconsistency. Only set this parameter to a nondefault value if you have exhausted other options to resolve the problem and you are sure of the potential impact on your read replica's data.

Working with InnoDB tablespaces to improve crash recovery times for RDS for MySQL

Every table in MySQL consists of a table definition, data, and indexes. The MySQL storage engine InnoDB stores table data and indexes in a *tablespace*. InnoDB creates a global shared tablespace that contains a data dictionary and other relevant metadata, and it can contain table data and indexes. InnoDB can also create separate tablespaces for each table and partition. These separate tablespaces are stored in files with a .ibd extension and the header of each tablespace contains a number that uniquely identifies it.

Amazon RDS provides a parameter in a MySQL parameter group called `innodb_file_per_table`. This parameters controls whether InnoDB adds new table data and indexes to the shared tablespace (by setting the parameter value to 0) or to individual tablespaces (by setting the parameter value to 1). Amazon RDS sets the default value for `innodb_file_per_table` parameter to 1, which allows you to drop individual InnoDB tables and reclaim storage used by those tables for the DB instance. In most use cases, setting the `innodb_file_per_table` parameter to 1 is the recommended setting.

You should set the `innodb_file_per_table` parameter to 0 when you have a large number of tables, such as over 1000 tables when you use standard (magnetic) or general purpose SSD storage

or over 10,000 tables when you use Provisioned IOPS storage. When you set this parameter to 0, individual tablespaces are not created and this can improve the time it takes for database crash recovery.

MySQL processes each metadata file, which includes tablespaces, during the crash recovery cycle. The time it takes MySQL to process the metadata information in the shared tablespace is negligible compared to the time it takes to process thousands of tablespace files when there are multiple tablespaces. Because the tablespace number is stored within the header of each file, the aggregate time to read all the tablespace files can take up to several hours. For example, a million InnoDB tablespaces on standard storage can take from five to eight hours to process during a crash recovery cycle. In some cases, InnoDB can determine that it needs additional cleanup after a crash recovery cycle so it will begin another crash recovery cycle, which will extend the recovery time. Keep in mind that a crash recovery cycle also entails rolling-back transactions, fixing broken pages, and other operations in addition to the processing of tablespace information.

Since the `innodb_file_per_table` parameter resides in a parameter group, you can change the parameter value by editing the parameter group used by your DB instance without having to reboot the DB instance. After the setting is changed, for example, from 1 (create individual tables) to 0 (use shared tablespace), new InnoDB tables will be added to the shared tablespace while existing tables continue to have individual tablespaces. To move an InnoDB table to the shared tablespace, you must use the `ALTER TABLE` command.

Migrating multiple tablespaces to the shared tablespace

You can move an InnoDB table's metadata from its own tablespace to the shared tablespace, which will rebuild the table metadata according to the `innodb_file_per_table` parameter setting. First connect to your MySQL DB instance, then issue the appropriate commands as shown following. For more information, see Connecting to your MySQL DB instance.

```
ALTER TABLE table_name ENGINE = InnoDB, ALGORITHM=COPY;
```

For example, the following query returns an `ALTER TABLE` statement for every InnoDB table that is not in the shared tablespace.

For MySQL 5.7 DB instances:

```
SELECT CONCAT('ALTER TABLE `',
REPLACE(LEFT(NAME , INSTR((NAME), '/') - 1), '`', '``'), '`.`',
REPLACE(SUBSTR(NAME FROM INSTR(NAME, '/') + 1), '`', '``'), '` ENGINE=InnoDB,
 ALGORITHM=COPY;') AS Query
```

```
FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES
WHERE SPACE <> 0 AND LEFT(NAME, INSTR((NAME), '/') - 1) NOT IN ('mysql','');
```

For MySQL 8.4 and 8.0 DB instances:

```
SELECT CONCAT('ALTER TABLE `',
REPLACE(LEFT(NAME , INSTR((NAME), '/') - 1), '`', '``'), '`.`',
REPLACE(SUBSTR(NAME FROM INSTR(NAME, '/') + 1), '`', '``'), '` ENGINE=InnoDB,
 ALGORITHM=COPY;') AS Query
FROM INFORMATION_SCHEMA.INNODB_TABLES
WHERE SPACE <> 0 AND LEFT(NAME, INSTR((NAME), '/') - 1) NOT IN ('mysql','');
```

Rebuilding a MySQL table to move the table's metadata to the shared tablespace requires additional storage space temporarily to rebuild the table, so the DB instance must have storage space available. During rebuilding, the table is locked and inaccessible to queries. For small tables or tables not frequently accessed, this might not be an issue. For large tables or tables frequently accessed in a heavily concurrent environment, you can rebuild tables on a read replica.

You can create a read replica and migrate table metadata to the shared tablespace on the read replica. While the ALTER TABLE statement blocks access on the read replica, the source DB instance is not affected. The source DB instance will continue to generate its binary logs while the read replica lags during the table rebuilding process. Because the rebuilding requires additional storage space and the replay log file can become large, you should create a read replica with storage allocated that is larger than the source DB instance.

To create a read replica and rebuild InnoDB tables to use the shared tablespace, take the following steps:

1. Make sure that backup retention is enabled on the source DB instance so that binary logging is enabled.
2. Use the AWS Management Console or AWS CLI to create a read replica for the source DB instance. Because the creation of a read replica involves many of the same processes as crash recovery, the creation process can take some time if there is a large number of InnoDB tablespaces. Allocate more storage space on the read replica than is currently used on the source DB instance.
3. When the read replica has been created, create a parameter group with the parameter settings `read_only = 0` and `innodb_file_per_table = 0`. Then associate the parameter group with the read replica.
4. Issue the following SQL statement for all tables that you want migrated on the replica:

```
ALTER TABLE name ENGINE = InnoDB
```

5. When all of your `ALTER TABLE` statements have completed on the read replica, verify that the read replica is connected to the source DB instance and that the two instances are in sync.
6. Use the console or CLI to promote the read replica to be the instance. Make sure that the parameter group used for the new standalone DB instance has the `innodb_file_per_table` parameter set to 0. Change the name of the new standalone DB instance, and point any applications to the new standalone DB instance.

Managing the Global Status History for RDS for MySQL

> **Tip**
>
> To analyze database performance, you can also use Performance Insights on Amazon RDS. For more information, see Monitoring DB load with Performance Insights on Amazon RDS.

MySQL maintains many status variables that provide information about its operation. Their value can help you detect locking or memory issues on a DB instance. The values of these status variables are cumulative since last time the DB instance was started. You can reset most status variables to 0 by using the `FLUSH STATUS` command.

To allow for monitoring of these values over time, Amazon RDS provides a set of procedures that will snapshot the values of these status variables over time and write them to a table, along with any changes since the last snapshot. This infrastructure, called Global Status History (GoSH), is installed on all MySQL DB instances starting with versions 5.5.23. GoSH is disabled by default.

To enable GoSH, you first enable the event scheduler from a DB parameter group by setting the parameter `event_scheduler` to `ON`. For MySQL DB instances running MySQL 5.7, also set the parameter `show_compatibility_56` to 1. For information about creating and modifying a DB parameter group, see Parameter groups for Amazon RDS. For information about the side effects of enabling this parameter, see show_compatibility_56 in the *MySQL 5.7 Reference Manual*.

You can then use the procedures in the following table to enable and configure GoSH. First connect to your MySQL DB instance, then issue the appropriate commands as shown following. For more information, see Connecting to your MySQL DB instance. For each procedure, run the following command and replace *procedure-name*:

```
CALL procedure-name;
```

The following table lists all of the procedures that you can use for *procedure-name* in the previous command.

Procedure	Description
mysql.rds_enable_gsh_collector	Enables GoSH to take default snapshots at intervals specified by rds_set_gsh_collector .
mysql.rds_set_gsh_collector	Specifies the interval, in minutes, between snapshots. Default value is 5.
mysql.rds_disable_gsh_collector	Disables snapshots.
mysql.rds_collect_global_status_history	Takes a snapshot on demand.
mysql.rds_enable_gsh_rotation	Enables rotation of the contents of the mysql.rds_global_status_history table to mysql.rds_global_status_history_old at intervals specified by rds_set_gsh_rotation .
mysql.rds_set_gsh_rotation	Specifies the interval, in days, between table rotations. Default value is 7.
mysql.rds_disable_gsh_rotation	Disables table rotation.
mysql.rds_rotate_global_status_history	Rotates the contents of the mysql.rds_global_status_history table to mysql.rds_global_status_history_old on demand.

When GoSH is running, you can query the tables that it writes to. For example, to query the hit ratio of the Innodb buffer pool, you would issue the following query:

```
select a.collection_end, a.collection_start, (( a.variable_Delta-b.variable_delta)/
a.variable_delta)*100 as "HitRatio"
    from mysql.rds_global_status_history as a join mysql.rds_global_status_history as b
 on a.collection_end = b.collection_end
    where a. variable_name = 'Innodb_buffer_pool_read_requests' and b.variable_name =
 'Innodb_buffer_pool_reads'
```

Configuring buffer pool size and redo log capacity in MySQL 8.4

In MySQL 8.4, Amazon RDS enables the `innodb_dedicated_server` parameter by default. With the `innodb_dedicated_server` parameter, the database engine calculates the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters. For information about how these parameters are calculated, see Configuring InnoDB Buffer Pool Size and Redo Log in the MySQL documentation.

With `innodb_dedicated_server` enabled, the `innodb_buffer_pool_size` parameter is calculated based on the DB instance class memory. The following table shows the detected server memory and the corresponding buffer pool size.

Detected server memory	Buffer pool size
< 1 GB	Default value of 128 MB
1 GB to 4 GB	*Detected server memory* * 0.5
> 4 GB	*Detected server memory* * 0.75

The `innodb_redo_log_capacity` parameter automatically scales with the instance class to (number of vCPUs / 2) GB up to a maximum of 16 GB. Larger instance classes have a larger redo log capacity, which can improve performance and resilience for write-intensive workloads.

Before upgrading from MySQL 8.0 to MySQL 8.4, be sure to increase your storage space to accommodate a potential increase in the size of the redo logs that might occur after the upgrade completes. For more information, see Increasing DB instance storage capacity.

If you don't want the `innodb_dedicated_server` parameter to calculate the values for the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters, you can override these values by setting specific values for them in a custom parameter group.

Alternatively, you can disable the `innodb_dedicated_server` parameter and set values for the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters in a custom parameter group. For more information, see Default and custom parameter groups.

If you disable the `innodb_dedicated_server` parameter by setting it to `0` and don't set values for the `innodb_buffer_pool_size` and `innodb_redo_log_capacity` parameters, then Amazon RDS sets the latter two parameters to 128 MB and 100 MB, respectively. These defaults result in poor performance on larger instance classes.

Local time zone for MySQL DB instances

By default, the time zone for a MySQL DB instance is Universal Time Coordinated (UTC). You can set the time zone for your DB instance to the local time zone for your application instead.

To set the local time zone for a DB instance, set the `time_zone` parameter in the parameter group for your DB instance to one of the supported values listed later in this section. When you set the `time_zone` parameter for a parameter group, all DB instances and read replicas that are using that parameter group change to use the new local time zone. For information on setting parameters in a parameter group, see Parameter groups for Amazon RDS.

After you set the local time zone, all new connections to the database reflect the change. If you have any open connections to your database when you change the local time zone, you won't see the local time zone update until after you close the connection and open a new connection.

You can set a different local time zone for a DB instance and one or more of its read replicas. To do this, use a different parameter group for the DB instance and the replica or replicas and set the `time_zone` parameter in each parameter group to a different local time zone.

If you are replicating across AWS Regions, then the source DB instance and the read replica use different parameter groups (parameter groups are unique to an AWS Region). To use the same local time zone for each instance, you must set the `time_zone` parameter in the instance's and read replica's parameter groups.

When you restore a DB instance from a DB snapshot, the local time zone is set to UTC. You can update the time zone to your local time zone after the restore is complete. If you restore a DB instance to a point in time, then the local time zone for the restored DB instance is the time zone setting from the parameter group of the restored DB instance.

The Internet Assigned Numbers Authority (IANA) publishes new time zones at https://www.iana.org/time-zones several times a year. Every time RDS releases a new minor maintenance release of MySQL, it ships with the latest time zone data at the time of the release. When you use the latest RDS for MySQL versions, you have recent time zone data from RDS. To ensure that your DB instance has recent time zone data, we recommend upgrading to a higher DB engine version. Alternatively, you can modify the time zone tables in MariaDB DB instances manually. To do so, you can use SQL commands or run the mysql_tzinfo_to_sql tool in a SQL client. After updating the time zone data manually, reboot your DB instance so that the changes take effect. RDS doesn't modify or reset the time zone data of running DB instances. New time zone data is installed only when you perform a database engine version upgrade.

You can set your local time zone to one of the following values.

Zone	Time zone
Africa	Africa/Cairo, Africa/Casablanca, Africa/Harare, Africa/Monrovia, Africa/Nairobi, Africa/Tripoli, Africa/Windhoek
America	America/Araguaina, America/Asuncion, America/Bogota, America/Buenos_Aires, America/Caracas, America/Chihuahua, America/Cuiaba, America/Denver, America/Fortaleza, America/Guatemala, America/Halifax, America/Manaus, America/Matamoros, America/Monterrey, America/Montevideo, America/Phoenix, America/Santiago, America/Tijuana
Asia	Asia/Amman, Asia/Ashgabat, Asia/Baghdad, Asia/Baku, Asia/Bangkok, Asia/Beirut, Asia/Calcutta, Asia/Damascus, Asia/Dhaka, Asia/Irkutsk, Asia/Jerusalem, Asia/Kabul, Asia/Karachi, Asia/Kathmandu, Asia/Krasnoyarsk, Asia/Magadan, Asia/Muscat, Asia/Novosibirsk, Asia/Riyadh, Asia/Seoul, Asia/Shanghai, Asia/Singapore, Asia/Taipei, Asia/Tehran, Asia/Tokyo, Asia/Ulaanbaatar, Asia/Vladivostok, Asia/Yakutsk, Asia/Yerevan
Atlantic	Atlantic/Azores
Australia	Australia/Adelaide, Australia/Brisbane, Australia/Darwin, Australia/Hobart, Australia/Perth, Australia/Sydney
Brazil	Brazil/DeNoronha, Brazil/East
Canada	Canada/Newfoundland, Canada/Saskatchewan, Canda/Yukon
Europe	Europe/Amsterdam, Europe/Athens, Europe/Dublin, Europe/Helsinki, Europe/Istanbul, Europe/Kaliningrad Europe/Moscow, Europe/Paris, Europe/Prague, Europe/Sarajevo
Pacific	Pacific/Auckland, Pacific/Fiji, Pacific/Guam, Pacific/Honolulu, Pacific/Samoa
US	US/Alaska, US/Central, US/East-Indiana, US/Eastern, US/Pacific
UTC	UTC

Known issues and limitations for Amazon RDS for MySQL

Known issues and limitations for working with Amazon RDS for MySQL are as follows.

Topics

- InnoDB reserved word
- Storage-full behavior for Amazon RDS for MySQL
- Inconsistent InnoDB buffer pool size
- Index merge optimization returns incorrect results
- MySQL parameter exceptions for Amazon RDS DB instances
- MySQL file size limits in Amazon RDS
- MySQL Keyring Plugin not supported
- Custom ports
- MySQL stored procedure limitations
- GTID-based replication with an external source instance
- MySQL default authentication plugin
- Overriding innodb_buffer_pool_size
- Upgrading from MySQL 5.7 to MySQL 8.4
- InnoDB page compression

InnoDB reserved word

`InnoDB` is a reserved word for RDS for MySQL. You can't use this name for a MySQL database.

Storage-full behavior for Amazon RDS for MySQL

When storage becomes full for a MySQL DB instance, there can be metadata inconsistencies, dictionary mismatches, and orphan tables. To prevent these issues, Amazon RDS automatically stops a DB instance that reaches the `storage-full` state.

A MySQL DB instance reaches the `storage-full` state in the following cases:

- The DB instance has less than 20,000 MiB of storage, and available storage reaches 200 MiB or less.

- The DB instance has more than 102,400 MiB of storage, and available storage reaches 1024 MiB or less.
- The DB instance has between 20,000 MiB and 102,400 MiB of storage, and has less than 1% of storage available.

After Amazon RDS stops a DB instance automatically because it reached the `storage-full` state, you can still modify it. To restart the DB instance, complete at least one of the following:

- Modify the DB instance to enable storage autoscaling.

  For more information about storage autoscaling, see Managing capacity automatically with Amazon RDS storage autoscaling.
- Modify the DB instance to increase its storage capacity.

  For more information about increasing storage capacity, see Increasing DB instance storage capacity.

After you make one of these changes, the DB instance is restarted automatically. For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

Inconsistent InnoDB buffer pool size

For MySQL 5.7, there is currently a bug in the way that the InnoDB buffer pool size is managed. MySQL 5.7 might adjust the value of the `innodb_buffer_pool_size` parameter to a large value that can result in the InnoDB buffer pool growing too large and using up too much memory. This effect can cause the MySQL database engine to stop running or can prevent it from starting. This issue is more common for DB instance classes that have less memory available.

To resolve this issue, set the value of the `innodb_buffer_pool_size` parameter to a multiple of the product of the `innodb_buffer_pool_instances` parameter value and the `innodb_buffer_pool_chunk_size` parameter value. For example, you might set the `innodb_buffer_pool_size` parameter value to a multiple of eight times the product of the `innodb_buffer_pool_instances` and `innodb_buffer_pool_chunk_size` parameter values, as shown in the following example.

```
innodb_buffer_pool_chunk_size = 536870912
innodb_buffer_pool_instances = 4
innodb_buffer_pool_size = (536870912 * 4) * 8 = 17179869184
```

For details on this MySQL 5.7 bug, see https://bugs.mysql.com/bug.php?id=79379 in the MySQL documentation.

Index merge optimization returns incorrect results

Queries that use index merge optimization might return incorrect results because of a bug in the MySQL query optimizer that was introduced in MySQL 5.5.37. When you issue a query against a table with multiple indexes, the optimizer scans ranges of rows based on the multiple indexes, but does not merge the results together correctly. For more information on the query optimizer bug, see http://bugs.mysql.com/bug.php?id=72745 and http://bugs.mysql.com/bug.php?id=68194 in the MySQL bug database.

For example, consider a query on a table with two indexes where the search arguments reference the indexed columns.

```
SELECT * FROM table1
WHERE indexed_col1 = 'value1' AND indexed_col2 = 'value2';
```

In this case, the search engine will search both indexes. However, because of the bug, the merged results are incorrect.

To resolve this issue, you can do one of the following:

- Set the `optimizer_switch` parameter to `index_merge=off` in the DB parameter group for your MySQL DB instance. For information on setting DB parameter group parameters, see Parameter groups for Amazon RDS.
- Upgrade your MySQL DB instance to MySQL version 5.7 or 8.0. For more information, see Upgrades of the RDS for MySQL DB engine.
- If you cannot upgrade your instance or change the `optimizer_switch` parameter, you can work around the bug by explicitly identifying an index for the query, for example:

  ```
  SELECT * FROM table1
  USE INDEX covering_index
  WHERE indexed_col1 = 'value1' AND indexed_col2 = 'value2';
  ```

For more information, see Index merge optimization in the MySQL documentation.

MySQL parameter exceptions for Amazon RDS DB instances

Some MySQL parameters require special considerations when used with an Amazon RDS DB instance.

lower_case_table_names

Because Amazon RDS uses a case-sensitive file system, setting the value of the `lower_case_table_names` server parameter to 2 (names stored as given but compared in lowercase) is not supported. The following are the supported values for Amazon RDS for MySQL DB instances:

- 0 (names stored as given and comparisons are case-sensitive) is supported for all RDS for MySQL versions.
- 1 (names stored in lowercase and comparisons are not case-sensitive) is supported for RDS for MySQL version 5.7, version 8.0.28 and higher 8.0 versions, and version 8.4.

Set the `lower_case_table_names` parameter in a custom DB parameter group before creating a DB instance. Then, specify the custom DB parameter group when you create the DB instance.

When a parameter group is associated with a MySQL DB instance with a version lower than 8.0, we recommend that you avoid changing the `lower_case_table_names` parameter in the parameter group. Changing it could cause inconsistencies with point-in-time recovery backups and read replica DB instances.

When a parameter group is associated with a version 8.0 or 8.4 MySQL DB instance, you can't modify the `lower_case_table_names` parameter in the parameter group.

Read replicas should always use the same `lower_case_table_names` parameter value as the source DB instance.

long_query_time

You can set the `long_query_time` parameter to a floating point value so that you can log slow queries to the MySQL slow query log with microsecond resolution. You can set a value such as 0.1 seconds, which would be 100 milliseconds, to help when debugging slow transactions that take less than one second.

MySQL file size limits in Amazon RDS

For MySQL versions 8.0 and higher DB instances, the maximum file size is 16 TiB. When using file-per-table tablespaces, the maximum file size limits the size of an InnoDB table to 16 TiB. InnoDB file-per-table tablespaces (with tables each in their own tablespace) is set by default for MySQL DB instances. For more information, see InnoDB Limits in the MySQL documentation.

> **Note**
>
> Some existing DB instances have a lower limit. For example, MySQL DB instances created before April 2014 have a file and table size limit of 2 TB. This 2 TB file size limit also applies to DB instances or read replicas created from DB snapshots taken before April 2014, regardless of when the DB instance was created.

There are advantages and disadvantages to using InnoDB file-per-table tablespaces, depending on your application. To determine the best approach for your application, see File-per-table tablespaces in the MySQL documentation.

We don't recommend allowing tables to grow to the maximum file size. In general, a better practice is to partition data into smaller tables, which can improve performance and recovery times.

One option that you can use for breaking up a large table into smaller tables is partitioning. Partitioning distributes portions of your large table into separate files based on rules that you specify. For example, if you store transactions by date, you can create partitioning rules that distribute older transactions into separate files using partitioning. Then periodically, you can archive the historical transaction data that doesn't need to be readily available to your application. For more information, see Partitioning in the MySQL documentation.

Because there is no single system table or view that provides the size of all the tables and the InnoDB system tablespace, you must query multiple tables to determine the size of the tablespaces.

To determine the size of the InnoDB system tablespace and the data dictionary tablespace

- Use the following SQL command to determine if any of your tablespaces are too large and are candidates for partitioning.

> **Note**
>
> The data dictionary tablespace is specific to MySQL 8.0 and higher versions.

```
select FILE_NAME,TABLESPACE_NAME, ROUND(((TOTAL_EXTENTS*EXTENT_SIZE)
/1024/1024/1024), 2)  as "File Size (GB)" from information_schema.FILES
where tablespace_name in ('mysql','innodb_system');
```

To determine the size of InnoDB user tables outside of the InnoDB system tablespace (for MySQL 5.7 versions)

- Use the following SQL command to determine if any of your tables are too large and are candidates for partitioning.

```
SELECT SPACE,NAME,ROUND((ALLOCATED_SIZE/1024/1024/1024), 2)
as "Tablespace Size (GB)"
FROM information_schema.INNODB_SYS_TABLESPACES ORDER BY 3 DESC;
```

To determine the size of InnoDB user tables outside of the InnoDB system tablespace (for MySQL 8.0 and higher versions)

- Use the following SQL command to determine if any of your tables are too large and are candidates for partitioning.

```
SELECT SPACE,NAME,ROUND((ALLOCATED_SIZE/1024/1024/1024), 2)
as "Tablespace Size (GB)"
FROM information_schema.INNODB_TABLESPACES ORDER BY 3 DESC;
```

To determine the size of non-InnoDB user tables

- Use the following SQL command to determine if any of your non-InnoDB user tables are too large.

```
SELECT TABLE_SCHEMA, TABLE_NAME, round(((DATA_LENGTH + INDEX_LENGTH+DATA_FREE)
/ 1024 / 1024/ 1024), 2) As "Approximate size (GB)" FROM information_schema.TABLES
```

```
WHERE TABLE_SCHEMA NOT IN ('mysql', 'information_schema', 'performance_schema')
and ENGINE<>'InnoDB';
```

To enable InnoDB file-per-table tablespaces

- Set the *innodb_file_per_table* parameter to 1 in the parameter group for the DB instance.

To disable InnoDB file-per-table tablespaces

- Set the *innodb_file_per_table* parameter to `0` in the parameter group for the DB instance.

For information on updating a parameter group, see Parameter groups for Amazon RDS.

When you have enabled or disabled InnoDB file-per-table tablespaces, you can issue an `ALTER TABLE` command to move a table from the global tablespace to its own tablespace, or from its own tablespace to the global tablespace as shown in the following example:

```
ALTER TABLE table_name TABLESPACE=innodb_file_per_table;
```

MySQL Keyring Plugin not supported

Currently, Amazon RDS for MySQL doesn't support the MySQL `keyring_aws` Amazon Web Services Keyring Plugin.

Custom ports

Amazon RDS blocks connections to custom port 33060 for the MySQL engine. Choose a different port for your MySQL engine.

MySQL stored procedure limitations

The mysql.rds_kill and mysql.rds_kill_query stored procedures can't terminate sessions or queries owned by MySQL users with usernames longer than 16 characters on the following RDS for MySQL versions:

- 8.0.32 and lower 8 versions
- 5.7.41 and lower 5.7 versions

GTID-based replication with an external source instance

Amazon RDS supports replication based on global transaction identifiers (GTIDs) from an external MySQL instance into an Amazon RDS for MySQL DB instance that requires setting GTID_PURGED during configuration. However, only RDS for MySQL 8.0.37 and higher versions support this functionality.

MySQL default authentication plugin

RDS for MySQL version 8.0.34 and higher 8.0 versions use the `mysql_native_password` plugin. You can't change the `default_authentication_plugin` setting.

RDS for MySQL version 8.4 and higher versions use the `caching_sha2_password` plugin as the default authentication plugin. You can change the default authentication plugin for MySQL 8.4. The `mysql_native_password` plugin still works with MySQL 8.4, but support of this plugin ends with MySQL 8.4. To change the default authentication plugin, create a custom parameter group and modify the value of the `authentication_policy` parameter. For more information, see the section called “Default and custom parameter groups”.

Overriding innodb_buffer_pool_size

With micro or small DB instance classes, the default value for the `innodb_buffer_pool_size` parameter might differ from the value returned by running the following command:

```
mysql> SELECT @@innodb_buffer_pool_size;
```

This difference can occur when Amazon RDS needs to override the default value as part of managing the DB instance classes. If necessary, you can override the default value and set it to a value that your DB instance class supports. To determine a valid value, add the memory usage and the total memory available on your DB instance. For more information, see Amazon RDS instance types.

If your DB instance has only 4 GB of memory, you can't set `innodb_buffer_pool_size` to 8 GB but you might be able to set it to 3 GB, depending on how much memory you allocated for other parameters.

If the value that you input is too large, Amazon RDS lowers the value to the following limits:

- Micro DB instance classes: 256 MB

- db.t4g.micro DB instance classes: 128 MB

Upgrading from MySQL 5.7 to MySQL 8.4

You can't upgrade directly from MySQL 5.7 to MySQL 8.4. You must first upgrade from MySQL 5.7 to MySQL 8.0, and then upgrade from MySQL 8.0 to MySQL 8.4. For more information, see Major version upgrades for RDS for MySQL.

InnoDB page compression

InnoDB page compression doesn't work with Amazon RDS DB instances that have a file system block size of 16k because the file system block size must be smaller than the InnoDB page size. Starting in February 2024, all newly created DB instances have a file system block size of 16k, which increases throughput and decreases IOPS consumption during page flushes.

RDS for MySQL stored procedure reference

These topics describe system stored procedures that are available for Amazon RDS instances running the MySQL DB engine. The master user must run these procedures.

Topics

- Collecting and maintaining the Global Status History
- Configuring, starting, and stopping binary log (binlog) replication
- Ending a session or query
- Managing active-active clusters
- Managing multi-source replication
- Replicating transactions using GTIDs
- Rotating the query logs
- Setting and showing binary log configuration
- Warming the InnoDB cache

Collecting and maintaining the Global Status History

Amazon RDS provides a set of procedures that take snapshots of the values of status variables over time and write them to a table, along with any changes since the last snapshot. This infrastructure is called Global Status History. For more information, see Managing the Global Status History for RDS for MySQL.

The following stored procedures manage how the Global Status History is collected and maintained.

Topics

- mysql.rds_collect_global_status_history
- mysql.rds_disable_gsh_collector
- mysql.rds_disable_gsh_rotation
- mysql.rds_enable_gsh_collector
- mysql.rds_enable_gsh_rotation
- mysql.rds_rotate_global_status_history
- mysql.rds_set_gsh_collector
- mysql.rds_set_gsh_rotation

mysql.rds_collect_global_status_history

Takes a snapshot on demand for the Global Status History.

Syntax

```
CALL mysql.rds_collect_global_status_history;
```

mysql.rds_disable_gsh_collector

Turns off snapshots taken by the Global Status History.

Syntax

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

Turns off rotation of the `mysql.global_status_history` table.

Syntax

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

Turns on the Global Status History to take default snapshots at intervals specified by `rds_set_gsh_collector`.

Syntax

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

Turns on rotation of the contents of the `mysql.global_status_history` table to `mysql.global_status_history_old` at intervals specified by `rds_set_gsh_rotation`.

Syntax

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

Rotates the contents of the `mysql.global_status_history` table to `mysql.global_status_history_old` on demand.

Syntax

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

Specifies the interval, in minutes, between snapshots taken by the Global Status History.

Syntax

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

Parameters

intervalPeriod

The interval, in minutes, between snapshots. Default value is 5.

mysql.rds_set_gsh_rotation

Specifies the interval, in days, between rotations of the `mysql.global_status_history` table.

Syntax

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

Parameters

intervalPeriod

The interval, in days, between table rotations. Default value is 7.

Configuring, starting, and stopping binary log (binlog) replication

The following stored procedures control how transactions are replicated from an external database into RDS for MySQL, or from RDS for MySQL to an external database.

When using these stored procedures to manage replication with a replication user configured with `caching_sha2_password`, you must configure TLS by specifying `SOURCE_SSL=1`. `caching_sha2_password` is the default authentication plugin for RDS for MySQL 8.4 For more information, see Encrypting with SSL/TLS.

For information about configuring, using, and managing read replicas, see the section called "MySQL read replicas".

Topics

- mysql.rds_next_master_log (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)
- mysql.rds_next_source_log (RDS for MySQL major versions 8.4 and higher)
- mysql.rds_reset_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)
- mysql.rds_reset_external_source (RDS for MySQL major versions 8.4 and higher)
- mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)
- mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher)
- mysql.rds_set_external_master_with_auto_position (RDS for MySQL major versions 8.0 and lower)
- mysql.rds_set_external_source_with_auto_position (RDS for MySQL major versions 8.4 and higher)
- mysql.rds_set_external_master_with_delay (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)
- mysql.rds_set_external_source_with_delay (RDS for MySQL major versions 8.4 and higher)
- mysql.rds_set_external_source_gtid_purged
- mysql.rds_set_master_auto_position (RDS for MySQL major versions 8.0 and lower)
- mysql.rds_set_source_auto_position (RDS for MySQL major versions 8.4 and higher)
- mysql.rds_set_source_delay
- mysql.rds_skip_repl_error

- mysql.rds_start_replication
- mysql.rds_start_replication_until
- mysql.rds_stop_replication

mysql.rds_next_master_log (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)

Changes the source database instance log position to the start of the next binary log on the source database instance. Use this procedure only if you are receiving replication I/O error 1236 on a read replica.

Syntax

```
CALL mysql.rds_next_master_log(
curr_master_log
);
```

Parameters

curr_master_log

The index of the current master log file. For example, if the current file is named `mysql-bin-changelog.012345`, then the index is 12345. To determine the current master log file name, run the `SHOW REPLICA STATUS` command and view the `Master_Log_File` field.

Usage notes

The master user must run the `mysql.rds_next_master_log` procedure.

Warning

Call `mysql.rds_next_master_log` only if replication fails after a failover of a Multi-AZ DB instance that is the replication source, and the `Last_IO_Errno` field of `SHOW REPLICA STATUS` reports I/O error 1236.
Calling `mysql.rds_next_master_log` can result in data loss in the read replica if transactions in the source instance were not written to the binary log on disk before the failover event occurred. You can reduce the chance of this happening by setting the source

instance parameters `sync_binlog` and `innodb_support_xa` to 1, even though this might reduce performance. For more information, see Troubleshooting a MySQL read replica problem.

Examples

Assume replication fails on an RDS for MySQL read replica. Running `SHOW REPLICA STATUS\G` on the read replica returns the following result:

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
```

```
        Seconds_Behind_Master: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from master when reading data from
 binary log: 'Client requested master to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

The `Last_IO_Errno` field shows that the instance is receiving I/O error 1236. The `Master_Log_File` field shows that the file name is `mysql-bin-changelog.012345`, which means that the log file index is `12345`. To resolve the error, you can call `mysql.rds_next_master_log` with the following parameter:

```
CALL mysql.rds_next_master_log(12345);
```

mysql.rds_next_source_log (RDS for MySQL major versions 8.4 and higher)

Changes the source database instance log position to the start of the next binary log on the source database instance. Use this procedure only if you are receiving replication I/O error 1236 on a read replica.

Syntax

```
CALL mysql.rds_next_source_log(
curr_source_log
);
```

Parameters

curr_source_log

The index of the current source log file. For example, if the current file is named `mysql-bin-changelog.012345`, then the index is 12345. To determine the current source log file name, run the `SHOW REPLICA STATUS` command and view the `Source_Log_File` field.

Usage notes

The administrative user must run the `mysql.rds_next_source_log` procedure.

> **Warning**
>
> Call `mysql.rds_next_source_log` only if replication fails after a failover of a Multi-AZ DB instance that is the replication source, and the `Last_IO_Errno` field of `SHOW REPLICA STATUS` reports I/O error 1236.
> Calling `mysql.rds_next_source_log` can result in data loss in the read replica if transactions in the source instance were not written to the binary log on disk before the failover event occurred. You can reduce the chance of this happening by setting the source instance parameters `sync_binlog` and `innodb_support_xa` to 1, even though this might reduce performance. For more information, see Troubleshooting a MySQL read replica problem.

Examples

Assume replication fails on an RDS for MySQL read replica. Running `SHOW REPLICA STATUS\G` on the read replica returns the following result:

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
```

```
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from source when reading data from
 binary log: 'Client requested source to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

The `Last_IO_Errno` field shows that the instance is receiving I/O error 1236. The `Source_Log_File` field shows that the file name is `mysql-bin-changelog.012345`, which means that the log file index is `12345`. To resolve the error, you can call `mysql.rds_next_source_log` with the following parameter:

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_reset_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)

Reconfigures an RDS for MySQL DB instance to no longer be a read replica of an instance of MySQL running external to Amazon RDS.

Important

To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_reset_external_master;
```

Usage notes

The master user must run the `mysql.rds_reset_external_master` procedure. This procedure must be run on the MySQL DB instance to be removed as a read replica of a MySQL instance running external to Amazon RDS.

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas.

For more information about using replication to import data from an instance of MySQL running external to Amazon RDS, see Configuring binary log file position replication with an external source instance.

mysql.rds_reset_external_source (RDS for MySQL major versions 8.4 and higher)

Reconfigures an RDS for MySQL DB instance to no longer be a read replica of an instance of MySQL running external to Amazon RDS.

Important

To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_reset_external_source;
```

Usage notes

The administrative user must run the `mysql.rds_reset_external_source` procedure. This procedure must be run on the MySQL DB instance to be removed as a read replica of a MySQL instance running external to Amazon RDS.

Note

We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS.

For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas. For more information about using replication to import data from an instance of MySQL running external to Amazon RDS, see Configuring binary log file position replication with an external source instance.

mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS.

Important

To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Note

You can use the mysql.rds_set_external_master_with_delay (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) stored procedure to configure an external source database instance and delayed replication.

Syntax

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS to become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in replication_user_name.

mysql_binary_log_file_name

The name of the binary log on the source database instance that contains the replication information.

mysql_binary_log_file_location

The location in the mysql_binary_log_file_name binary log at which replication starts reading the replication information.

You can determine the binlog file name and location by running SHOW MASTER STATUS on the source database instance.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The MASTER_SSL_VERIFY_SERVER_CERT option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

Usage notes

The master user must run the mysql.rds_set_external_master procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run mysql.rds_set_external_master, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify replication_user_name and

`replication_user_password` values that indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

   MySQL 5.7

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
    'password';
   ```

   > **Note**
   >
   > Specify a password other than the prompt shown here as a security best practice.

2. On the external instance of MySQL, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the 'repl_user' user for your domain.

   MySQL 5.7

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

To use encrypted replication, configure source database instance to use SSL connections.

> **Note**
>
> We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas.

After calling `mysql.rds_set_external_master` to configure an Amazon RDS DB instance as a read replica, you can call mysql.rds_start_replication on the read replica to start the replication process. You can call mysql.rds_reset_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) to remove the read replica configuration.

When `mysql.rds_set_external_master` is called, Amazon RDS records the time, user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS.

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS.

Important

To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS to become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source database instance that contains the replication information.

mysql_binary_log_file_location

The location in the mysql_binary_log_file_name binary log at which replication starts reading the replication information.

You can determine the binlog file name and location by running SHOW MASTER STATUS on the source database instance.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The SOURCE_SSL_VERIFY_SERVER_CERT option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

Usage notes

The administrative user must run the mysql.rds_set_external_source procedure. This procedure must be run on the RDS for MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run mysql.rds_set_external_source, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify replication_user_name and replication_user_password values that indicate a replication user that has REPLICATION CLIENT and REPLICATION SLAVE permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

2. On the external instance of MySQL, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the 'repl_user' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

To use encrypted replication, configure source database instance to use SSL connections. Also, import the certificate authority certificate, client certificate, and client key into the DB instance or DB cluster using the mysql.rds_import_binlog_ssl_material procedure.

> **Note**
>
> We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas.

After calling `mysql.rds_set_external_source` to configure an RDS for MySQL DB instance as a read replica, you can call mysql.rds_start_replication on the read replica to start the replication process. You can call mysql.rds_reset_external_source (RDS for MySQL major versions 8.4 and higher) to remove the read replica configuration.

When `mysql.rds_set_external_source` is called, Amazon RDS records the time, user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

Examples

When run on an RDS for MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS.

```
call mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  1);
```

mysql.rds_set_external_master_with_auto_position (RDS for MySQL major versions 8.0 and lower)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. This procedure also configures delayed replication and replication based on global transaction identifiers (GTIDs).

> **Important**
>
> To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_set_external_master_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
  , delay
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS to become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The `MASTER_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source database instance.

The limit for this parameter is one day (86,400 seconds).

Usage notes

The master user must run the `mysql.rds_set_external_master_with_auto_position` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

This procedure is supported for all RDS for MySQL 5.7 versions, and RDS for MySQL 8.0.26 and higher 8.0 versions.

Before you run `mysql.rds_set_external_master_with_auto_position`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. On the external instance of MySQL, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the `'repl_user'` user for your domain.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   IDENTIFIED BY 'SomePassW0rd'
   ```

For more information, see Configuring binary log file position replication with an external source instance.

> **Note**
>
> We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored

procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas.

Before you call `mysql.rds_set_external_master_with_auto_position`, make sure to call the section called "mysql.rds_set_external_source_gtid_purged" to set the `gtid_purged` system variable with a specified GTID range from an external source.

After calling `mysql.rds_set_external_master_with_auto_position` to configure an Amazon RDS DB instance as a read replica, you can call mysql.rds_start_replication on the read replica to start the replication process. You can call mysql.rds_reset_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) to remove the read replica configuration.

When you call `mysql.rds_set_external_master_with_auto_position`, Amazon RDS records the time, the user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

For disaster recovery, you can use this procedure with the mysql.rds_start_replication_until or mysql.rds_start_replication_until_gtid stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_external_master_with_auto_position` procedure. After the `mysql.rds_start_replication_until_gtid` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

To use the `mysql.rds_rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the mysql.rds_skip_transaction_with_gtid stored procedure. For more information about working with GTID-based replication, see Using GTID-based replication.

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. It sets the minimum replication delay to one hour (3,600 seconds) on the MySQL DB instance. A change from the MySQL source database instance running external to Amazon RDS isn't applied on the MySQL DB instance read replica for at least one hour.

```
call mysql.rds_set_external_master_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'SomePassW0rd',
  1,
  3600);
```

mysql.rds_set_external_source_with_auto_position (RDS for MySQL major versions 8.4 and higher)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. This procedure also configures delayed replication and replication based on global transaction identifiers (GTIDs).

> **Important**
>
> To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_set_external_source_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
  , delay
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS to become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source database instance.

The limit for this parameter is one day (86,400 seconds).

Usage notes

The administrative user must run the `mysql.rds_set_external_source_with_auto_position` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run `mysql.rds_set_external_source_with_auto_position`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify

values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. On the external instance of MySQL, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the `'repl_user'` user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
IDENTIFIED BY 'SomePassW0rd'
```

For more information, see Configuring binary log file position replication with an external source instance.

> **Note**
>
> We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas.

Before you call `mysql.rds_set_external_source_with_auto_position`, make sure to call the section called “mysql.rds_set_external_source_gtid_purged” to set the `gtid_purged` system variable with a specified GTID range from an external source.

After calling `mysql.rds_set_external_source_with_auto_position` to configure an Amazon RDS DB instance as a read replica, you can call mysql.rds_start_replication on the read

replica to start the replication process. You can call mysql.rds_reset_external_source (RDS for MySQL major versions 8.4 and higher) to remove the read replica configuration.

When you call `mysql.rds_set_external_source_with_auto_position`, Amazon RDS records the time, the user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

For disaster recovery, you can use this procedure with the mysql.rds_start_replication_until or mysql.rds_start_replication_until_gtid stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_external_source_with_auto_position` procedure. After the `mysql.rds_start_replication_until_gtid` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

To use the `mysql.rds_rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the mysql.rds_skip_transaction_with_gtid stored procedure. For more information about working with GTID-based replication, see Using GTID-based replication.

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. It sets the minimum replication delay to one hour (3,600 seconds) on the MySQL DB instance. A change from the MySQL source database instance running external to Amazon RDS isn't applied on the MySQL DB instance read replica for at least one hour.

```
call mysql.rds_set_external_source_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'SomePassW0rd',
  1,
  3600);
```

mysql.rds_set_external_master_with_delay (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS and configures delayed replication.

> **Important**
>
> To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_set_external_master_with_delay(
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
  , delay
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS that will become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes SSH port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source database instance contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication will start reading the replication information.

You can determine the binlog file name and location by running `SHOW MASTER STATUS` on the source database instance.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The `MASTER_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source database instance.

The limit for this parameter is one day (86400 seconds).

Usage notes

The master user must run the `mysql.rds_set_external_master_with_delay` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run `mysql.rds_set_external_master_with_delay`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. On the external instance of MySQL, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the `'repl_user'` user for your domain.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   IDENTIFIED BY 'SomePassW0rd'
   ```

For more information, see Configuring binary log file position replication with an external source instance.

> **Note**
>
> We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to

Amazon RDS. For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas.

After calling `mysql.rds_set_external_master_with_delay` to configure an Amazon RDS DB instance as a read replica, you can call mysql.rds_start_replication on the read replica to start the replication process. You can call mysql.rds_reset_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) to remove the read replica configuration.

When you call `mysql.rds_set_external_master_with_delay`, Amazon RDS records the time, the user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

For disaster recovery, you can use this procedure with the mysql.rds_start_replication_until or mysql.rds_start_replication_until_gtid stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_external_master_with_delay` procedure. After the `mysql.rds_start_replication_until` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

To use the `mysql.rds_rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the mysql.rds_skip_transaction_with_gtid stored procedure. For more information about working with GTID-based replication, see Using GTID-based replication.

The `mysql.rds_set_external_master_with_delay` procedure is available in these versions of RDS for MySQL:

- MySQL 8.0.26 and higher 8.0 versions
- All 5.7 versions

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. It sets the minimum replication delay to one hour (3,600 seconds) on the MySQL DB instance. A change from the MySQL source database instance running external to Amazon RDS isn't applied on the MySQL DB instance read replica for at least one hour.

```
call mysql.rds_set_external_master_with_delay(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'SomePassW0rd',
  'mysql-bin-changelog.000777',
  120,
  1,
  3600);
```

mysql.rds_set_external_source_with_delay (RDS for MySQL major versions 8.4 and higher)

Configures an RDS for MySQL DB instance to be a read replica of an instance of MySQL running external to Amazon RDS and configures delayed replication.

> **Important**
>
> To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_set_external_source_with_delay (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
  , delay
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Amazon RDS that will become the source database instance.

host_port

The port used by the MySQL instance running external to Amazon RDS to be configured as the source database instance. If your network configuration includes SSH port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the MySQL instance running external to Amazon RDS. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source database instance contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication will start reading the replication information.

You can determine the binlog file name and location by running `SHOW MASTER STATUS` on the source database instance.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source database instance.

The limit for this parameter is one day (86400 seconds).

Usage notes

The administrative user must run the `mysql.rds_set_external_source_with_delay` procedure. This procedure must be run on the MySQL DB instance to be configured as the read replica of a MySQL instance running external to Amazon RDS.

Before you run `mysql.rds_set_external_source_with_delay`, you must configure the instance of MySQL running external to Amazon RDS to be a source database instance. To connect to the MySQL instance running external to Amazon RDS, you must specify values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external instance of MySQL.

To configure an external instance of MySQL as a source database instance

1. Using the MySQL client of your choice, connect to the external instance of MySQL and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. On the external instance of MySQL, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the `'repl_user'` user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
IDENTIFIED BY 'SomePassW0rd'
```

For more information, see Configuring binary log file position replication with an external source instance.

> **Note**
>
> We recommend that you use read replicas to manage replication between two Amazon RDS DB instances when possible. When you do so, we recommend that you use only this and other replication-related stored procedures. These practices enable more complex replication topologies between Amazon RDS DB instances. We offer these stored procedures primarily to enable replication with MySQL instances running external to Amazon RDS. For information about managing replication between Amazon RDS DB instances, see Working with DB instance read replicas.

After calling `mysql.rds_set_external_source_with_delay` to configure an Amazon RDS DB instance as a read replica, you can call mysql.rds_start_replication on the read replica to start the replication process. You can call mysql.rds_reset_external_source (RDS for MySQL major versions 8.4 and higher) to remove the read replica configuration.

When you call `mysql.rds_set_external_source_with_delay`, Amazon RDS records the time, the user, and an action of `set master` in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

For disaster recovery, you can use this procedure with the mysql.rds_start_replication_until or mysql.rds_start_replication_until_gtid stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_external_source_with_delay` procedure. After the `mysql.rds_start_replication_until` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

To use the `mysql.rds_rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the mysql.rds_skip_transaction_with_gtid stored procedure. For more information about working with GTID-based replication, see Using GTID-based replication.

Examples

When run on a MySQL DB instance, the following example configures the DB instance to be a read replica of an instance of MySQL running external to Amazon RDS. It sets the minimum replication delay to one hour (3,600 seconds) on the MySQL DB instance. A change from the MySQL source

database instance running external to Amazon RDS isn't applied on the MySQL DB instance read replica for at least one hour.

```
call mysql.rds_set_external_source_with_delay(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'SomePassW0rd',
  'mysql-bin-changelog.000777',
  120,
  1,
  3600);
```

mysql.rds_set_external_source_gtid_purged

Sets the gtid_purged system variable with a specified GTID range from an external source. The `gtid_purged` value is required for configuring GTID-based replication to resume the replication using auto positioning.

> **Important**
>
> To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_set_external_source_gtid_purged(
  server_uuid
  , start_pos
  , end_pos
);
```

Parameters

server_uuid

The universally unique identifier (UUID) of the external server from which the GTID range is being imported.

start_pos

The starting position of the GTID range to be set.

end_pos

The ending position of the GTID range to be set.

Usage notes

The mysql.rds_set_external_source_gtid_purged procedure is only available with MySQL 8.0.37 and higher 8.0 versions.

Call mysql.rds_set_external_source_gtid_purged before you call mysql.rds_set_external_master_with_auto_position (RDS for MySQL major versions 8.0 and lower), mysql.rds_set_external_source_with_auto_position (RDS for MySQL major versions 8.4 and higher), or mysql.rds_set_external_source_with_auto_position_for_channel.

Before you call mysql.rds_set_external_source_gtid_purged, make sure to stop all active replication channels for the database. To check the status of a channel, use the SHOW REPLICA STATUS MySQL statement. To stop replication on a channel, call the section called "mysql.rds_stop_replication_for_channel".

The GTID range that you specify must be a superset of the existing GTID_PURGED value. This stored procedure checks the following values before it sets the GTID_PURGED value:

- The server_uuid is valid.
- The value of start_pos is greater than 0 and less than the value of end_pos.
- The value of end_pos is greater than or equal to the value of start_pos.

If the GTID set on your external server contains multiple ranges of values, consider calling the procedure multiple times with different GTID set values.

When you call mysql.rds_set_external_source_gtid_purged, Amazon RDS records the time, the user, and an action of set gtid_purged in the mysql.rds_history table.

If you don't set the gtid_purged value appropriately for the backup that you use for replication, this can result in missing or duplicated transactions during the replication process. Perform the following steps to set the correct gtid_purged value.

To set the gtid_purged value on the replica

1. Determine the point in time or the specific backup file to use as the starting point for replication. This could be a logical backup (a mysqldump file) or a physical backup (an Amazon RDS snapshot).
2. Determine the `gtid_executed` value. This value represents the set of all GTIDs that were committed on the server. To retrieve this value, on the source instance, do one of the following:
   - Run the SQL statement `SELECT @@GLOBAL.GTID_EXECUTED;` at the time the backup was taken.
   - If any related options are included in the respective backup utility, extract the value from the backup file. For more information, see the set-gtid-purged option in the MySQL documentation.
3. Determine the `gtid_purged` value to use for the call to `mysql.rds_set_external_source_gtid_purged`. The `gtid_purged` value should include all the GTIDs that were executed on the source instance and are no longer needed for replication. Therefore, the `gtid_purged` value should be a subset of the `gtid_executed` value that you retrieved in the previous step.

   To determine the `gtid_purged` value, identify the GTIDs that aren't included in the backup and are no longer needed for replication. You can do so by analyzing the binary logs or by using a tool such as mysqlbinlog to find the GTIDs that were purged from the binary logs.

   Alternatively, if you have a consistent backup that includes all of the binary logs up to the backup point, you can set the `gtid_purged` value to be the same as the `gtid_executed` value at the backup point.
4. After you determine the appropriate `gtid_purged` value that's consistent with your backup, call the `mysql.rds_set_external_source_gtid_purged` stored procedure on your RDS for MySQL DB instance to set the value.

Examples

When run on a MySQL DB instance, the following example sets the GTID range from an external MySQL server with the UUID `12345678-abcd-1234-efgh-123456789abc`, a starting position of 1, and an ending position of `100`. The resulting GTID value is set to `+12345678-abcd-1234-efgh-123456789abc:1-100`.

```
CALL mysql.rds_set_external_source_gtid_purged('12345678-abcd-1234-efgh-123456789abc',
 1, 100);
```

mysql.rds_set_master_auto_position (RDS for MySQL major versions 8.0 and lower)

Sets the replication mode to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_master_auto_position (
auto_position_mode
);
```

Parameters

auto_position_mode

A value that indicates whether to use log file position replication or GTID-based replication:

- `0` – Use the replication method based on binary log file position. The default is `0`.
- 1 – Use the GTID-based replication method.

Usage notes

The master user must run the `mysql.rds_set_master_auto_position` procedure.

This procedure is supported for all RDS for MySQL 5.7 versions and RDS for MySQL 8.0.26 and higher 8.0 versions.

mysql.rds_set_source_auto_position (RDS for MySQL major versions 8.4 and higher)

Sets the replication mode to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

Parameters

auto_position_mode

A value that indicates whether to use log file position replication or GTID-based replication:

- 0 – Use the replication method based on binary log file position. The default is 0.
- 1 – Use the GTID-based replication method.

Usage notes

The administrative user must run the mysql.rds_set_source_auto_position procedure.

mysql.rds_set_source_delay

Sets the minimum number of seconds to delay replication from source database instance to the current read replica. Use this procedure when you are connected to a read replica to delay replication from its source database instance.

Syntax

```
CALL mysql.rds_set_source_delay(
delay
);
```

Parameters

delay

The minimum number of seconds to delay replication from the source database instance.

The limit for this parameter is one day (86400 seconds).

Usage notes

The master user must run the mysql.rds_set_source_delay procedure.

For disaster recovery, you can use this procedure with the mysql.rds_start_replication_until stored procedure or the mysql.rds_start_replication_until_gtid stored procedure.
To roll forward changes to a delayed read replica to the time just before a

disaster, you can run the `mysql.rds_set_source_delay` procedure. After the `mysql.rds_start_replication_until` or `mysql.rds_start_replication_until_gtid` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

To use the `mysql.rds_rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the mysql.rds_skip_transaction_with_gtid stored procedure. For more information on GTID-based replication, see Using GTID-based replication.

The `mysql.rds_set_source_delay` procedure is available in these versions of RDS for MySQL:

- All RDS for MySQL 8.4 versions
- MySQL 8.0.26 and higher 8.0 versions
- All 5.7 versions

Examples

To delay replication from source database instance to the current read replica for at least one hour (3,600 seconds), you can call `mysql.rds_set_source_delay` with the following parameter:

```
CALL mysql.rds_set_source_delay(3600);
```

mysql.rds_skip_repl_error

Skips and deletes a replication error on a MySQL DB read replica.

Syntax

```
CALL mysql.rds_skip_repl_error;
```

Usage notes

The master user must run the `mysql.rds_skip_repl_error` procedure on a read replica. For more information about this procedure, see Calling the mysql.rds_skip_repl_error procedure.

To determine if there are errors, run the MySQL `SHOW REPLICA STATUS\G` command. If a replication error isn't critical, you can run `mysql.rds_skip_repl_error` to skip the error. If

there are multiple errors, `mysql.rds_skip_repl_error` deletes the first error, then warns that others are present. You can then use `SHOW REPLICA STATUS\G` to determine the correct course of action for the next error. For information about the values returned, see SHOW REPLICA STATUS statement in the MySQL documentation.

For more information about addressing replication errors with Amazon RDS, see Troubleshooting a MySQL read replica problem.

Replication stopped error

When you call the `mysql.rds_skip_repl_error` procedure, you might receive an error message stating that the replica is down or disabled.

This error message appears if you run the procedure on the primary instance instead of the read replica. You must run this procedure on the read replica for the procedure to work.

This error message might also appear if you run the procedure on the read replica, but replication can't be restarted successfully.

If you need to skip a large number of errors, the replication lag can increase beyond the default retention period for binary log (binlog) files. In this case, you might encounter a fatal error because of binlog files being purged before they have been replayed on the read replica. This purge causes replication to stop, and you can no longer call the `mysql.rds_skip_repl_error` command to skip replication errors.

You can mitigate this issue by increasing the number of hours that binlog files are retained on your source database instance. After you have increased the binlog retention time, you can restart replication and call the `mysql.rds_skip_repl_error` command as needed.

To set the binlog retention time, use the mysql.rds_set_configuration procedure and specify a configuration parameter of `'binlog retention hours'` along with the number of hours to retain binlog files on the DB cluster. The following example sets the retention period for binlog files to 48 hours.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

Initiates replication from an RDS for MySQL DB instance.

> **Note**
>
> You can use the mysql.rds_start_replication_until or mysql.rds_start_replication_until_gtid stored procedure to initiate replication from an RDS for MySQL DB instance and stop replication at the specified binary log file location.

Syntax

```
CALL mysql.rds_start_replication;
```

Usage notes

The master user must run the `mysql.rds_start_replication` procedure.

To import data from an instance of MySQL external to Amazon RDS, call `mysql.rds_start_replication` on the read replica to start the replication process after you call mysql.rds_set_external_master (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) or mysql.rds_set_external_source (RDS for MySQL major versions 8.4 and higher) to build the replication configuration. For more information, see Restoring a backup into an Amazon RDS for MySQL DB instance.

To export data to an instance of MySQL external to Amazon RDS, call `mysql.rds_start_replication` and `mysql.rds_stop_replication` on the read replica to control some replication actions, such as purging binary logs. For more information, see Exporting data from a MySQL DB instance by using replication.

You can also call `mysql.rds_start_replication` on the read replica to restart any replication process that you previously stopped by calling `mysql.rds_stop_replication`. For more information, see Working with DB instance read replicas.

mysql.rds_start_replication_until

Initiates replication from an RDS for MySQL DB instance and stops replication at the specified binary log file location.

Syntax

```
CALL mysql.rds_start_replication_until (
replication_log_file
  , replication_stop_point
);
```

Parameters

replication_log_file

The name of the binary log on the source database instance that contains the replication information.

replication_stop_point

The location in the `replication_log_file` binary log at which replication will stop.

Usage notes

The master user must run the `mysql.rds_start_replication_until` procedure.

The `mysql.rds_start_replication_until` procedure is available in these versions of RDS for MySQL:

- All RDS for MySQL 8.4 versions
- MySQL 8.0.26 and higher 8.0 versions
- All 5.7 versions

You can use this procedure with delayed replication for disaster recovery. If you have delayed replication configured, you can use this procedure to roll forward changes to a delayed read replica to the time just before a disaster. After this procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

You can configure delayed replication using the following stored procedures:

- mysql.rds_set_configuration
- mysql.rds_set_external_master_with_delay (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)
- mysql.rds_set_external_source_with_delay (RDS for MySQL major versions 8.4 and higher)

- mysql.rds_set_source_delay

The file name specified for the `replication_log_file` parameter must match the source database instance binlog file name.

When the `replication_stop_point` parameter specifies a stop location that is in the past, replication is stopped immediately.

Examples

The following example initiates replication and replicates changes until it reaches location `120` in the `mysql-bin-changelog.000777` binary log file.

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

mysql.rds_stop_replication

Stops replication from a MySQL DB instance.

Syntax

```
CALL mysql.rds_stop_replication;
```

Usage notes

The master user must run the `mysql.rds_stop_replication` procedure.

If you are configuring replication to import data from an instance of MySQL running external to Amazon RDS, you call `mysql.rds_stop_replication` on the read replica to stop the replication process after the import has completed. For more information, see Restoring a backup into an Amazon RDS for MySQL DB instance.

If you are configuring replication to export data to an instance of MySQL external to Amazon RDS, you call `mysql.rds_start_replication` and `mysql.rds_stop_replication` on the read replica to control some replication actions, such as purging binary logs. For more information, see Exporting data from a MySQL DB instance by using replication.

You can also use `mysql.rds_stop_replication` to stop replication between two Amazon RDS DB instances. You typically stop replication to perform a long running operation on the read replica, such as creating a large index on the read replica. You can restart any replication process that you stopped by calling mysql.rds_start_replication on the read replica. For more information, see Working with DB instance read replicas.

Ending a session or query

The following stored procedures end a session or query.

Topics

- mysql.rds_kill
- mysql.rds_kill_query

mysql.rds_kill

Ends a connection to the MySQL server.

Syntax

```
CALL mysql.rds_kill(processID);
```

Parameters

processID

The identity of the connection thread to be ended.

Usage notes

Each connection to the MySQL server runs in a separate thread. To end a connection, use the `mysql.rds_kill` procedure and pass in the thread ID of that connection. To obtain the thread ID, use the MySQL SHOW PROCESSLIST command.

For information about limitations, see MySQL stored procedure limitations.

Examples

The following example ends a connection with a thread ID of 4243:

```
CALL mysql.rds_kill(4243);
```

mysql.rds_kill_query

Ends a query running against the MySQL server.

Syntax

```
CALL mysql.rds_kill_query(processID);
```

Parameters

processID

The identity of the process or thread that is running the query to be ended.

Usage notes

To stop a query running against the MySQL server, use the `mysql_rds_kill_query` procedure and pass in the connection ID of the thread that is running the query. The procedure then terminates the connection.

To obtain the ID, query the MySQL INFORMATION_SCHEMA PROCESSLIST table or use the MySQL SHOW PROCESSLIST command. The value in the ID column from `SHOW PROCESSLIST` or `SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST` is the *processID*.

For information about limitations, see MySQL stored procedure limitations.

Examples

The following example stops a query with a query thread ID of 230040:

```
CALL mysql.rds_kill_query(230040);
```

Managing active-active clusters

The following stored procedures set up and manage RDS for MySQL active-active clusters. For more information, see the section called “Configuring active-active clusters”.

These stored procedures are only available with RDS for MySQL DB instances running the following versions:

- All MySQL 8.4 versions
- MySQL 8.0.35 and higher minor versions

Topics

- mysql.rds_group_replication_advance_gtid
- mysql.rds_group_replication_create_user
- mysql.rds_group_replication_set_recovery_channel
- mysql.rds_group_replication_start
- mysql.rds_group_replication_stop

mysql.rds_group_replication_advance_gtid

Creates placeholder GTIDs on the current DB instance.

Syntax

```
CALL mysql.rds_group_replication_advance_gtid(
begin_id
, end_id
, server_uuid
);
```

Parameters

begin_id

The start transaction ID to be created.

end_id

The end transaction ID to be created.

begin_id

The `group_replication_group_name` for the transaction to be created. The `group_replication_group_name` is specified as a UUID in the DB parameter group associated with the DB instance.

Usage notes

In an active-active cluster, for a DB instance to join a group, all GTID transactions executed on the new DB instance must exist on the other members in the cluster. In unusual cases, a new DB instance might have more transactions when transactions are executed before joining the instance to group. In this case, you can't remove any existing transactions, but you can use this procedure to create the corresponding placeholder GTIDs on the othe DB instances in the group. Before doing so, verify that the transactions *don't affect the replicated data*.

When you call this procedure, GTID transactions of `server_uuid:begin_id-end_id` are created with empty content. To avoid replication issues, don't use this procedure under any other conditions.

> **Important**
>
> Avoid calling this procedure when the active-active cluster is functioning normally. Don't call this procedure unless you understand the possible consequences of the transactions you are creating. Calling this procedure might result in inconsistent data.

Example

The following example creates placeholder GTIDs on current DB instance.:

```
CALL mysql.rds_group_replication_advance_gtid(5, 6,
 '11111111-2222-3333-4444-555555555555');
```

mysql.rds_group_replication_create_user

Creates the replication user `rdsgrprepladmin` for group replication on the DB instance.

Syntax

```
CALL mysql.rds_group_replication_create_user(
```

```
replication_user_password
);
```

Parameters

replication_user_password

The password of the replication user `rdsgrprepladmin`.

Usage notes

- The password of the replication user `rdsgrprepladmin` must be the same on all of the DB instances in an active-active cluster.
- The `rdsgrprepladmin` user name is reserved for group replication connections. No other user, including the master user, can have this user name.

Example

The following example creates the replication user `rdsgrprepladmin` for group replication on the DB instance:

```
CALL mysql.rds_group_replication_create_user('password');
```

mysql.rds_group_replication_set_recovery_channel

Sets the `group_replication_recovery` channel for an active-active cluster. The procedure uses the reserved `rdsgrprepladmin` user to configure the channel.

Syntax

```
CALL mysql.rds_group_replication_set_recovery_channel(
replication_user_password);
```

Parameters

replication_user_password

The password of the replication user `rdsgrprepladmin`.

Usage notes

The password of the replication user `rdsgrprepladmin` must be the same on all of the DB instances in an active-active cluster. A call to the `mysql.rds_group_replication_create_user` specifies the password.

Example

The following example sets the `group_replication_recovery` channel for an active-active cluster:

```
CALL mysql.rds_group_replication_set_recovery_channel('password');
```

mysql.rds_group_replication_start

Starts group replication on the current DB instance.

Syntax

```
CALL mysql.rds_group_replication_start(
bootstrap
);
```

Parameters

`bootstrap`

A value that specifies whether to initialize a new group or join an existing group. `1` initializes a new group with the current DB instance. `0` joins the current DB instance to an existing group by connecting to the endpoints defined in `group_replication_group_seeds` parameter in the of DB parameter group associated with the DB instance.

Example

The following example initializes a new group with the current DB instance:

```
CALL mysql.rds_group_replication_start(1);
```

mysql.rds_group_replication_stop

Stops group replication on the current DB instance.

Syntax

```
CALL mysql.rds_group_replication_stop();
```

Usage notes

When you stop replication on a DB instance, it doesn't affect any other DB instance in the active-active cluster.

Managing multi-source replication

The following stored procedures set up and manage replication channels on a RDS for MySQL multi-source replica. For more information, see the section called "Configuring multi-source replication".

These stored procedures are only available with RDS for MySQL DB instances running the following engine versions:

- All 8.4 versions
- 8.0.35 and higher minor versions
- 5.7.44 and higher minor versions

When using stored procedures to manage replication with a replication user configured with `caching_sha2_passwword`, you must configure TLS by specifying `SOURCE_SSL=1`. `caching_sha2_password` is the default authentication plugin for RDS for MySQL 8.4.

> **Note**
>
> Although this documentation refers to source DB instances as RDS for MySQL DB instances, these procedures also work for MySQL instances running external to Amazon RDS.

Topics

- mysql.rds_next_source_log_for_channel
- mysql.rds_reset_external_source_for_channel
- mysql.rds_set_external_source_for_channel
- mysql.rds_set_external_source_with_auto_position_for_channel
- mysql.rds_set_external_source_with_delay_for_channel
- mysql.rds_set_source_auto_position_for_channel
- mysql.rds_set_source_delay_for_channel
- mysql.rds_skip_repl_error_for_channel
- mysql.rds_start_replication_for_channel
- mysql.rds_start_replication_until_for_channel
- mysql.rds_start_replication_until_gtid_for_channel

- mysql.rds_stop_replication_for_channel

mysql.rds_next_source_log_for_channel

Changes the source DB instance log position to the start of the next binary log on the source DB instance for the channel. Use this procedure only if you are receiving replication I/O error 1236 on a multi-source replica.

Syntax

```
CALL mysql.rds_next_source_log_for_channel(
curr_master_log,
channel_name
);
```

Parameters

curr_master_log

The index of the current source log file. For example, if the current file is named `mysql-bin-changelog.012345`, then the index is 12345. To determine the current source log file name, run the `SHOW REPLICA STATUS FOR CHANNEL '`*channel_name*`'` command and view the `Source_Log_File` field.

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_next_source_log_for_channel` procedure. If there is an IO_Thread error, for example, you can use this procedure to skip all the events in the current binary log file and resume the replication from the next binary log file for the channel specified in `channel_name`.

Example

Assume replication fails on a channel on a multi-source replica. Running `SHOW REPLICA STATUS FOR CHANNEL 'channel_1'\G` on the multi-source replica returns the following result:

```
mysql> SHOW REPLICA STATUS FOR CHANNEL 'channel_1'\G
*************************** 1. row ***************************
             Replica_IO_State: Waiting for source to send event
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: ReplicationUser
                  Source_Port: 3306
                Connect_Retry: 60
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: replica-relay-bin.000003
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:.
              .
              .
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from master when reading data from
 binary log: 'Client requested master to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
               .
               .
                 Channel_name: channel_1
              .
              .
 -- Some fields are omitted in this example output
```

The `Last_IO_Errno` field shows that the instance is receiving I/O error 1236. The `Source_Log_File` field shows that the file name is `mysql-bin-changelog.012345`, which means that the log file index is `12345`. To resolve the error, you can call `mysql.rds_next_source_log_for_channel` with the following parameters:

```
CALL mysql.rds_next_source_log_for_channel(12345,'channel_1');
```

mysql.rds_reset_external_source_for_channel

Stops the replication process on the specified channel, and removes the channel and associated configurations from the multi-source replica.

> **Important**
>
> To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_reset_external_source_for_channel (channel_name);
```

Parameters

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_reset_external_source_for_channel` procedure. This procedure deletes all relay logs that belong to the channel being removed.

mysql.rds_set_external_source_for_channel

Configures a replication channel on an RDS for MySQL DB instance to replicate the data from another RDS for MySQL DB instance.

Important

To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Note

You can use the the section called "mysql.rds_set_external_source_with_delay_for_channel" stored procedure instead to configure this channel with delayed replication.

Syntax

```
CALL mysql.rds_set_external_source_for_channel (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
  , channel_name
);
```

Parameters

host_name

The host name or IP address of the RDS for MySQL source DB instance.

host_port

The port used by the RDS for MySQL source DB instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the RDS for MySQL source DB instance. We recommend that you provide an account that is used solely for replication with the source DB instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source DB instance that contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication starts reading the replication information.

You can determine the binlog file name and location by running `SHOW BINARY LOG STATUS` on the source DB instance.

> **Note**
>
> Previous versions of MySQL used `SHOW MASTER STATUS` instead of `SHOW BINARY LOG STATUS`. If you are using a MySQL version before 8.4, then use `SHOW MASTER STATUS`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

channel_name

The name of the replication channel. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_external_source_for_channel` procedure. This procedure must be run on the target RDS for MySQL DB instance on which you're creating the replication channel.

Before you run `mysql.rds_set_external_source_for_channel`, configure a replication user on the source DB instance with the privileges required for the multi-source replica. To connect the multi-source replica to the source DB instance, you must specify `replication_user_name` and `replication_user_password` values of a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the source DB instance.

To configure a replication user on the source DB instance

1. Using the MySQL client of your choice, connect to the source DB instance and create a user account to be used for replication. The following is an example.

   > **Important**
   >
   > As a security best practice, specify a password other than the placeholder value shown in the following examples.

   ```
   CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
   ```

2. On the source DB instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the 'repl_user' user for your domain.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com';
   ```

To use encrypted replication, configure the source DB instance to use SSL connections.

After calling `mysql.rds_set_external_source_for_channel` to configure this replication channel, you can call mysql.rds_start_replication_for_channel on the replica to start the replication process on the channel. You can call the section called "mysql.rds_reset_external_source_for_channel" to stop replication on the channel and remove the channel configuration from the replica.

When you call `mysql.rds_set_external_source_for_channel`, Amazon RDS records the time, user, and an action of `set channel source` in the `mysql.rds_history` table without channel-specific details, and in the `mysql.rds_replication_status` table, with the channel name. This information is recorded only for internal usage and monitoring purposes. To record the complete procedure call for auditing purpose, consider enabling audit logs or general logs, based on the specific requirements of your application.

Examples

When run on a RDS for MySQL DB instance, the following example configures a replication channel named `channel_1` on this DB instance to replicate data from the source specified by host `sourcedb.example.com` and port `3306`.

```
call mysql.rds_set_external_source_for_channel(
  'sourcedb.example.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  0,
  'channel_1');
```

mysql.rds_set_external_source_with_auto_position_for_channel

Configures a replication channel on an RDS for MySQL DB instance with an optional replication delay. The replication is based on global transaction identifiers (GTIDs).

> **Important**
>
> To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_set_external_source_with_auto_position_for_channel (
  host_name
  , host_port
```

```
  , replication_user_name
  , replication_user_password
  , ssl_encryption
  , delay
  , channel_name
);
```

Parameters

host_name

The host name or IP address of the RDS for MySQL source DB instance.

host_port

The port used by the RDS for MySQL source DB instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the RDS for MySQL source DB instance. We recommend that you provide an account that is used solely for replication with the source DB instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source DB instance.

The limit for this parameter is one day (86,400 seconds).

channel_name

The name of the replication channel. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_external_source_with_auto_position_for_channel` procedure. This procedure must be run on the target RDS for MySQL DB instance on which you're creating the replication channel.

Before you run `rds_set_external_source_with_auto_position_for_channel`, configure a replication user on the source DB instance with the privileges required for the multi-source replica. To connect the multi-source replica to the source DB instance, you must specify `replication_user_name` and `replication_user_password` values of a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the source DB instance.

To configure a replication user on the source DB instance

1. Using the MySQL client of your choice, connect to the source DB instance and create a user account to be used for replication. The following is an example.

   Important

   As a security best practice, specify a password other than the placeholder value shown in the following examples.

   ```
   CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
   ```

2. On the source DB instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the 'repl_user' user for your domain.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com';
   ```

To use encrypted replication, configure the source DB instance to use SSL connections.

Before you call `mysql.rds_set_external_source_with_auto_position_for_channel`, make sure to call the section called "mysql.rds_set_external_source_gtid_purged" to set the `gtid_purged` system variable with a specified GTID range from an external source.

After calling `mysql.rds_set_external_source_with_auto_position_for_channel` to configure an Amazon RDS DB instance as a read replica on a specific channel, you can call the section called "mysql.rds_start_replication_for_channel" on the read replica to start the replication process on that channel.

After calling `mysql.rds_set_external_source_with_auto_position_for_channel` to configure this replication channel, you can call mysql.rds_start_replication_for_channel on the replica to start the replication process on the channel. You can call the section called "mysql.rds_reset_external_source_for_channel" to stop replication on the channel and remove the channel configuration from the replica.

Examples

When run on a RDS for MySQL DB instance, the following example configures a replication channel named `channel_1` on this DB instance to replicate data from the source specified by host `sourcedb.example.com` and port `3306` It sets the minimum replication delay to one hour (3,600 seconds). This means that a change from the source RDS for MySQL DB instance isn't applied on the multi-source replica for at least one hour.

```
call mysql.rds_set_external_source_with_auto_position_for_channel(
  'sourcedb.example.com',
  3306,
  'repl_user',
  'password',
  1,
  3600,
  'channel_1');
```

mysql.rds_set_external_source_with_delay_for_channel

Configures a replication channel on an RDS for MySQL DB instance with a specified replication delay.

Important

To run this procedure, `autocommit` must be enabled. To enable it, set the `autocommit` parameter to 1. For information about modifying parameters, see Modifying parameters in a DB parameter group in Amazon RDS.

Syntax

```
CALL mysql.rds_set_external_source_with_delay_for_channel (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
  , delay
  , channel_name
);
```

Parameters

host_name

The host name or IP address of the RDS for MySQL source DB instance.

host_port

The port used by the RDS for MySQL source DB instance. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the RDS for MySQL source DB instance. We recommend that you provide an account that is used solely for replication with the source DB instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

mysql_binary_log_file_name

The name of the binary log on the source DB instance contains the replication information.

mysql_binary_log_file_location

The location in the `mysql_binary_log_file_name` binary log at which replication will start reading the replication information.

You can determine the binlog file name and location by running `SHOW BINARY LOG STATUS` on the source database instance.

> **Note**
>
> Previous versions of MySQL used `SHOW MASTER STATUS` instead of `SHOW BINARY LOG STATUS`. If you are using a MySQL version before 8.4, then use `SHOW MASTER STATUS`.

ssl_encryption

A value that specifies whether Secure Socket Layer (SSL) encryption is used on the replication connection. 1 specifies to use SSL encryption, 0 specifies to not use encryption. The default is 0.

> **Note**
>
> The `SOURCE_SSL_VERIFY_SERVER_CERT` option isn't supported. This option is set to 0, which means that the connection is encrypted, but the certificates aren't verified.

delay

The minimum number of seconds to delay replication from source DB instance.

The limit for this parameter is one day (86400 seconds).

channel_name

The name of the replication channel. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_external_source_with_delay_for_channel` procedure. This procedure must be run on the target RDS for MySQL DB instance on which you're creating the replication channel.

Before you run `mysql.rds_set_external_source_with_delay_for_channel`, configure a replication user on the source DB instance with the privileges required for the multi-source replica. To connect the multi-source replica to the source DB instance, you must specify `replication_user_name` and `replication_user_password` values of a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the source DB instance.

To configure a replication user on the source DB instance

1. Using the MySQL client of your choice, connect to the source DB instance and create a user account to be used for replication. The following is an example.

   > **Important**
   >
   > As a security best practice, specify a password other than the placeholder value shown in the following examples.

   ```
   CREATE USER 'repl_user'@'example.com' IDENTIFIED BY 'password';
   ```

2. On the source DB instance, grant `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges to your replication user. The following example grants `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges on all databases for the 'repl_user' user for your domain.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'example.com';
   ```

To use encrypted replication, configure the source DB instance to use SSL connections.

After calling `mysql.rds_set_external_source_with_delay_for_channel` to configure this replication channel, you can call mysql.rds_start_replication_for_channel on the replica to start the replication process on the channel. You can call the section called "mysql.rds_reset_external_source_for_channel" to stop replication on the channel and remove the channel configuration from the replica.

When you call `mysql.rds_set_external_source_with_delay_for_channel`, Amazon RDS records the time, user, and an action of `set channel source` in the `mysql.rds_history` table without channel-specific details, and in the `mysql.rds_replication_status` table, with the channel name. This information is recorded only for internal usage and monitoring purposes. To record the complete procedure call for auditing purpose, consider enabling audit logs or general logs, based on the specific requirements of your application.

Examples

When run on a RDS for MySQL DB instance, the following example configures a replication channel named `channel_1` on this DB instance to replicate data from the source specified by host `sourcedb.example.com` and port `3306` It sets the minimum replication delay to one hour (3,600 seconds). This means that a change from the source RDS for MySQL DB instance isn't applied on the multi-source replica for at least one hour.

```
call mysql.rds_set_external_source_with_delay_for_channel(
  'sourcedb.example.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.000777',
  120,
  1,
  3600,
  'channel_1');
```

mysql.rds_set_source_auto_position_for_channel

Sets the replication mode for the specified channel to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_source_auto_position_for_channel (
auto_position_mode
 , channel_name
);
```

Parameters

auto_position_mode

A value that indicates whether to use log file position replication or GTID-based replication:

- `0` – Use the replication method based on binary log file position. The default is `0`.
- 1 – Use the GTID-based replication method.

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_source_auto_position_for_channel` procedure. This procedure restarts replication on the specified channel to apply the specified auto position mode.

Examples

The following example sets the auto position mode for channel_1 to use the GTID-based replication method.

```
call mysql.rds_set_source_auto_position_for_channel(1,'channel_1');
```

mysql.rds_set_source_delay_for_channel

Sets the minimum number of seconds to delay replication from the source database instance to the multi-source replica for the specified channel.

Syntax

```
CALL mysql.rds_set_source_delay_for_channel(delay, channel_name);
```

Parameters

delay

The minimum number of seconds to delay replication from the source DB instance.

The limit for this parameter is one day (86400 seconds).

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_set_source_delay_for_channel` procedure. To use this procedure, first call `mysql.rds_stop_replication_for_channel` to stop the replication. Then, call this procedure to set the replication delay value. When the delay is set, call `mysql.rds_start_replication_for_channel` to restart the replication.

Examples

The following example sets the delay for replication from the source database instance on `channel_1` of the multi-source replica for at least one hour (3,600 seconds).

```
CALL mysql.rds_set_source_delay_for_channel(3600,'channel_1');
```

mysql.rds_skip_repl_error_for_channel

Skips a binary log event and deletes a replication error on a MySQL DB multi-source replica for the specified channel.

Syntax

```
CALL mysql.rds_skip_repl_error_for_channel(channel_name);
```

Parameters

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_skip_repl_error_for_channel` procedure on a read replica. You can use this procedure in a similar way `mysql.rds_skip_repl_error` is used to skip an error on a read replica. For more information, see Calling the mysql.rds_skip_repl_error procedure.

> **Note**
>
> To skip errors in GTID-based replication, we recommend that you use the procedure the section called “mysql.rds_skip_transaction_with_gtid” instead.

To determine if there are errors, run the MySQL `SHOW REPLICA STATUS FOR CHANNEL 'channel_name'\G` command. If a replication error isn't critical, you can run `mysql.rds_skip_repl_error_for_channel` to skip the error. If there are multiple errors, `mysql.rds_skip_repl_error_for_channel` deletes the first error on the specified replication channel, then warns that others are present. You can then use `SHOW REPLICA STATUS FOR CHANNEL 'channel_name'\G` to determine the correct course of action for the next error. For information about the values returned, see SHOW REPLICA STATUS statement in the MySQL documentation.

mysql.rds_start_replication_for_channel

Initiates replication from an RDS for MySQL DB instance to a multi-source replica on the specified channel.

> **Note**
>
> You can use the mysql.rds_start_replication_until_for_channel or mysql.rds_start_replication_until_gtid_for_channel stored procedure to initiate replication from an RDS for MySQL DB instance and stop replication at the specified binary log file location.

Syntax

```
CALL mysql.rds_start_replication_for_channel(channel_name);
```

Parameters

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_start_replication_for_channel` procedure. After you import the data from the source RDS for MySQL DB instance, run this command on the multi-source replica to start replication on the specified channel.

Examples

The following example starts replication on `channel_1` of the multi-source replica.

```
CALL mysql.rds_start_replication_for_channel('channel_1');
```

mysql.rds_start_replication_until_for_channel

Initiates replication from an RDS for MySQL DB instance on the specified channel and stops replication at the specified binary log file location.

Syntax

```
CALL mysql.rds_start_replication_until_for_channel (
replication_log_file
  , replication_stop_point
  , channel_name
);
```

Parameters

replication_log_file

The name of the binary log on the source DB instance contains the replication information.

replication_stop_point

The location in the replication_log_file binary log at which replication will stop.

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the mysql.rds_start_replication_until_for_channel procedure. With this procedure, replication starts and then stops when the specified binlog file position is reached. This procedure stops both the SQL_THREAD and IO_THREAD.

The file name specified for the replication_log_file parameter must match the source DB instance binlog file name.

When the replication_stop_point parameter specifies a stop location that's in the past, replication is stopped immediately.

Examples

The following example initiates replication on channel_1, and replicates changes until it reaches location 120 in the mysql-bin-changelog.000777 binary log file.

```
call mysql.rds_start_replication_until_for_channel(
  'mysql-bin-changelog.000777',
  120,
  'channel_1'
  );
```

mysql.rds_start_replication_until_gtid_for_channel

Initiates replication on the specified channel from an RDS for MySQL DB instance and stops replication at the specified global transaction identifier (GTID).

Syntax

```
CALL mysql.rds_start_replication_until_gtid_for_channel(gtid,channel_name);
```

Parameters

gtid

The GTID after which to stop replication.

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_start_replication_until_gtid_for_channel` procedure. The procedure starts replication on the specified channel and applies all changes up to the specified GTID value. Then, it stops replication on the channel.

When the `gtid` parameter specifies a transaction that has already been run by the replica, replication is stopped immediately.

Before you run this procedure, you must disable multi-threaded replication by setting the value of `replica_parallel_workers` or `slave_parallel_workers` to `0`.

Examples

The following example initiates replication on `channel_1`, and replicates changes until it reaches GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
call mysql.rds_start_replication_until_gtid_for_channel('3E11FA47-71CA-11E1-9E33-
C80AA9429562:23','channel_1');
```

mysql.rds_stop_replication_for_channel

Stops replication from a MySQL DB instance on the specified channel.

Syntax

```
CALL mysql.rds_stop_replication_for_channel(channel_name);
```

Parameters

channel_name

The name of the replication channel on the multi-source replica. Each replication channel receives the binary log events from a single source RDS for MySQL DB instance running on a specific host and port.

Usage notes

The master user must run the `mysql.rds_stop_replication_for_channel` procedure.

Examples

The following example stops replication on `channel_1` of the multi-source replica.

```
CALL mysql.rds_stop_replication_for_channel('channel_1');
```

Replicating transactions using GTIDs

The following stored procedures control how transactions are replicated using global transaction identifiers (GTIDs) with RDS for MySQL. For more information about replication based on GTIDs with RDS for MySQL, see Using GTID-based replication.

When using stored procedures to manage replication with a replication user configured with `caching_sha2_password`, you must configure TLS by specifying `SOURCE_SSL=1`. `caching_sha2_password` is the default authentication plugin for RDS for MySQL 8.4.

Topics

- mysql.rds_skip_transaction_with_gtid
- mysql.rds_start_replication_until_gtid

mysql.rds_skip_transaction_with_gtid

Skips replication of a transaction with the specified global transaction identifier (GTID) on a MySQL DB instance.

You can use this procedure for disaster recovery when a specific GTID transaction is known to cause a problem. Use this stored procedure to skip the problematic transaction. Examples of problematic transactions include transactions that disable replication, delete important data, or cause the DB instance to become unavailable.

Syntax

```
CALL mysql.rds_skip_transaction_with_gtid (
gtid_to_skip
);
```

Parameters

gtid_to_skip

The GTID of the replication transaction to skip.

Usage notes

The master user must run the `mysql.rds_skip_transaction_with_gtid` procedure.

This procedure is supported for all RDS for MySQL 5.7 versions, all RDS for MySQL 8.0 versions, and all RDS for MySQL 8.4 versions.

Examples

The following example skips replication of the transaction with the GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_start_replication_until_gtid

Initiates replication from an RDS for MySQL DB instance and stops replication immediately after the specified global transaction identifier (GTID).

Syntax

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

Parameters

gtid

The GTID after which replication is to stop.

Usage notes

The master user must run the `mysql.rds_start_replication_until_gtid` procedure.

This procedure is supported for all RDS for MySQL 5.7 versions, all RDS for MySQL 8.0 versions, and all RDS for MySQL 8.4 versions.

You can use this procedure with delayed replication for disaster recovery. If you have delayed replication configured, you can use this procedure to roll forward changes to a delayed read replica to the time just before a disaster. After this procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

You can configure delayed replication using the following stored procedures:

- mysql.rds_set_configuration
- mysql.rds_set_external_master_with_delay (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower)
- mysql.rds_set_external_source_with_delay (RDS for MySQL major versions 8.4 and higher)
- mysql.rds_set_source_delay

When the `gtid` parameter specifies a transaction that has already been run by the replica, replication is stopped immediately.

Examples

The following example initiates replication and replicates changes until it reaches GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

Rotating the query logs

The following stored procedures rotate MySQL logs to backup tables. For more information, see MySQL database log files.

Topics

- mysql.rds_rotate_general_log
- mysql.rds_rotate_slow_log

mysql.rds_rotate_general_log

Rotates the `mysql.general_log` table to a backup table.

Syntax

```
CALL mysql.rds_rotate_general_log;
```

Usage notes

You can rotate the `mysql.general_log` table to a backup table by calling the `mysql.rds_rotate_general_log` procedure. When log tables are rotated, the current log table is copied to a backup log table and the entries in the current log table are removed. If a backup log table already exists, then it is deleted before the current log table is copied to the backup. You can query the backup log table if needed. The backup log table for the `mysql.general_log` table is named `mysql.general_log_backup`.

You can run this procedure only when the `log_output` parameter is set to `TABLE`.

mysql.rds_rotate_slow_log

Rotates the `mysql.slow_log` table to a backup table.

Syntax

```
CALL mysql.rds_rotate_slow_log;
```

Usage notes

You can rotate the `mysql.slow_log` table to a backup table by calling the `mysql.rds_rotate_slow_log` procedure. When log tables are rotated, the current log table is copied to a backup log table and the entries in the current log table are removed. If a backup log table already exists, then it is deleted before the current log table is copied to the backup.

You can query the backup log table if needed. The backup log table for the `mysql.slow_log` table is named `mysql.slow_log_backup`.

Setting and showing binary log configuration

The following stored procedures set and show configuration parameters, such as for binary log file retention.

Topics

- mysql.rds_set_configuration
- mysql.rds_show_configuration

mysql.rds_set_configuration

Specifies the number of hours to retain binary logs or the number of seconds to delay replication.

Syntax

```
CALL mysql.rds_set_configuration(name,value);
```

Parameters

name

The name of the configuration parameter to set.

value

The value of the configuration parameter.

Usage notes

The `mysql.rds_set_configuration` procedure supports the following configuration parameters:

- binlog retention hours
- source delay
- target delay

The configuration parameters are stored permanently and survive any DB instance reboot or failover.

binlog retention hours

The `binlog retention hours` parameter is used to specify the number of hours to retain binary log files. Amazon RDS normally purges a binary log as soon as possible, but the binary log might still be required for replication with a MySQL database external to RDS.

The default value of `binlog retention hours` is `NULL`. For RDS for MySQL, `NULL` means binary logs aren't retained (0 hours).

To specify the number of hours to retain binary logs on a DB instance, use the `mysql.rds_set_configuration` stored procedure and specify a period with enough time for replication to occur, as shown in the following example.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **Note**
>
> You can't use the value `0` for `binlog retention hours`.

For MySQL DB instances, the maximum `binlog retention hours` value is 168 (7 days).

After you set the retention period, monitor storage usage for the DB instance to make sure that the retained binary logs don't take up too much storage.

For Multi-AZ DB cluster deployments, you can only configure binary log retention from the writer DB instance, and the setting is propagated to all reader DB instances asynchronously. If binary logs on the DB cluster exceed half of the total local storage space, Amazon RDS automatically moves stale logs to the EBS volume. However, the newest logs remain in local storage, so they're subject to be lost if there's a failure that requires a host replacement, or if you scale the database up or down.

source delay

Use the `source delay` parameter in a read replica to specify the number of seconds to delay replication from the read replica to its source DB instance. Amazon RDS normally replicates changes as soon as possible, but you might want some environments to delay replication. For example, when replication is delayed, you can roll forward a delayed read replica to the time just before a disaster. If a table is dropped accidentally, you can use delayed replication to quickly recover it. The default value of `target delay` is `0` (don't delay replication).

When you use this parameter, it runs mysql.rds_set_source_delay and applies CHANGE primary TO MASTER_DELAY = input value. If successful, the procedure saves the `source delay` parameter to the `mysql.rds_configuration` table.

To specify the number of seconds for Amazon RDS to delay replication to a source DB instance, use the `mysql.rds_set_configuration` stored procedure and specify the number of seconds to delay replication. In the following example, the replication is delayed by at least one hour (3,600 seconds).

```
call mysql.rds_set_configuration('source delay', 3600);
```

The procedure then runs `mysql.rds_set_source_delay(3600)`.

The limit for the `source delay` parameter is one day (86400 seconds).

target delay

Use the `target delay` parameter to specify the number of seconds to delay replication between a DB instance and any future RDS-managed read replicas created from this instance. This parameter is ignored for non-RDS-managed read replicas. Amazon RDS normally replicates changes as soon as possible, but you might want some environments to delay replication. For example, when replication is delayed, you can roll forward a delayed read replica to the time just before a disaster. If a table is dropped accidentally, you can use delayed replication to recover it quickly. The default value of `target delay` is `0` (don't delay replication).

For disaster recovery, you can use this configuration parameter with the mysql.rds_start_replication_until stored procedure or the mysql.rds_start_replication_until_gtid stored procedure. To roll forward changes to a delayed read replica to the time just before a disaster, you can run the `mysql.rds_set_configuration` procedure with this parameter set. After the `mysql.rds_start_replication_until` or `mysql.rds_start_replication_until_gtid` procedure stops replication, you can promote the read replica to be the new primary DB instance by using the instructions in Promoting a read replica to be a standalone DB instance.

To use the `mysql.rds_rds_start_replication_until_gtid` procedure, GTID-based replication must be enabled. To skip a specific GTID-based transaction that is known to cause disaster, you can use the mysql.rds_skip_transaction_with_gtid stored procedure. For more information about working with GTID-based replication, see Using GTID-based replication.

To specify the number of seconds for Amazon RDS to delay replication to a read replica, use the `mysql.rds_set_configuration` stored procedure and specify the number of seconds to delay

replication. The following example specifies that replication is delayed by at least one hour (3,600 seconds).

```
call mysql.rds_set_configuration('target delay', 3600);
```

The limit for the `target delay` parameter is one day (86400 seconds).

mysql.rds_show_configuration

The number of hours that binary logs are retained.

Syntax

```
CALL mysql.rds_show_configuration;
```

Usage notes

To verify the number of hours that Amazon RDS retains binary logs, use the `mysql.rds_show_configuration` stored procedure.

Examples

The following example displays the retention period:

```
call mysql.rds_show_configuration;
                name                          value     description
                binlog retention hours        24        binlog retention hours specifies
 the duration in hours before binary logs are automatically deleted.
```

Warming the InnoDB cache

The following stored procedures save, load, or cancel loading the InnoDB buffer pool on RDS for MySQL DB instances. For more information, see InnoDB cache warming for MySQL on Amazon RDS.

Topics

- mysql.rds_innodb_buffer_pool_dump_now
- mysql.rds_innodb_buffer_pool_load_abort
- mysql.rds_innodb_buffer_pool_load_now

mysql.rds_innodb_buffer_pool_dump_now

Dumps the current state of the buffer pool to disk.

Syntax

```
CALL mysql.rds_innodb_buffer_pool_dump_now();
```

Usage notes

The master user must run the `mysql.rds_innodb_buffer_pool_dump_now` procedure.

mysql.rds_innodb_buffer_pool_load_abort

Cancels a load of the saved buffer pool state while in progress.

Syntax

```
CALL mysql.rds_innodb_buffer_pool_load_abort();
```

Usage notes

The master user must run the `mysql.rds_innodb_buffer_pool_load_abort` procedure.

mysql.rds_innodb_buffer_pool_load_now

Loads the saved state of the buffer pool from disk.

Syntax

```
CALL mysql.rds_innodb_buffer_pool_load_now();
```

Usage notes

The master user must run the `mysql.rds_innodb_buffer_pool_load_now` procedure.

Amazon RDS for Oracle

Amazon RDS supports DB instances that run the following versions and editions of Oracle Database:

- Oracle Database 21c (21.0.0.0)
- Oracle Database 19c (19.0.0.0)

> **Note**
>
> Oracle Database 11g, Oracle Database 12c, and Oracle Database 18c are legacy versions that are no longer supported in Amazon RDS.

Before creating a DB instance, complete the steps in the Setting up your Amazon RDS environment section of this guide. When you create a DB instance using your master account, the account gets DBA privileges, with some limitations. Use this account for administrative tasks such as creating additional database accounts. You can't use SYS, SYSTEM, or other Oracle-supplied administrative accounts.

You can create the following:

- DB instances
- DB snapshots
- Point-in-time restores
- Automated backups
- Manual backups

You can use DB instances running Oracle Database inside a VPC. You can also add features to your DB instance by enabling various options, such as Oracle Spatial or Oracle Statspack. Amazon RDS supports Multi-AZ deployments for Oracle as a high-availability, failover solution.

> **Important**
>
> To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. It also restricts access to certain system procedures and tables that need

advanced privileges. You can access your database using standard SQL clients such as Oracle SQL*Plus. However, you can't access the host directly by using Telnet or Secure Shell (SSH).

Topics

- Overview of Oracle on Amazon RDS
- Connecting to your Oracle DB instance
- Securing Oracle DB instance connections
- Working with CDBs in RDS for Oracle
- Administering your RDS for Oracle DB instance
- Working with storage in RDS for Oracle
- Configuring advanced RDS for Oracle features
- Importing data into Oracle on Amazon RDS
- Working with read replicas for Amazon RDS for Oracle
- Adding options to Oracle DB instances
- Upgrading the RDS for Oracle DB engine
- Using third-party software with your RDS for Oracle DB instance
- Oracle Database engine release notes

Overview of Oracle on Amazon RDS

You can read the following sections to get an overview of RDS for Oracle.

Topics

- RDS for Oracle features
- RDS for Oracle releases
- RDS for Oracle licensing options
- RDS for Oracle users and privileges
- RDS for Oracle DB instance classes
- RDS for Oracle database architecture
- RDS for Oracle initialization parameters

- RDS for Oracle character sets
- RDS for Oracle limitations

RDS for Oracle features

Amazon RDS for Oracle supports most of the features and capabilities of Oracle Database. Some features might have limited support or restricted privileges. Some features are only available in Enterprise Edition, and some require additional licenses. For more information about Oracle Database features for specific Oracle Database versions, see the *Oracle Database Licensing Information User Manual* for the version you're using.

Topics

- New features in RDS for Oracle
- Supported features in RDS for Oracle
- Unsupported features in RDS for Oracle

New features in RDS for Oracle

To see new features in RDS for Oracle, search Document history for the keyword **`Oracle`**.

Supported features in RDS for Oracle

Amazon RDS for Oracle supports the following Oracle Database features:

> **Note**
>
> The following list isn't exhaustive.

- Advanced Compression
- Oracle Application Express (APEX)

  For more information, see Oracle Application Express (APEX).
- Automatic Memory Management
- Automatic Undo Management
- Automatic Workload Repository (AWR)

For more information, see Generating performance reports with Automatic Workload Repository (AWR).

- Active Data Guard with Maximum Performance in the same AWS Region or across AWS Regions

  For more information, see Working with read replicas for Amazon RDS for Oracle.
- Blockchain tables (Oracle Database 21c and higher)

  For more information, see Managing Blockchain Tables in the Oracle Database documentation.
- Continuous Query Notification

  For more information, see Using Continuous Query Notification (CQN) in the Oracle documentation.
- Data Redaction
- Continuous Query Notification

  For more information, see Database Change Notification in the Oracle documentation.
- Database In-Memory
- Distributed Queries and Transactions
- Edition-Based Redefinition

  For more information, see Setting the default edition for a DB instance.
- EM Express (12c and higher)

  For more information, see Oracle Enterprise Manager.
- Fine-Grained Auditing
- Flashback Table, Flashback Query, Flashback Transaction Query
- Gradual password rollover for applications (Oracle Database 21c and higher)

  For more information, see Managing Gradual Database Password Rollover for Applications in the Oracle Database documentation.
- HugePages

  For more information, see Turning on HugePages for an RDS for Oracle instance.
- Import/export (legacy and Data Pump) and SQL*Loader

  For more information, see Importing data into Oracle on Amazon RDS.

- Java Virtual Machine (JVM)

  For more information, see Oracle Java virtual machine.
- JavaScript (Oracle Database 21c and higher)

  For more information, see DBMS_MLE in the Oracle Database documentation.
- Label Security

  For more information, see Oracle Label Security.
- Locator

  For more information, see Oracle Locator.
- Materialized Views
- Multitenant

  The Oracle multitenant architecture is supported for all Oracle Database 19c and higher releases. For more information, see Working with CDBs in RDS for Oracle.
- Network encryption

  For more information, see Oracle native network encryption and Oracle Secure Sockets Layer.
- Partitioning
- Real Application Testing

  To use the full capture and replay capabilities, you must use Amazon Elastic File System (Amazon EFS) to access files generated by Oracle Real Application Testing. For more information, see Amazon EFS integration and the blog post Use Oracle Real Application Testing features with Amazon RDS for Oracle.
- Sharding at the application level (but not the Oracle Sharding feature)
- Spatial and Graph

  For more information, see Oracle Spatial.
- Star Query Optimization
- Streams and Advanced Queuing
- Summary Management – Materialized View Query Rewrite
- Text (File and URL data store types are not supported)
- Total Recall

- Transparent Data Encryption (TDE)

  For more information, see Oracle Transparent Data Encryption.
- Unified Auditing, Mixed Mode

  For more information, see Mixed mode auditing in the Oracle documentation.
- XML DB (without the XML DB Protocol Server)

  For more information, see Oracle XML DB.
- Virtual Private Database

Unsupported features in RDS for Oracle

Amazon RDS for Oracle doesn't support the following Oracle Database features:

> **Note**
>
> The following list isn't exhaustive.

- Automatic Storage Management (ASM)
- Database Vault
- Flashback Database

> **Note**
>
> For alternative solutions, see the AWS Database Blog entry Alternatives to the Oracle flashback database feature in Amazon RDS for Oracle.

- FTP and SFTP
- Hybrid partitioned tables
- Messaging Gateway
- Oracle Enterprise Manager Cloud Control Management Repository
- Real Application Clusters (Oracle RAC)
- Real Application Security (RAS)
- Unified Auditing, Pure Mode

- Workspace Manager (WMSYS) schema

> **Warning**
>
> In general, Amazon RDS doesn't prevent you from creating schemas for unsupported features. However, if you create schemas for Oracle features and components that require SYSDBA privileges, you can damage the data dictionary and affect the availability of your DB instance. Use only supported features and schemas that are available in Adding options to Oracle DB instances.

RDS for Oracle releases

RDS for Oracle for Oracle supports multiple Oracle Database releases.

> **Note**
>
> For information about upgrading your releases, see Upgrading the RDS for Oracle DB engine.

Topics

- Oracle Database 21c with Amazon RDS
- Oracle Database 19c with Amazon RDS

Oracle Database 21c with Amazon RDS

Amazon RDS supports Oracle Database 21c, which includes Oracle Enterprise Edition and Oracle Standard Edition 2. Oracle Database 21c (21.0.0.0) includes many new features and updates from the previous version. A key change is that Oracle Database 21c supports only the multitenant architecture: you can no longer create a database as a traditional non-CDB. To learn more about the differences between CDBs and non-CDBs, see Limitations of RDS for Oracle CDBs.

In this section, you can find the features and changes important to using Oracle Database 21c (21.0.0.0) on Amazon RDS. For a complete list of the changes, see the Oracle database 21c documentation. For a complete list of features supported by each Oracle Database 21c edition,

see Permitted features, options, and management packs by Oracle database offering in the Oracle documentation.

Amazon RDS parameter changes for Oracle Database 21c (21.0.0.0)

Oracle Database 21c (21.0.0.0) includes several new parameters and parameters with new ranges and new default values.

Topics

- New parameters
- Changes for the compatible parameter
- Removed parameters

New parameters

The following table shows the new Amazon RDS parameters for Oracle Database 21c (21.0.0.0).

Name	Range of values	Default value	Modifiable	Description
blockchain_table_max_no_drop	NONE | 0	NONE	Y	Lets you control the maximum amount of idle time that can be specified when creating a blockchain table.
dbnest_enable	NONE | CDB_RESOURCE_PDB_ALL	NONE	N	Allows you to enable or disable dbNest. DbNest provides operating system resource isolation and management, file system isolation, and secure computing for PDBs.
dbnest_pdb_fs_conf	NONE | *pathname*	NONE	N	Specifies the dbNest file system configuration file for a PDB.

Name	Range of values	Default value	Modifiable	Description
diagnostics_control	ERROR | WARNING | IGNORE	IGNORE	Y	Allows you to control and monitor the users who perform potentially unsafe database diagnostic operations.
drcp_dedicated_opt	YES | NO	YES	Y	Enables or disables the use of dedicated optimization with Database Resident Connection Pooling (DRCP).
enable_per_pdb_drcp	true | false	true	N	Controls whether Database Resident Connection Pooling (DRCP) configures one connection pool for the entire CDB or one isolated connection pool for each PDB.
inmemory_deep_vectorization	true | false	true	Y	Enables or disables the deep vectorization framework.
mandatory_user_profile	*profile_name*	N/A	N	Specifies the mandatory user profile for a CDB or PDB.
optimizer_capture_sql_quarantine	true | false	false	Y	Enables or disables the deep vectorization framework.
optimizer_use_sql_quarantine	true | false	false	Y	Enables or disables the automatic creation of SQL Quarantine configurations.

Name	Range of values	Default value	Modifiable	Description
result_cache_execution_threshold	0 to 2147483647	2	Y	Specifies the maximum number of times a PL/SQL function can be executed before its result is stored in the result cache.
result_cache_max_temp_result	0 to 100	5	Y	Specifies the percentage of RESULT_CACHE_MAX_TEMP_SIZE that any single cached query result can consume.
result_cache_max_temp_size	0 to 2199023255552	RESULT_CACHE_SIZE * 10	Y	Specifies the maximum amount of temporary tablespace (in bytes) that can be consumed by the result cache.
sga_min_size	0 to 2199023255552 (maximum value is 50% of sga_target)	0	Y	Indicates a possible minimum value for SGA usage of a pluggable database (PDB).

Name	Range of values	Default value	Modifiable	Description
tablespace_encryption_default_algorithm	GOST256 | SEED128 | ARIA256 | ARIA192 | ARIA128 | 3DES168 | AES256 | AES192 | AES128	AES128	Y	Specifies the default algorithm the database uses when encrypting a tablespace.

Changes for the compatible parameter

The `compatible` parameter has a new maximum value for Oracle Database 21c (21.0.0.0) on Amazon RDS. The following table shows the new default value.

Parameter name	Oracle Database 21c (21.0.0.0) maximum value
compatible	21.0.0

Removed parameters

The following parameters were removed in Oracle Database 21c (21.0.0.0):

- `remote_os_authent`
- `sec_case_sensitive_logon`
- `unified_audit_sga_queue_size`

Oracle Database 19c with Amazon RDS

Amazon RDS supports Oracle Database 19c, which includes Oracle Enterprise Edition and Oracle Standard Edition Two.

Oracle Database 19c (19.0.0.0) includes many new features and updates from the previous version. In this section, you can find the features and changes important to using Oracle Database 19c (19.0.0.0) on Amazon RDS. For a complete list of the changes, see the Oracle database 19c documentation. For a complete list of features supported by each Oracle Database 19c edition, see Permitted features, options, and management packs by Oracle database offering in the Oracle documentation.

Amazon RDS parameter changes for Oracle Database 19c (19.0.0.0)

Oracle Database 19c (19.0.0.0) includes several new parameters and parameters with new ranges and new default values.

Topics

- New parameters
- Changes to the compatible parameter
- Removed parameters

New parameters

The following table shows the new Amazon RDS parameters for Oracle Database 19c (19.0.0.0).

Name	Values	Modifiable	Description
lob_signature_enable	TRUE, FALSE (default)	Y	Enables or disables the LOB locator signature feature.
max_datapump_parallel_per_job	1 to 1024, or AUTO	Y	Specifies the maximum number of parallel processes allowed for each Oracle Data Pump job.

Changes to the compatible parameter

The `compatible` parameter has a new maximum value for Oracle Database 19c (19.0.0.0) on Amazon RDS. The following table shows the new default value.

Parameter name	Oracle Database 19c (19.0.0.0) maximum value
compatible	19.0.0

Removed parameters

The following parameters were removed in Oracle Database 19c (19.0.0.0):

- `exafusion_enabled`
- `max_connections`
- `o7_dictionary_access`

RDS for Oracle licensing options

Amazon RDS for Oracle has two licensing options: License Included (LI) and Bring Your Own License (BYOL). After you create an Oracle DB instance on Amazon RDS, you can change the licensing model by modifying the DB instance. For more information, see Modifying an Amazon RDS DB instance.

> **Important**
>
> Make sure that you have the appropriate Oracle Database license, with Software Update License and Support, for your DB instance class and Oracle Database edition. Also make sure that you have licenses for any separately licensed Oracle Database features.

Topics

- License Included model for SE2
- Bring Your Own License (BYOL) for EE and SE2
- Licensing Oracle Multi-AZ deployments

License Included model for SE2

In the License Included model, you don't need to purchase Oracle Database licenses separately. AWS holds the license for the Oracle database software. The License Included model is only supported on Amazon RDS for Oracle Database Standard Edition 2 (SE2).

In this model, if you have an AWS Support account with case support, contact Support for both Amazon RDS and Oracle Database service requests. Your use of RDS for Oracle the LI option is subject to Section 10.3.1 of the AWS Service Terms.

Bring Your Own License (BYOL) for EE and SE2

In the BYOL model, you can use your existing Oracle Database licenses to deploy databases on Amazon RDS. Amazon RDS supports the BYOL model only for Oracle Database Enterprise Edition (EE) and Oracle Database Standard Edition 2 (SE2).

Make sure that you have the appropriate Oracle Database license (with Software Update License and Support) for the DB instance class and Oracle Database edition you wish to run. You must also follow Oracle's policies for licensing Oracle Database software in the cloud computing environment. For more information on Oracle's licensing policy for Amazon EC2, see Licensing Oracle software in the cloud computing environment.

In this model, you continue to use your active Oracle support account, and you contact Oracle directly for Oracle Database service requests. If you have an AWS Support account with case support, you can contact Support for Amazon RDS issues.

Integrating with AWS License Manager

To make it easier to monitor Oracle license usage in the BYOL model, AWS License Manager integrates with Amazon RDS for Oracle. License Manager supports tracking of RDS for Oracle engine editions and licensing packs based on virtual cores (vCPUs). You can also use License Manager with AWS Organizations to manage all of your organizational accounts centrally.

The following table shows the product information filters for RDS for Oracle.

Filter	Name	Description
Engine Edition	`oracle-ee`	Oracle Database Enterprise Edition (EE)
	`oracle-se2`	Oracle Database Standard Edition 2 (SE2)
License Pack	`data guard`	See Working with read replicas for Amazon RDS for Oracle (Oracle Active Data Guard)
	`olap`	See Oracle OLAP
	`ols`	See Oracle Label Security

Filter	Name	Description
	`diagnostic pack sqlt`	See Oracle SQLT
	`tuning pack sqlt`	See Oracle SQLT

To track license usage of your Oracle DB instances, you can create a self-managed license using AWS License Manager. In this case, RDS for Oracle resources that match the product information filter are automatically associated with the self-managed license. Discovery of Oracle DB instances can take up to 24 hours. You can also track a license across accounts by using AWS Resource Access Manager.

Console

To create a self-managed license in AWS License Manager to track the license usage of your RDS for Oracle DB instances

1. Go to https://console.aws.amazon.com/license-manager/.
2. Choose **Create self-managed license**.

   For instructions, see Create a self-managed license in the *AWS License Manager User Guide*.

   Add a rule for an **RDS Product Information Filter** in the **Product Information** panel.

   For more information, see ProductInformation in the *AWS License Manager API Reference*.
3. (Cross-account tracking only) Use AWS Resource Access Manager to share your self-managed licenses with any AWS account or through AWS Organizations. For more information, see Sharing your AWS resources.

AWS CLI

To create a self-managed license by using the AWS CLI, call the create-license-configuration command. Use the `--cli-input-json` or `--cli-input-yaml` parameters to pass the parameters to the command.

Example

The following example creates a self-managed license for Oracle Enterprise Edition.

```
aws license-manager create-license-configuration --cli-input-json file://rds-oracle-
ee.json
```

The following is the sample `rds-oracle-ee.json` file used in the example.

```
{
    "Name": "rds-oracle-ee",
    "Description": "RDS Oracle Enterprise Edition",
    "LicenseCountingType": "vCPU",
    "LicenseCountHardLimit": false,
    "ProductInformationList": [
        {
            "ResourceType": "RDS",
            "ProductInformationFilterList": [
                {
                    "ProductInformationFilterName": "Engine Edition",
                    "ProductInformationFilterValue": ["oracle-ee"],
                    "ProductInformationFilterComparator": "EQUALS"
                }
            ]
        }
    ]
}
```

For more information about product information, see Automated discovery of resource inventory in the *AWS License Manager User Guide*.

For more information about the `--cli-input` parameter, see Generating AWS CLI skeleton and input parameters from a JSON or YAML input file in the *AWS CLI User Guide*.

Migrating between Oracle Database editions

If you have an unused BYOL Oracle Database license appropriate for the edition and class of DB instance that you plan to run, you can migrate from Standard Edition 2 (SE2) to Enterprise Edition (EE). You can't migrate from EE to other editions.

To change your Oracle Database edition and retain your data

1. Create a snapshot of the DB instance.

   For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

2. Restore the snapshot to a new DB instance, and select the Oracle database edition you want to use.

   For more information, see Restoring to a DB instance.

3. (Optional) Delete the old DB instance, unless you want to keep it running and have the appropriate Oracle Database licenses for it.

   For more information, see Deleting a DB instance.

Licensing Oracle Multi-AZ deployments

Amazon RDS supports Multi-AZ deployments for Oracle as a high-availability, failover solution. We recommend Multi-AZ for production workloads. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

If you use the Bring Your Own License model, you must have a license for both the primary DB instance and the standby DB instance in a Multi-AZ deployment.

RDS for Oracle users and privileges

When you create an Amazon RDS for Oracle DB instance, the default master user has most of the maximum user permissions on the DB instance. Use the master user account for any administrative tasks, such as creating additional user accounts in your database. Because RDS is a managed service, you aren't allowed to log in as SYS and SYSTEM, and thus don't have SYSDBA privileges.

Topics

- Limitations for Oracle DBA privileges
- How to manage privileges on SYS objects

Limitations for Oracle DBA privileges

In the database, a *role* is a collection of privileges that you can grant to or revoke from a user. An Oracle database uses roles to provide security. For more information, see Configuring Privilege and Role Authorization in the Oracle Database documentation.

The predefined role DBA normally allows all administrative privileges on an Oracle database. When you create a DB instance, your master user account gets DBA privileges (with some limitations). To

deliver a managed experience, an RDS for Oracle database doesn't provide the following privileges for the DBA role:

- `ALTER DATABASE`
- `ALTER SYSTEM`
- `CREATE ANY DIRECTORY`
- `DROP ANY DIRECTORY`
- `GRANT ANY PRIVILEGE`
- `GRANT ANY ROLE`

For more RDS for Oracle system privilege and role information, see Master user account privileges.

How to manage privileges on SYS objects

You can manage privileges on SYS objects by using the `rdsadmin.rdsadmin_util` package. For example, if you create the database user `myuser`, you could use the `rdsadmin.rdsadmin_util.grant_sys_object` procedure to grant `SELECT` privileges on `V_$SQLAREA` to `myuser`. For more information, see the following topics:

- Granting SELECT or EXECUTE privileges to SYS objects
- Revoking SELECT or EXECUTE privileges on SYS objects
- Granting privileges to non-master users

RDS for Oracle DB instance classes

The computation and memory capacity of an RDS for Oracle DB instance is determined by its instance class. The DB instance class you need depends on your processing power and memory requirements.

Supported RDS for Oracle DB instance classes

The supported RDS for Oracle instance classes are a subset of the RDS DB instance classes. For the complete list of RDS instance classes, see DB instance classes.

RDS for Oracle preconfigured DB instance classes

RDS for Oracle also offers instance classes that are preconfigured for workloads that require additional memory, storage, and I/O per vCPU. These instance classes use the following naming convention.

```
db.r5b.instance_size.tpcthreads_per_core.memratio
db.r5.instance_size.tpcthreads_per_core.memratio
```

The following is an example of an instance class that is preconfigured for additional memory:

```
db.r5b.4xlarge.tpc2.mem2x
```

The components of the preceding instance class name are as follows:

- `db.r5b.4xlarge` – The name of the instance class.
- `tpc2` – The threads per core. A value of 2 means that multithreading is turned on. A value of 1 means that multithreading is turned off.
- `mem2x` – The ratio of additional memory to the standard memory for the instance class. In this example, the optimization provides twice as much memory as a standard db.r5.4xlarge DB instance.

> **Note**
>
> For the normalization factors of the preconfigured RDS for Oracle DB instance classes, see Hardware specifications for DB instance classes.

Supported edition, instance class, and licensing combinations in RDS for Oracle

If you're using the RDS console, you can find out whether a specific edition, instance class, and license combination is supported by choosing **Create database** and specifying different option. In the AWS CLI, you can run the following command:

```
aws rds describe-orderable-db-instance-options --engine engine-type --license-
model license-type
```

The following table lists all editions, instance classes, and license types supported for RDS for Oracle. For information about the memory attributes of each type, see RDS for Oracle instance types. For information about pricing, see Amazon RDS for Oracle pricing models.

Oracle edition	Oracle Database 19c and higher
Enterprise Edition (EE)	**Standard instance classes**
Bring Your Own License (BYOL)	db.m7i.large–db.m7i.48xlarge, db.m7i.metal-24xl, db.m7i.metal-48xl
	db.m6in.large–db.m6in.32xlarge, db.m6in.metal
	db.m6id.large–db.m6id.32xlarge, db.m6id.metal
	db.m6i.large–db.m6i.32xlarge, db.m6i.metal
	db.m5d.large–db.m5d.24xlarge
	db.m5.large–db.m5.24xlarge
	Memory optimized instance classes
	db.r7i.large–db.r7i.48xlarge, db.r7i.metal-24xl, db.r7i.metal-48xl
	db.r6in.large–db.r6in.32xlarge, db.r6in.metal
	db.r6id.large–db.r6id.32xlarge, db.r6id.metal
	db.r6i.large–db.r6i.32xlarge, db.r6i.metal
	db.r5d.large–db.r5d.24xlarge
	db.r5b.large–db.r5b.24xlarge
	db.r5.large–db.r5.24xlarge
	db.x2iedn.xlarge–db.x2iedn.32xlarge, db.x2iedn.metal
	db.x2iezn.2xlarge–db.x2iezn.12xlarge, db.x2iezn.metal
	db.x2idn.16xlarge–db.x2idn.32xlarge, db.x2idn.metal

Oracle edition	Oracle Database 19c and higher
	db.z1d.large–db.z1d.12xlarge
	Memory optimized preconfigured instance classes

Oracle edition	Oracle Database 19c and higher
	db.r7i.8xlarge.tpc2.mem3x
	db.r7i.8xlarge.tpc2.mem2x
	db.r7i.6xlarge.tpc2.mem4x
	db.r7i.6xlarge.tpc2.mem2x
	db.r7i.4xlarge.tpc2.mem4x
	db.r7i.4xlarge.tpc2.mem3x
	db.r7i.4xlarge.tpc2.mem2x
	db.r7i.3xlarge.tpc2.mem4x
	db.r7i.2xlarge.tpc2.mem8x
	db.r7i.2xlarge.tpc2.mem4x
	db.r7i.xlarge.tpc2.mem4x
	db.r7i.xlarge.tpc2.mem2x
	db.r6i.8xlarge.tpc2.mem4x
	db.r6i.8xlarge.tpc2.mem3x
	db.r6i.6xlarge.tpc2.mem4x
	db.r6i.4xlarge.tpc2.mem4x
	db.r6i.4xlarge.tpc2.mem3x
	db.r6i.4xlarge.tpc2.mem2x
	db.r6i.2xlarge.tpc2.mem8x
	db.r6i.2xlarge.tpc2.mem4x
	db.r6i.2xlarge.tpc1.mem2x

Oracle edition	Oracle Database 19c and higher
	db.r6i.xlarge.tpc2.mem4x
	db.r6i.xlarge.tpc2.mem2x
	db.r6i.large.tpc1.mem2x
	db.r5b.8xlarge.tpc2.mem3x
	db.r5b.6xlarge.tpc2.mem4x
	db.r5b.4xlarge.tpc2.mem4x
	db.r5b.4xlarge.tpc2.mem3x
	db.r5b.4xlarge.tpc2.mem2x
	db.r5b.2xlarge.tpc2.mem8x
	db.r5b.2xlarge.tpc2.mem4x
	db.r5b.2xlarge.tpc1.mem2x
	db.r5b.xlarge.tpc2.mem4x
	db.r5b.xlarge.tpc2.mem2x
	db.r5b.large.tpc1.mem2x
	db.r5.12xlarge.tpc2.mem2x
	db.r5.8xlarge.tpc2.mem3x
	db.r5.6xlarge.tpc2.mem4x
	db.r5.4xlarge.tpc2.mem4x
	db.r5.4xlarge.tpc2.mem3x
	db.r5.4xlarge.tpc2.mem2x
	db.r5.2xlarge.tpc2.mem8x

Oracle edition	Oracle Database 19c and higher
	db.r5.2xlarge.tpc2.mem4x
	db.r5.2xlarge.tpc1.mem2x
	db.r5.xlarge.tpc2.mem4x
	db.r5.xlarge.tpc2.mem2x
	db.r5.large.tpc1.mem2x
	Burstable performance instance classes
	db.t3.small–db.t3.2xlarge
Standard Edition 2 (SE2)	**Standard instance classes**
Bring Your Own License (BYOL)	db.m7i.large–db.m7i.4xlarge, db.m7i.metal-24xl, db.m7i.metal-48xl
	db.m6in.large–db.m6in.4xlarge, db.m6in.metal
	db.m6id.large–db.m6id.4xlarge, db.m6id.metal
	db.m6i.large–db.m6i.4xlarge, db.m6i.metal
	db.m5d.large–db.m5d.4xlarge
	db.m5.large–db.m5.4xlarge
	Memory optimized instance classes

Oracle edition	Oracle Database 19c and higher
	db.r7i.large–db.r7i.4xlarge, db.r7i.metal-24xl, db.r7i.metal-48xl
	db.r6in.large–db.r6in.4xlarge, db.r6in.metal
	db.r6id.large–db.r6id.4xlarge, db.r6id.metal
	db.r6i.large–db.r6i.4xlarge, db.r6i.metal
	db.r5d.large–db.r5d.4xlarge
	db.r5b.large–db.r5b.4xlarge
	db.r5.large–db.r5.4xlarge
	db.x2iedn.xlarge–db.x2iedn.4xlarge, db.x2iedn.metal
	db.x2iezn.2xlarge–db.x2iezn.4xlarge, db.x2iezn.metal
	db.x2idn.metal
	db.z1d.large–db.z1d.3xlarge
	Memory optimized preconfigured instance classes

Oracle edition	Oracle Database 19c and higher
	db.r7i.xlarge.tpc2.mem2x
	db.r7i.xlarge.tpc2.mem4x
	db.r7i.2xlarge.tpc2.mem4x
	db.r7i.2xlarge.tpc2.mem8x
	db.r7i.3xlarge.tpc2.mem4x
	db.r7i.4xlarge.tpc2.mem2x
	db.r7i.4xlarge.tpc2.mem3x
	db.r7i.4xlarge.tpc2.mem4x
	db.r6i.4xlarge.tpc2.mem4x
	db.r6i.4xlarge.tpc2.mem3x
	db.r6i.4xlarge.tpc2.mem2x
	db.r6i.2xlarge.tpc2.mem8x
	db.r6i.2xlarge.tpc2.mem4x
	db.r6i.2xlarge.tpc1.mem2x
	db.r6i.xlarge.tpc2.mem4x
	db.r6i.xlarge.tpc2.mem2x
	db.r6i.large.tpc1.mem2x
	db.r5b.4xlarge.tpc2.mem4x
	db.r5b.4xlarge.tpc2.mem3x
	db.r5b.4xlarge.tpc2.mem2x
	db.r5b.2xlarge.tpc2.mem8x

Oracle edition	Oracle Database 19c and higher
	db.r5b.2xlarge.tpc2.mem4x
	db.r5b.2xlarge.tpc1.mem2x
	db.r5b.xlarge.tpc2.mem4x
	db.r5b.xlarge.tpc2.mem2x
	db.r5b.large.tpc1.mem2x
	db.r5.4xlarge.tpc2.mem4x
	db.r5.4xlarge.tpc2.mem3x
	db.r5.4xlarge.tpc2.mem2x
	db.r5.2xlarge.tpc2.mem8x
	db.r5.2xlarge.tpc2.mem4x
	db.r5.2xlarge.tpc1.mem2x
	db.r5.xlarge.tpc2.mem4x
	db.r5.xlarge.tpc2.mem2x
	db.r5.large.tpc1.mem2x
	Burstable performance instance classes
	db.t3.small–db.t3.2xlarge
Standard Edition 2 (SE2)	**Standard instance classes**
License Included	db.m7i.large–db.m7i.4xlarge
	db.m5.large–db.m5.4xlarge
	Memory optimized instance classes

Oracle edition	Oracle Database 19c and higher
	db.r7i.large–db.r7i.4xlarge
	db.r6i.large–db.r6i.4xlarge
	db.r5.large–db.r5.4xlarge
	Burstable performance instance classes
	db.t3.small–db.t3.2xlarge

Deprecated RDS for Oracle DB instance classes

The following DB instance classes are deprecated for RDS for Oracle:

- db.m1, db.m2, db.m3, db.m4
- db.t1, db.t2
- db.r1, db.r2, db.r3, db.r4
- db.x1, db.x1e

The preceding DB instance classes have been replaced by better performing DB instance classes that are generally available at a lower cost. If you have DB instances that use deprecated DB instance classes, you have the following options:

- Allow Amazon RDS to modify each DB instance automatically to use a comparable non-deprecated DB instance class. For deprecation timelines, see DB instance class types.
- Change the DB instance class yourself by modifying the DB instance. For more information, see Modifying an Amazon RDS DB instance.

> **Note**
>
> If you have DB snapshots of DB instances that were using deprecated DB instance classes, you can choose a DB instance class that is not deprecated when you restore the DB snapshots. For more information, see Restoring to a DB instance.

RDS for Oracle database architecture

The *Oracle multitenant architecture*, also known as the *CDB architecture*, enables an Oracle database to function as a *multitenant container database (CDB)*. A CDB can include customer-created *pluggable databases (PDBs)*. A *non-CDB* is an Oracle database that uses the traditional architecture, which can't contain PDBs. For more information about the multitenant architecture, see *Oracle Multitenant Administrator's Guide*.

For Oracle Database 19c and higher, you can create an RDS for Oracle DB instance that uses the CDB architecture. In RDS for Oracle, PDBs are referred to as *tenant databases*. Your client applications connect at the tenant database (PDB) level rather than the CDB level. RDS for Oracle supports the following configurations of the CDB architecture:

Multi-tenant configuration

Amazon RDS allows a CDB instance to contain between 1–30 tenant databases, depending on the database edition and any required option licenses. You can use RDS APIs to add, modify, and remove tenant databases. The multi-tenant configuration in RDS for Oracle doesn't support application PDBs or proxy PDBs, which are special types of PDBs. For more information about application PDBs and proxy PDBs, see Types of PDBs in the Oracle Database documentation.

> **Note**
>
> The Amazon RDS configuration is called "multi-tenant" rather than "multitenant" because it is a capability of Amazon RDS, not just the Oracle DB engine. Similarly, the RDS term "tenant" refers to any tenant in an RDS configuration, not just Oracle PDBs. In the RDS documentation, the unhyphenated term "Oracle multitenant" refers exclusively to the Oracle database CDB architecture, which is compatible with both on-premises and RDS deployments.

Single-tenant configuration

Amazon RDS limits an RDS for Oracle CDB instance to 1 tenant database (PDB). You can't add more PDBs using RDS APIs. The single-tenant configuration uses the same RDS APIs as the non-CDB architecture. Thus, the experience of working with a CDB in the single-tenant configuration is mostly the same as working with a non-CDB.

You can convert a CDB that uses the single-tenant configuration to the multi-tenant configuration, thus allowing you to add PDBs to your CDB. This architecture change is permanent and irreversible. For more information, see Converting the single-tenant configuration to multi-tenant.

Note

You can't access the CDB itself.

In Oracle Database 21c and higher, all databases are CDBs. In contrast, you can create an Oracle Database 19c DB instance as either a CDB or non-CDB. You can't upgrade a non-CDB to a CDB, but you convert an Oracle Database 19c non-CDB to a CDB, and then upgrade it. You can't convert a CDB to a non-CDB.

For more information, see the following resources:

- Working with CDBs in RDS for Oracle
- Limitations of RDS for Oracle CDBs
- Creating an Amazon RDS DB instance

RDS for Oracle initialization parameters

In Amazon RDS, you manage parameters using a DB parameter group. Using this group, you can customize initialization parameters. For example, you can configure the sort area size with `sort_area_size`. All RDS for Oracle DB instances associated with a specific DB parameter group use the same parameter settings. For more information, see Parameter groups for Amazon RDS.

Supported initialization parameters in RDS for Oracle

Supported parameters for your DB instance depend on your Oracle Database edition and version. To view the supported initialization parameters for a specific Oracle Database edition and version, run the AWS CLI command `describe-engine-default-parameters`. For example, to list names of the supported initialization parameters for the Enterprise Edition of Oracle Database 19c, run the following command (sample output included).

```
aws rds describe-engine-default-parameters \
```

```
    --db-parameter-group-family oracle-ee-19 \
    --output json | jq -r '.EngineDefaults.Parameters[].ParameterName'

_add_col_optim_enabled
_adg_parselock_timeout
_allow_insert_with_update_check
_allow_level_without_connect_by
_always_semi_join
_autotask_max_window
_awr_disabled_flush_tables
_awr_mmon_cpuusage
_awr_mmon_deep_purge_all_expired
_b_tree_bitmap_plans
_bct_bitmaps_per_file
_bloom_filter_enabled
_buffered_publisher_flow_control_threshold
_bug29394014_allow_triggers_on_vpd_table
_cleanup_rollback_entries
_client_enable_auto_unregister
_clusterwide_global_transactions
_complex_view_merging
_connect_by_use_union_all
_cost_equality_semi_join
_cursor_features_enabled
_cursor_obsolete_threshold
_datafile_write_errors_crash_instance
_db_block_buffers
...
```

To describe a single initialization parameter, use the following command, replacing `sga_max_size` with the name of your parameter (sample output included).

```
aws rds describe-engine-default-parameters \
    --db-parameter-group-family oracle-ee-19 \
    --query 'EngineDefaults.Parameters[?ParameterName==`sga_max_size`]' \
    --output json

[
    {
        "ParameterName": "sga_max_size",
        "Description": "max total SGA size",
        "Source": "engine-default",
        "ApplyType": "static",
```

```
        "DataType": "integer",
        "AllowedValues": "0-2199023255552",
        "IsModifiable": true
    }
]
```

To find general documentation for the Oracle database initialization parameters, see Initialization Parameters in the Oracle Database documentation. Note that the parameter ARCHIVE_LAG_TARGET has special considerations in RDS for Oracle. This parameter forces an online redo log switch after the specified time elapses. In RDS for Oracle, ARCHIVE_LAG_TARGET is set to 300 because the recovery point objective (RPO) is 5 minutes. To honor this objective, RDS for Oracle switches the online redo log every 5 minutes and stores it in an Amazon S3 bucket.

If the frequency of the online redo log switch degrades the performance of your RDS for Oracle database, you can scale your DB instance and storage to use higher IOPS and throughput. Alternatively, if you use RDS Custom for Oracle or deploy an Oracle database on Amazon EC2, you can adjust the setting of the ARCHIVE_LAG_TARGET initialization parameter.

Valid parameter values in RDS for Oracle

In RDS for Oracle, only the following characters are valid for parameter values:

- Letters (A-Z and a-z)
- Numbers (0-9)
- Whitespace (spaces, tabs, and line breaks)
- The following special characters: _ / . : + = ( ) ' * , % $ - (hyphen)

RDS for Oracle character sets

RDS for Oracle supports two types of character sets: the DB character set and national character set.

DB character set

The Oracle database character set is used in the CHAR, VARCHAR2, and CLOB data types. The database also uses this character set for metadata such as table names, column names, and SQL statements. The Oracle database character set is typically referred to as the *DB character set*.

You set the character set when you create a DB instance. You can't change the DB character set after you create the database.

Supported DB character sets

The following table lists the Oracle DB character sets that are supported in Amazon RDS. You can use a value from this table with the `--character-set-name` parameter of the AWS CLI create-db-instance command or with the `CharacterSetName` parameter of the Amazon RDS API CreateDBInstance operation.

> **Note**
>
> The character set for a CDB is always AL32UTF8. You can set a different character set for the PDB only.

Value	Description
AL32UTF8	Unicode 5.0 UTF-8 Universal character set (default)
AR8ISO8859P6	ISO 8859-6 Latin/Arabic
AR8MSWIN1256	Microsoft Windows Code Page 1256 8-bit Latin/Arabic
BLT8ISO8859P13	ISO 8859-13 Baltic
BLT8MSWIN1257	Microsoft Windows Code Page 1257 8-bit Baltic
CL8ISO8859P5	ISO 88559-5 Latin/Cyrillic
CL8MSWIN1251	Microsoft Windows Code Page 1251 8-bit Latin/Cyrillic
EE8ISO8859P2	ISO 8859-2 East European
EL8ISO8859P7	ISO 8859-7 Latin/Greek

Value	Description
EE8MSWIN1250	Microsoft Windows Code Page 1250 8-bit East European
EL8MSWIN1253	Microsoft Windows Code Page 1253 8-bit Latin/Greek
IW8ISO8859P8	ISO 8859-8 Latin/Hebrew
IW8MSWIN1255	Microsoft Windows Code Page 1255 8-bit Latin/Hebrew
JA16EUC	EUC 24-bit Japanese
JA16EUCTILDE	Same as JA16EUC except for mapping of wave dash and tilde to and from Unicode
JA16SJIS	Shift-JIS 16-bit Japanese
JA16SJISTILDE	Same as JA16SJIS except for mapping of wave dash and tilde to and from Unicode
KO16MSWIN949	Microsoft Windows Code Page 949 Korean
NE8ISO8859P10	ISO 8859-10 North European
NEE8ISO8859P4	ISO 8859-4 North and Northeast European
TH8TISASCII	Thai Industrial Standard 620-2533-ASCII 8-bit
TR8MSWIN1254	Microsoft Windows Code Page 1254 8-bit Turkish
US7ASCII	ASCII 7-bit American
UTF8	Unicode 3.0 UTF-8 Universal character set, CESU-8 compliant
VN8MSWIN1258	Microsoft Windows Code Page 1258 8-bit Vietnamese

Value	Description
WE8ISO8859P1	Western European 8-bit ISO 8859 Part 1
WE8ISO8859P15	ISO 8859-15 West European
WE8ISO8859P9	ISO 8859-9 West European and Turkish
WE8MSWIN1252	Microsoft Windows Code Page 1252 8-bit West European
ZHS16GBK	GBK 16-bit Simplified Chinese
ZHT16HKSCS	Microsoft Windows Code Page 950 with Hong Kong Supplementary Character Set HKSCS-2001. Character set conversion is based on Unicode 3.0.
ZHT16MSWIN950	Microsoft Windows Code Page 950 Traditional Chinese
ZHT32EUC	EUC 32-bit Traditional Chinese

NLS_LANG environment variable

A *locale* is a set of information addressing linguistic and cultural requirements that corresponds to a given language and country. Setting the NLS_LANG environment variable in your client's environment is the simplest way to specify locale behavior for Oracle. This variable sets the language and territory used by the client application and the database server. It also indicates the client's character set, which corresponds to the character set for data entered or displayed by a client application. For more information on NLS_LANG and character sets, see What is a character set or code page? in the Oracle documentation.

NLS initialization parameters

You can also set the following National Language Support (NLS) initialization parameters at the instance level for an Oracle DB instance in Amazon RDS:

- NLS_COMP

- NLS_DATE_FORMAT
- NLS_LENGTH_SEMANTICS
- NLS_NCHAR_CONV_EXCP
- NLS_SORT
- NLS_TIME_FORMAT
- NLS_TIME_TZ_FORMAT
- NLS_TIMESTAMP_FORMAT
- NLS_TIMESTAMP_TZ_FORMAT

For information about modifying instance parameters, see Parameter groups for Amazon RDS.

You can set other NLS initialization parameters in your SQL client. For example, the following statement sets the NLS_LANGUAGE initialization parameter to GERMAN in a SQL client that is connected to an Oracle DB instance:

```
ALTER SESSION SET NLS_LANGUAGE=GERMAN;
```

For information about connecting to an Oracle DB instance with a SQL client, see Connecting to your Oracle DB instance.

National character set

The national character set is used in the NCHAR, NVARCHAR2, and NCLOB data types. The national character set is typically referred to as the *NCHAR character set*. Unlike the DB character set, the NCHAR character set doesn't affect database metadata.

The NCHAR character set supports the following character sets:

- AL16UTF16 (default)
- UTF8

You can specify either value with the `--nchar-character-set-name` parameter of the create-db-instance command (AWS CLI version 2 only). If you use the Amazon RDS API, specify the `NcharCharacterSetName` parameter of CreateDBInstance operation. You can't change the national character set after you create the database.

For more information about Unicode in Oracle databases, see Supporting multilingual databases with unicode in the Oracle documentation.

RDS for Oracle limitations

In the following sections, you can find important limitations of using RDS for Oracle. For limitations specific to CDBs, see Limitations of RDS for Oracle CDBs.

> **Note**
>
> This list is not exhaustive.

Topics

- Oracle file size limits in Amazon RDS
- Block size limits in RDS for Oracle
- Public synonyms for Oracle-supplied schemas
- Schemas for unsupported features in RDS for Oracle
- Limitations for DBA privileges in RDS for Oracle
- Deprecation of TLS 1.0 and 1.1 Transport Layer Security in RDS for Oracle

Oracle file size limits in Amazon RDS

The maximum size of a single file on RDS for Oracle DB instances is 16 TiB (tebibytes). This limit is imposed by the ext4 filesystem used by the instance. Thus, Oracle bigfile data files are limited to 16 TiB. If you try to resize a data file in a bigfile tablespace to a value over the limit, you receive an error such as the following.

```
ORA-01237: cannot extend datafile 6
ORA-01110: data file 6: '/rdsdbdata/db/mydir/datafile/myfile.dbf'
ORA-27059: could not reduce file size
Linux-x86_64 Error: 27: File too large
Additional information: 2
```

Block size limits in RDS for Oracle

RDS for Oracle DB instances are created with a default database block size (`DB_BLOCK_SIZE`) of 8 KB. The default database block size is set at database creation and cannot be changed. The `SYSTEM`

and temporary tablespaces always use the default database block size. You can create additional tablespaces with non-default block sizes by configuring the corresponding DB_nK_CACHE_SIZE parameter (for example, DB_16K_CACHE_SIZE) to allocate a buffer cache for that block size, and then specifying the BLOCKSIZE clause in your CREATE TABLESPACE statement.

Public synonyms for Oracle-supplied schemas

Don't create or modify public synonyms for Oracle-supplied schemas, including SYS, SYSTEM, and RDSADMIN. Such actions might result in invalidation of core database components and affect the availability of your DB instance.

You can create public synonyms referencing objects in your own schemas.

Schemas for unsupported features in RDS for Oracle

In general, Amazon RDS doesn't prevent you from creating schemas for unsupported features. However, if you create schemas for Oracle features and components that require SYS privileges, you can damage the data dictionary and affect your instance availability. Use only supported features and schemas that are available in Adding options to Oracle DB instances.

Limitations for DBA privileges in RDS for Oracle

In the database, a *role* is a collection of privileges that you can grant to or revoke from a user. An Oracle database uses roles to provide security.

The predefined role DBA normally allows all administrative privileges on an Oracle database. When you create a DB instance, your master user account gets DBA privileges (with some limitations). To deliver a managed experience, an RDS for Oracle database doesn't provide the following privileges for the DBA role:

- ALTER DATABASE
- ALTER SYSTEM
- CREATE ANY DIRECTORY
- DROP ANY DIRECTORY
- GRANT ANY PRIVILEGE
- GRANT ANY ROLE

Use the master user account for administrative tasks such as creating additional user accounts in the database. You can't use SYS, SYSTEM, and other Oracle-supplied administrative accounts.

Deprecation of TLS 1.0 and 1.1 Transport Layer Security in RDS for Oracle

Transport Layer Security protocol versions 1.0 and 1.1 (TLS 1.0 and TLS 1.1) are deprecated. In accordance with security best practices, Oracle has deprecated the use of TLS 1.0 and TLS 1.1. To meet your security requirements, we strongly recommends that you use TLS 1.2 instead.

Connecting to your Oracle DB instance

After Amazon RDS provisions your Oracle DB instance, you can use any standard SQL client application to log in to your DB instance. Because RDS is a managed service, you can't log in as SYS or SYSTEM. For more information, see RDS for Oracle users and privileges.

In this topic, you learn how to use Oracle SQL Developer or SQL*Plus to connect to an RDS for Oracle DB instance. For an example that walks you through the process of creating and connecting to a sample DB instance, see Creating and connecting to an Oracle DB instance.

Topics

- Finding the endpoint of your RDS for Oracle DB instance
- Connecting to your DB instance using Oracle SQL developer
- Connecting to your DB instance using SQL*Plus
- Considerations for security groups
- Considerations for process architecture
- Troubleshooting connections to your Oracle DB instance
- Modifying connection properties using sqlnet.ora parameters

Finding the endpoint of your RDS for Oracle DB instance

Each Amazon RDS DB instance has an endpoint, and each endpoint has the DNS name and port number for the DB instance. To connect to your DB instance using a SQL client application, you need the DNS name and port number for your DB instance.

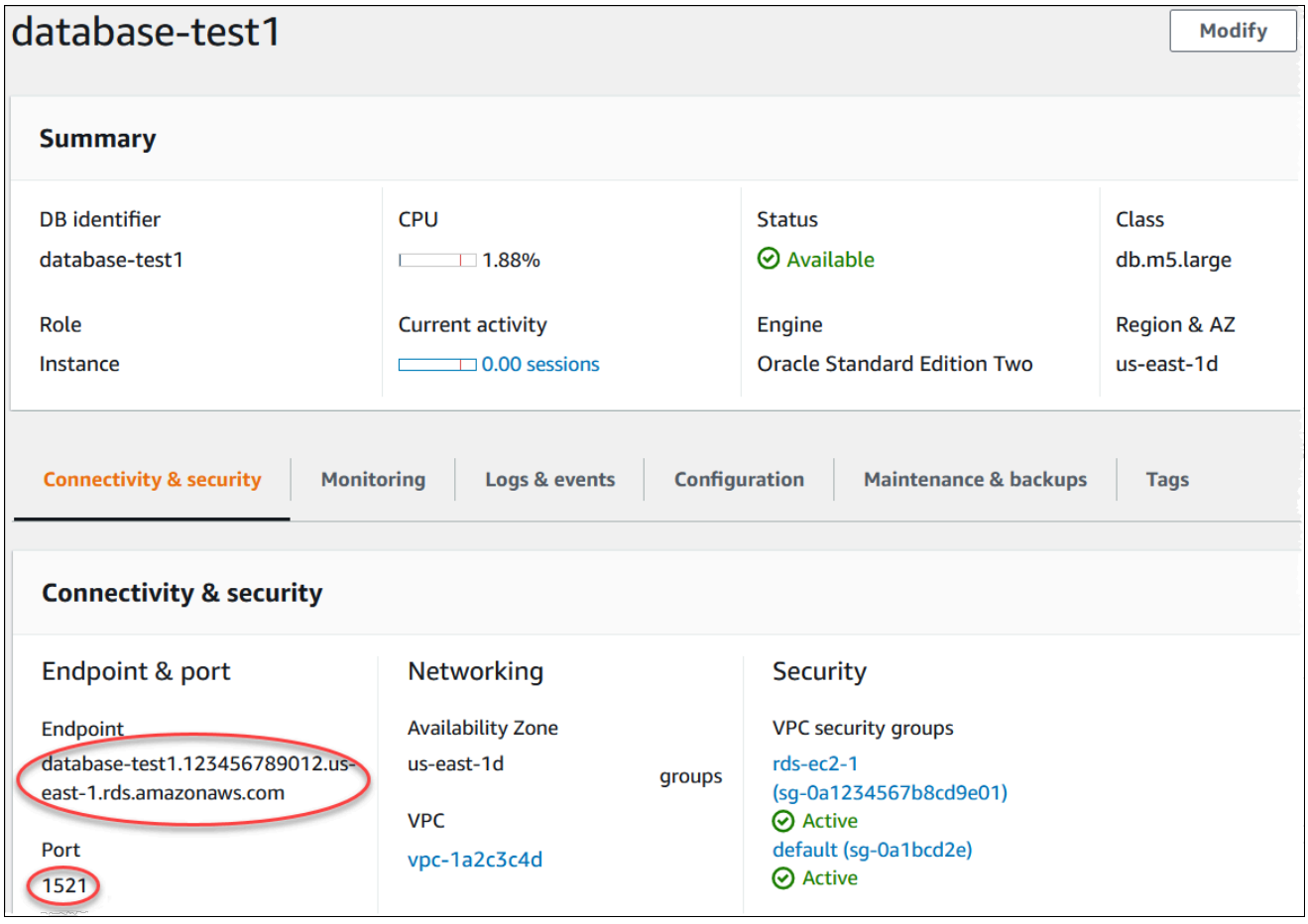
You can find the endpoint for a DB instance using the Amazon RDS console or the AWS CLI.

> **Note**
>
> If you are using Kerberos authentication, see Connecting to Oracle with Kerberos authentication.

Console

To find the endpoint using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the console, choose the AWS Region of your DB instance.
3. Find the DNS name and port number for your DB instance.
   a. Choose **Databases** to display a list of your DB instances.
   b. Choose the Oracle DB instance name to display the instance details.
   c. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



AWS CLI

To find the endpoint of an Oracle DB instance by using the AWS CLI, call the describe-db-instances command.

Example To find the endpoint using the AWS CLI

```
aws rds describe-db-instances
```

Search for `Endpoint` in the output to find the DNS name and port number for your DB instance. The `Address` line in the output contains the DNS name. The following is an example of the JSON endpoint output.

```
"Endpoint": {
    "HostedZoneId": "Z1PVIF0B656C1W",
    "Port": 3306,
    "Address": "myinstance.123456789012.us-west-2.rds.amazonaws.com"
},
```

> **Note**
>
> The output might contain information for multiple DB instances.

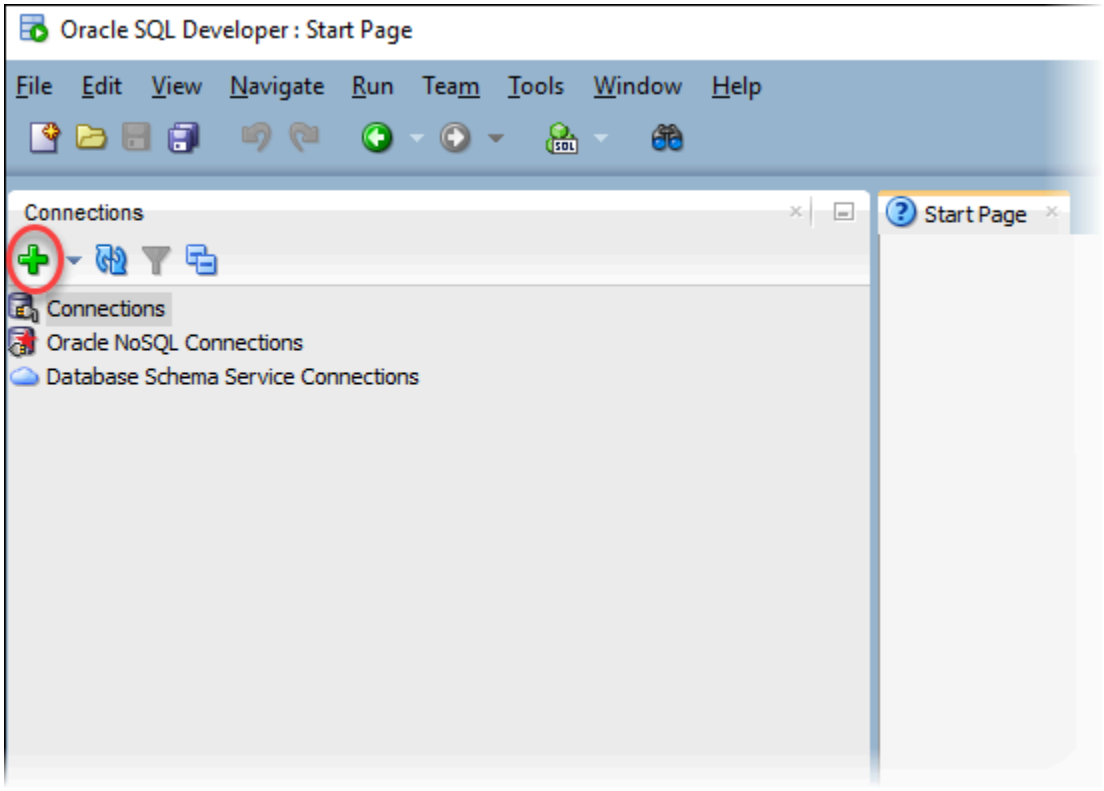
Connecting to your DB instance using Oracle SQL developer

In this procedure, you connect to your DB instance by using Oracle SQL Developer. To download a standalone version of this utility, see the Oracle SQL developer downloads page.

To connect to your DB instance, you need its DNS name and port number. For information about finding the DNS name and port number for a DB instance, see Finding the endpoint of your RDS for Oracle DB instance.

To connect to a DB instance using SQL developer

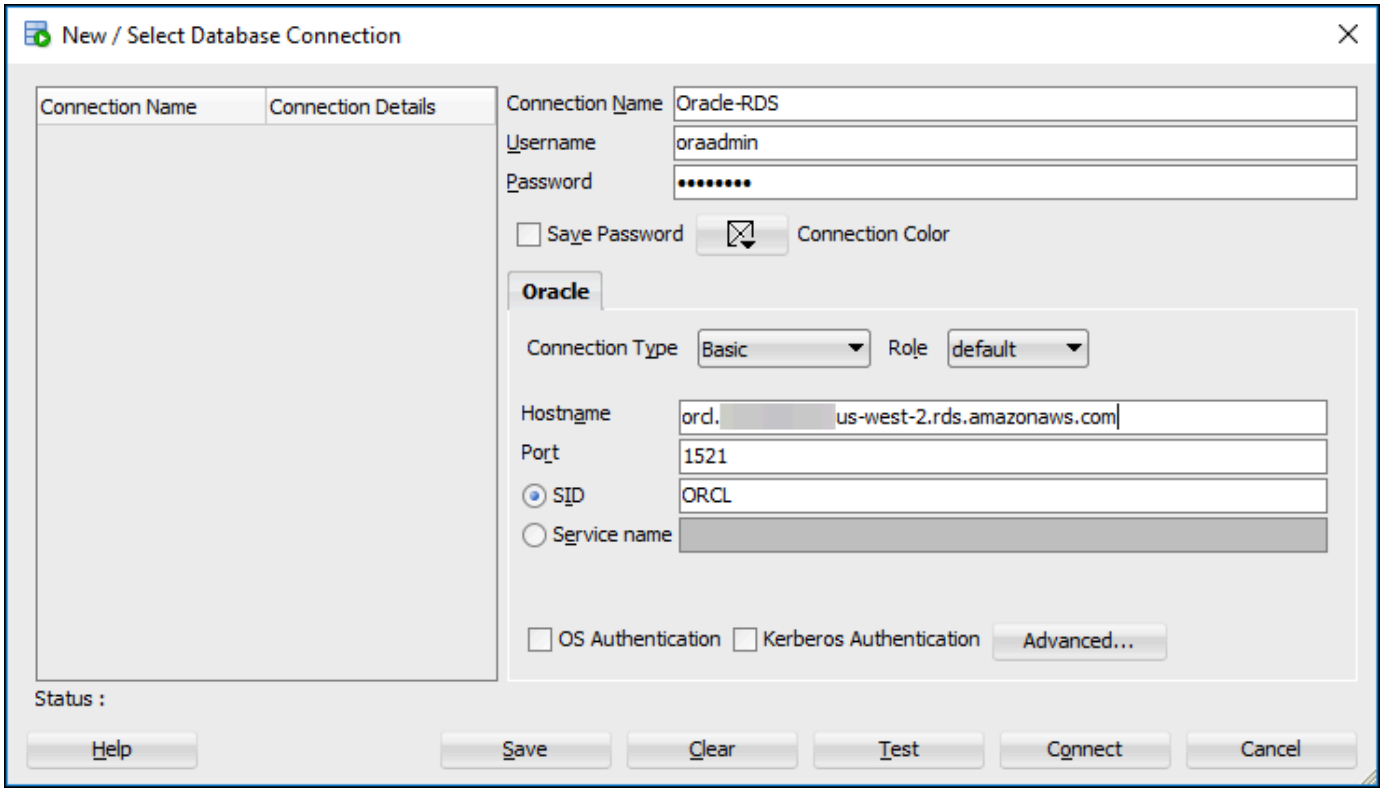
1. Start Oracle SQL Developer.
2. On the **Connections** tab, choose the **add (+)** icon.



3. In the **New/Select Database Connection** dialog box, provide the information for your DB instance:

   - For **Connection Name**, enter a name that describes the connection, such as `Oracle-RDS`.
   - For **Username**, enter the name of the database administrator for the DB instance.
   - For **Password**, enter the password for the database administrator.
   - For **Hostname**, enter the DNS name of the DB instance.
   - For **Port**, enter the port number.
   - For **SID**, enter the DB name. You can find the DB name on the **Configuration** tab of your database details page.

   The completed dialog box should look similar to the following.

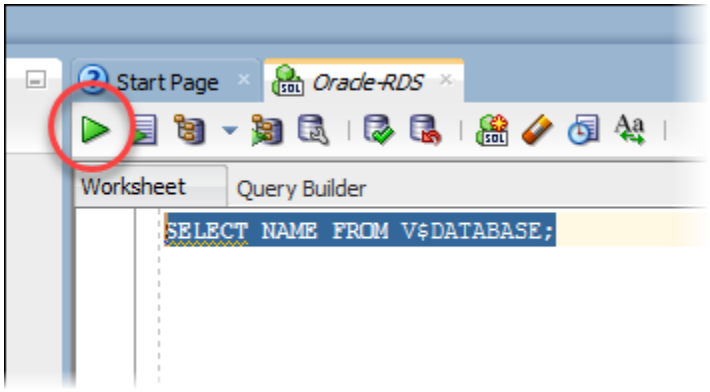


4. Choose **Connect**.
5. You can now start creating your own databases and running queries against your DB instance and databases as usual. To run a test query against your DB instance, do the following:

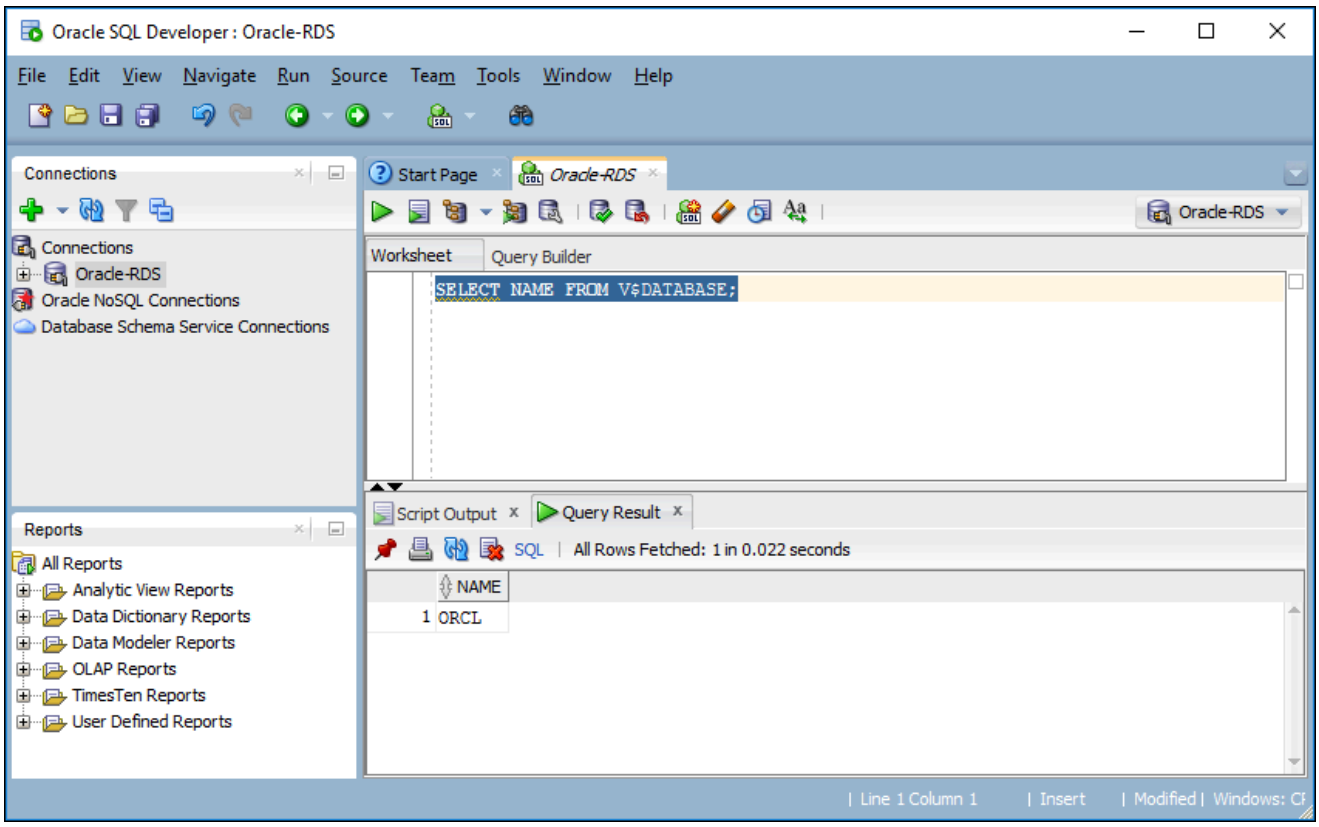
   a. In the **Worksheet** tab for your connection, enter the following SQL query.

   ```
   SELECT NAME FROM V$DATABASE;
   ```

   b. Choose the **execute** icon to run the query.

   SQL Developer returns the database name.



Connecting to your DB instance using SQL*Plus

You can use a utility like SQL*Plus to connect to an Amazon RDS DB instance running Oracle. To download Oracle Instant Client, which includes a standalone version of SQL*Plus, see Oracle Instant Client Downloads.

To connect to your DB instance, you need its DNS name and port number. For information about finding the DNS name and port number for a DB instance, see Finding the endpoint of your RDS for Oracle DB instance.

Example To connect to an Oracle DB instance using SQL*Plus

In the following examples, substitute the user name of your DB instance administrator. Also, substitute the DNS name for your DB instance, and then include the port number and the Oracle SID. The SID value is the name of the DB instance's database that you specified when you created the DB instance, and not the name of the DB instance.

For Linux, macOS, or Unix:

```
sqlplus 'user_name@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=dns_name)(PORT=port))
(CONNECT_DATA=(SID=database_name)))'
```

For Windows:

```
sqlplus user_name@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=dns_name)(PORT=port))
(CONNECT_DATA=(SID=database_name)))
```

You should see output similar to the following.

```
SQL*Plus: Release 12.1.0.2.0 Production on Mon Aug 21 09:42:20 2017
```

After you enter the password for the user, the SQL prompt appears.

```
SQL>
```

> **Note**
>
> The shorter format connection string (EZ Connect), such as `sqlplus USER/PASSWORD@longer-than-63-chars-rds-endpoint-here:1521/database-identifier`, might encounter a maximum character limit, so you we recommend that you don't use it to connect.

Considerations for security groups

For you to connect to your DB instance, it must be associated with a security group that contains the necessary IP addresses and network configuration. Your DB instance might use the default security group. If you assigned a default, nonconfigured security group when you created the DB instance, the firewall prevents connections. For information about creating a new security group, see Controlling access with security groups.

After you create the new security group, you modify your DB instance to associate it with the security group. For more information, see Modifying an Amazon RDS DB instance.

You can enhance security by using SSL to encrypt connections to your DB instance. For more information, see Oracle Secure Sockets Layer.

Considerations for process architecture

Server processes handle user connections to an Oracle DB instance. By default, the Oracle DB instance uses dedicated server processes. With dedicated server processes, each server process services only one user process. You can optionally configure shared server processes. With shared server processes, each server process can service multiple user processes.

You might consider using shared server processes when a high number of user sessions are using too much memory on the server. You might also consider shared server processes when sessions connect and disconnect very often, resulting in performance issues. There are also disadvantages to using shared server processes. For example, they can strain CPU resources, and they are more complicated to configure and administer.

For more information about dedicated and shared server processes, see About dedicated and shared server processes in the Oracle documentation. For more information about configuring shared server processes on an RDS for Oracle DB instance, see How do I configure Amazon RDS for Oracle database to work with shared servers? in the Knowledge Center.

Troubleshooting connections to your Oracle DB instance

The following are issues you might encounter when you try to connect to your Oracle DB instance.

Issue	Troubleshooting suggestions
Unable to connect to your DB instance.	For a newly created DB instance, the DB instance has a status of **creating** until it is ready to use. When the state changes to **available**, you can connect to the DB instance. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new DB instance is available.
Unable to connect to your DB instance.	If you can't send or receive communications over the port that you specified when you created the DB instance, you can't connect to the DB instance. Check with your network administrator to verify that the port you specified for your DB instance allows inbound and outbound communication.
Unable to connect to your DB instance.	The access rules enforced by your local firewall and the IP addresses you authorized to access your DB instance in the

Issue	Troubleshooting suggestions
	security group for the DB instance might not match. The problem is most likely the inbound or outbound rules on your firewall. You can add or edit an inbound rule in the security group. For **Source**, choose **My IP**. This allows access to the DB instance from the IP address detected in your browser. For more information, see Amazon VPC and Amazon RDS. For more information about security groups, see Controlling access with security groups. To walk through the process of setting up rules for your security group, see Tutorial: Create a VPC for use with a DB instance (IPv4 only).
Connect failed because target host or object does not exist – Oracle, Error: ORA-12545	Make sure that you specified the server name and port number correctly. For **Server name**, enter the DNS name from the console. For information about finding the DNS name and port number for a DB instance, see Finding the endpoint of your RDS for Oracle DB instance.
Invalid username/ password; logon denied – Oracle, Error: ORA-01017	You were able to reach the DB instance, but the connection was refused. This is usually caused by providing an incorrect user name or password. Verify the user name and password, and then retry.
TNS:listener does not currently know of SID given in connect descriptor - Oracle, ERROR: ORA-12505	Ensure the correct SID is entered. The SID is the same as your DB name. Find the DB name in the **Configuration** tab of the **Databases** page for your instance. You can also find the DB name using the AWS CLI: `aws rds describe-db-instances --query 'DBInstances[*].[DBInstanceIdentifier,DBName]' --output text`

For more information on connection issues, see Can't connect to Amazon RDS DB instance.

Modifying connection properties using sqlnet.ora parameters

The sqlnet.ora file includes parameters that configure Oracle Net features on Oracle database servers and clients. Using the parameters in the sqlnet.ora file, you can modify properties for connections in and out of the database.

For more information about why you might set sqlnet.ora parameters, see Configuring profile parameters in the Oracle documentation.

Setting sqlnet.ora parameters

Amazon RDS for Oracle parameter groups include a subset of sqlnet.ora parameters. You set them in the same way that you set other Oracle parameters. The `sqlnetora.` prefix identifies which parameters are sqlnet.ora parameters. For example, in an Oracle parameter group in Amazon RDS, the `default_sdu_size` sqlnet.ora parameter is `sqlnetora.default_sdu_size`.

For information about managing parameter groups and setting parameter values, see Parameter groups for Amazon RDS.

Supported sqlnet.ora parameters

Amazon RDS supports the following sqlnet.ora parameters. Changes to dynamic sqlnet.ora parameters take effect immediately.

Parameter	Valid values	Static/ Dynamic	Description
`sqlnetora.default_sdu_size`	`512` to `20971`	Dynamic	The session data unit (SDU) size, in bytes. The SDU is the amount of data that is put in a buffer and sent across the network at one time.
`sqlnetora.diag_adr_enabled`	`ON`, `OFF`	Dynamic	A value that enables or disables Automatic Diagnostic Repository (ADR) tracing.

Parameter	Valid values	Static/ Dynamic	Description
			ON specifies that ADR file tracing is used. OFF specifies that non-ADR file tracing is used.
sqlnetora.recv_buf_size	8192 to 26843	Dynamic	The buffer space limit for receive operations of sessions, supported by the TCP/IP, TCP/IP with SSL, and SDP protocols.
sqlnetora.send_buf_size	8192 to 26843	Dynamic	The buffer space limit for send operations of sessions, supported by the TCP/IP, TCP/IP with SSL, and SDP protocols.
sqlnetora.sqlnet.allowed_logon_version_client	8, 10, 11, 12	Dynamic	Minimum authentication protocol version allowed for clients, and servers acting as clients, to establish a connection to Oracle DB instances.
sqlnetora.sqlnet.allowed_logon_version_server	8, 9, 10, 11, 12, 12a	Dynamic	Minimum authentication protocol version allowed to establish a connection to Oracle DB instances.

Parameter	Valid values	Static/ Dynamic	Description
sqlnetora.sqlnet.expire_time	0 to 1440	Dynamic	Time interval, in minutes, to send a check to verify that client-server connections are active.
sqlnetora.sqlnet.inbound_connect_timeout	0 or 10 to 7200	Dynamic	Time, in seconds, for a client to connect with the database server and provide the necessary authentication information.
sqlnetora.sqlnet.outbound_connect_timeout	0 or 10 to 7200	Dynamic	Time, in seconds, for a client to establish an Oracle Net connection to the DB instance.
sqlnetora.sqlnet.recv_timeout	0 or 10 to 7200	Dynamic	Time, in seconds, for a database server to wait for client data after establishing a connection.
sqlnetora.sqlnet.send_timeout	0 or 10 to 7200	Dynamic	Time, in seconds, for a database server to complete a send operation to clients after establishing a connection.
sqlnetora.tcp.connect_timeout	0 or 10 to 7200	Dynamic	Time, in seconds, for a client to establish a TCP connection to the database server.

Parameter	Valid values	Static/ Dynamic	Description
sqlnetora.trace_level_server	0, 4, 10, 16, OFF, USER, ADMIN, SUPPOF	Dynamic	For non-ADR tracing, turns server tracing on at a specified level or turns it off.

The default value for each supported sqlnet.ora parameter is the Oracle Database default for the release.

Viewing sqlnet.ora parameters

You can view sqlnet.ora parameters and their settings using the AWS Management Console, the AWS CLI, or a SQL client.

Viewing sqlnet.ora parameters using the console

For information about viewing parameters in a parameter group, see Parameter groups for Amazon RDS.

In Oracle parameter groups, the `sqlnetora.` prefix identifies which parameters are sqlnet.ora parameters.

Viewing sqlnet.ora parameters using the AWS CLI

To view the sqlnet.ora parameters that were configured in an Oracle parameter group, use the AWS CLI describe-db-parameters command.

To view the all of the sqlnet.ora parameters for an Oracle DB instance, call the AWS CLI download-db-log-file-portion command. Specify the DB instance identifier, the log file name, and the type of output.

Example

The following code lists all of the sqlnet.ora parameters for `mydbinstance`.

For Linux, macOS, or Unix:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier mydbinstance \
    --log-file-name trace/sqlnet-parameters \
    --output text
```

For Windows:

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier mydbinstance ^
    --log-file-name trace/sqlnet-parameters ^
    --output text
```

Viewing sqlnet.ora parameters using a SQL client

After you connect to the Oracle DB instance in a SQL client, the following query lists the sqlnet.ora parameters.

```
SELECT * FROM TABLE
   (rdsadmin.rds_file_util.read_text_file(
        p_directory => 'BDUMP',
        p_filename  => 'sqlnet-parameters'));
```

For information about connecting to an Oracle DB instance in a SQL client, see Connecting to your Oracle DB instance.

Securing Oracle DB instance connections

Amazon RDS for Oracle supports SSL/TLS encrypted connections and also the Oracle Native Network Encryption (NNE) option to encrypt connections between your application and your Oracle DB instance. For more information about the Oracle Native Network Encryption option, see Oracle native network encryption.

Topics

- Using SSL with an RDS for Oracle DB instance
- Updating applications to connect to Oracle DB instances using new SSL/TLS certificates
- Using native network encryption with an RDS for Oracle DB instance
- Configuring Kerberos authentication for Amazon RDS for Oracle
- Configuring UTL_HTTP access using certificates and an Oracle wallet

Using SSL with an RDS for Oracle DB instance

Secure Sockets Layer (SSL) is an industry-standard protocol for securing network connections between client and server. After SSL version 3.0, the name was changed to Transport Layer Security (TLS), but we still often refer to the protocol as SSL. Amazon RDS supports SSL encryption for Oracle DB instances. Using SSL, you can encrypt a connection between your application client and your Oracle DB instance. SSL support is available in all AWS Regions for Oracle.

To enable SSL encryption for an Oracle DB instance, add the Oracle SSL option to the option group associated with the DB instance. Amazon RDS uses a second port, as required by Oracle, for SSL connections. Doing this allows both clear text and SSL-encrypted communication to occur at the same time between a DB instance and an Oracle client. For example, you can use the port with clear text communication to communicate with other resources inside a VPC while using the port with SSL-encrypted communication to communicate with resources outside the VPC.

For more information, see Oracle Secure Sockets Layer.

> **Note**
>
> You can't use both SSL and Oracle native network encryption (NNE) on the same DB instance. Before you can use SSL encryption, you must disable any other connection encryption.

Updating applications to connect to Oracle DB instances using new SSL/TLS certificates

As of January 13, 2023, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your RDS DB instances using Secure Socket Layer or Transport Layer Security (SSL/TLS). Following, you can find information about updating your applications to use the new certificates.

This topic can help you to determine whether any client applications use SSL/TLS to connect to your DB instances.

Important

When you change the certificate for an Amazon RDS for Oracle DB instance, only the database listener is restarted. The DB instance isn't restarted. Existing database connections are unaffected, but new connections will encounter errors for a brief period while the listener is restarted.
We recommend that you restart your Oracle DB to prevent any connection errors.

Note

For client applications that use SSL/TLS to connect to your DB instances, you must update your client application trust stores to include the new CA certificates.

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB instances. We strongly recommend testing these procedures in a development or staging environment before implementing them in your production environments.

For more information about certificate rotation, see Rotating your SSL/TLS certificate. For more information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . For information about using SSL/TLS with Oracle DB instances, see Oracle Secure Sockets Layer.

Topics

- Finding out whether applications connect using SSL

- Updating your application trust store
- Example Java code for establishing SSL connections

Finding out whether applications connect using SSL

If your Oracle DB instance uses an option group with the SSL option added, you might be using SSL. Check this by following the instructions in Listing the options and option settings for an option group. For information about the SSL option, see Oracle Secure Sockets Layer.

Check the listener log to determine whether there are SSL connections. The following is sample output in a listener log.

```
date time * (CONNECT_DATA=(CID=(PROGRAM=program)
(HOST=host)(USER=user))(SID=sid)) *
(ADDRESS=(PROTOCOL=tcps)(HOST=host)(PORT=port)) * establish * ORCL * 0
```

When PROTOCOL has the value `tcps` for an entry, it shows an SSL connection. However, when HOST is `127.0.0.1`, you can ignore the entry. Connections from `127.0.0.1` are a local management agent on the DB instance. These connections aren't external SSL connections. Therefore, you have applications connecting using SSL if you see listener log entries where PROTOCOL is `tcps` and HOST is *not* `127.0.0.1`.

To check the listener log, you can publish the log to Amazon CloudWatch Logs. For more information, see Publishing Oracle logs to Amazon CloudWatch Logs.

Updating your application trust store

You can update the trust store for applications that use SQL*Plus or JDBC for SSL/TLS connections.

Updating your application trust store for SQL*Plus

You can update the trust store for applications that use SQL*Plus for SSL/TLS connections.

> **Note**
>
> When you update the trust store, you can retain older certificates in addition to adding the new certificates.

To update the trust store for SQL*Plus applications

1. Download the new root certificate that works for all AWS Regions and put the file in the `ssl_wallet` directory.

   For information about downloading the root certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

2. Run the following command to update the Oracle wallet.

   ```
   prompt>orapki wallet add -wallet $ORACLE_HOME/ssl_wallet -trusted_cert -cert
         $ORACLE_HOME/ssl_wallet/ssl-cert.pem -auto_login_only
   ```

   Replace the file name with the one that you downloaded.

3. Run the following command to confirm that the wallet was updated successfully.

   ```
   prompt>orapki wallet display -wallet $ORACLE_HOME/ssl_wallet
   ```

   Your output should contain the following.

   ```
   Trusted Certificates:
   Subject: CN=Amazon RDS Root 2019 CA,OU=Amazon RDS,O=Amazon Web Services\,
    Inc.,L=Seattle,ST=Washington,C=US
   ```

Updating your application trust store for JDBC

You can update the trust store for applications that use JDBC for SSL/TLS connections.

For information about downloading the root certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

For sample scripts that import certificates, see Sample script for importing certificates into your trust store.

Example Java code for establishing SSL connections

The following code example shows how to set up the SSL connection using JDBC.

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class OracleSslConnectionTest {
    private static final String DB_SERVER_NAME = "<dns-name-provided-by-amazon-rds>";
    private static final Integer SSL_PORT = "<ssl-option-port-configured-in-option-
group>";
    private static final String DB_SID = "<oracle-sid>";
    private static final String DB_USER = "<user name>";
    private static final String DB_PASSWORD = "<password>";
    // This key store has only the prod root ca.
    private static final String KEY_STORE_FILE_PATH = "<file-path-to-keystore>";
    private static final String KEY_STORE_PASS = "<keystore-password>";

    public static void main(String[] args) throws SQLException {
        final Properties properties = new Properties();
        final String connectionString = String.format(
                "jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS)(HOST=%s)(PORT=
%d))(CONNECT_DATA=(SID=%s)))",
                DB_SERVER_NAME, SSL_PORT, DB_SID);
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);
        properties.put("oracle.jdbc.J2EE13Compliant", "true");
        properties.put("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        properties.put("javax.net.ssl.trustStoreType", "JKS");
        properties.put("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);
        final Connection connection = DriverManager.getConnection(connectionString,
 properties);
        // If no exception, that means handshake has passed, and an SSL connection can
 be opened
    }
}
```

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-rsa2048-g1 certificates. For instructions, see step 3 in Updating your CA certificate by modifying your DB instance or cluster.

Using native network encryption with an RDS for Oracle DB instance

Oracle Database offers two ways to encrypt data over the network: native network encryption (NNE) and Transport Layer Security (TLS). NNE is a proprietary Oracle security feature, whereas TLS is an industry standard. RDS for Oracle supports NNE for all editions of Oracle Database.

NNE has the following advantages over TLS:

- You can control NNE on the client and server using settings in the NNE option:
  - `SQLNET.ALLOW_WEAK_CRYPTO_CLIENTS` and `SQLNET.ALLOW_WEAK_CRYPTO`
  - `SQLNET.CRYPTO_CHECKSUM_CLIENT` and `SQLNET.CRYPTO_CHECKSUM_SERVER`
  - `SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT` and `SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER`
  - `SQLNET.ENCRYPTION_CLIENT` and `SQLNET.ENCRYPTION_SERVER`
  - `SQLNET.ENCRYPTION_TYPES_CLIENT` and `SQLNET.ENCRYPTION_TYPES_SERVER`
- In most cases, you don't need to configure your client or server. In contrast, TLS requires you to configure both client and server.
- No certificates are required. In TLS, the server requires a certificate (which eventually expires), and the client requires a trusted root certificate for the certificate authority that issued the server's certificate.

To enable NNE encryption for an Oracle DB instance, add the Oracle NNE option to the option group associated with the DB instance. For more information, see Oracle native network encryption.

> **Note**
>
> You can't use both NNE and TLS on the same DB instance.

Configuring Kerberos authentication for Amazon RDS for Oracle

You can use Kerberos authentication to authenticate users when they connect to your Amazon RDS for Oracle DB instance. In this configuration, your DB instance works with AWS Directory Service for Microsoft Active Directory, also called AWS Managed Microsoft AD. When users authenticate

with an RDS for Oracle DB instance joined to the trusting domain, authentication requests are forwarded to the directory that you create with Directory Service.

Keeping all of your credentials in the same directory can save you time and effort. You have a centralized place for storing and managing credentials for multiple database instances. A directory can also improve your overall security profile.

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability of RDS for Oracle with Kerberos authentication, see Supported Regions and DB engines for Kerberos authentication in Amazon RDS.

> **Note**
>
> Kerberos authentication isn't supported for DB instance classes that are deprecated for RDS for Oracle DB instances. For more information, see RDS for Oracle DB instance classes.

Topics

- Setting up Kerberos for Oracle DB instances
- Managing a DB instance in a domain
- Connecting to Oracle with Kerberos authentication

Setting up Kerberos for Oracle DB instances

Use AWS Directory Service for Microsoft Active Directory, also called AWS Managed Microsoft AD, to set up Kerberos authentication for an Oracle DB instance. To set up Kerberos authentication, complete the following steps:

- Step 1: Create a directory using the AWS Managed Microsoft AD
- Step 2: Create a trust
- Step 3: Configure IAM permissions for Amazon RDS
- Step 4: Create and configure users
- Step 5: Enable cross-VPC traffic between the directory and the DB instance
- Step 6: Create or modify an Oracle DB instance

- Step 7: Create Kerberos authentication Oracle logins
- Step 8: Configure an Oracle client

> **Note**
>
> During the setup, RDS creates an Oracle database user named *managed_service_user*@*example.com* with the `CREATE SESSION` privilege, where *example.com* is your domain name. This user corresponds to the user that Directory Service creates inside your Managed Active Directory. Periodically, RDS uses the credentials provided by the Directory Service to log in to your Oracle database. Afterwards, RDS immediately destroys the ticket cache.

Step 1: Create a directory using the AWS Managed Microsoft AD

Directory Service creates a fully managed Active Directory in the AWS Cloud. When you create an AWS Managed Microsoft AD directory, Directory Service creates two domain controllers and Domain Name System (DNS) servers on your behalf. The directory servers are created in different subnets in a VPC. This redundancy helps make sure that your directory remains accessible even if a failure occurs.

When you create an AWS Managed Microsoft AD directory, Directory Service performs the following tasks on your behalf:

- Sets up an Active Directory within the VPC.
- Creates a directory administrator account with the user name Admin and the specified password. You use this account to manage your directory.

  > **Note**
  >
  > Be sure to save this password. Directory Service doesn't store it. You can reset it, but you can't retrieve it.

- Creates a security group for the directory controllers.

When you launch an AWS Managed Microsoft AD, AWS creates an Organizational Unit (OU) that contains all of your directory's objects. This OU has the NetBIOS name that you typed when you

created your directory and is located in the domain root. The domain root is owned and managed by AWS.

The Admin account that was created with your AWS Managed Microsoft AD directory has permissions for the most common administrative activities for your OU:

- Create, update, or delete users
- Add resources to your domain such as file or print servers, and then assign permissions for those resources to users in your OU
- Create additional OUs and containers
- Delegate authority
- Restore deleted objects from the Active Directory Recycle Bin
- Run AD and DNS Windows PowerShell modules on the Active Directory Web Service

The Admin account also has rights to perform the following domain-wide activities:

- Manage DNS configurations (add, remove, or update records, zones, and forwarders)
- View DNS event logs
- View security event logs

To create the directory, use the AWS Management Console, the AWS CLI, or the Directory Service API. Make sure to open the relevant outbound ports on the directory security group so that the directory can communicate with the Oracle DB instance.

To create a directory with AWS Managed Microsoft AD

1. Sign in to the AWS Management Console and open the Directory Service console at https://console.aws.amazon.com/directoryservicev2/.
2. In the navigation pane, choose **Directories** and choose **Set up Directory**.
3. Choose **AWS Managed Microsoft AD**. AWS Managed Microsoft AD is the only option that you can currently use with Amazon RDS.
4. Enter the following information:

   Directory DNS name

   The fully qualified name for the directory, such as **`corp.example.com`**.

Directory NetBIOS name

The short name for the directory, such as **CORP**.

Directory description

(Optional) A description for the directory.

Admin password

The password for the directory administrator. The directory creation process creates an administrator account with the user name Admin and this password.

The directory administrator password and can't include the word "admin." The password is case-sensitive and must be 8–64 characters in length. It must also contain at least one character from three of the following four categories:

- Lowercase letters (a–z)
- Uppercase letters (A–Z)
- Numbers (0–9)
- Non-alphanumeric characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Confirm password

The administrator password retyped.

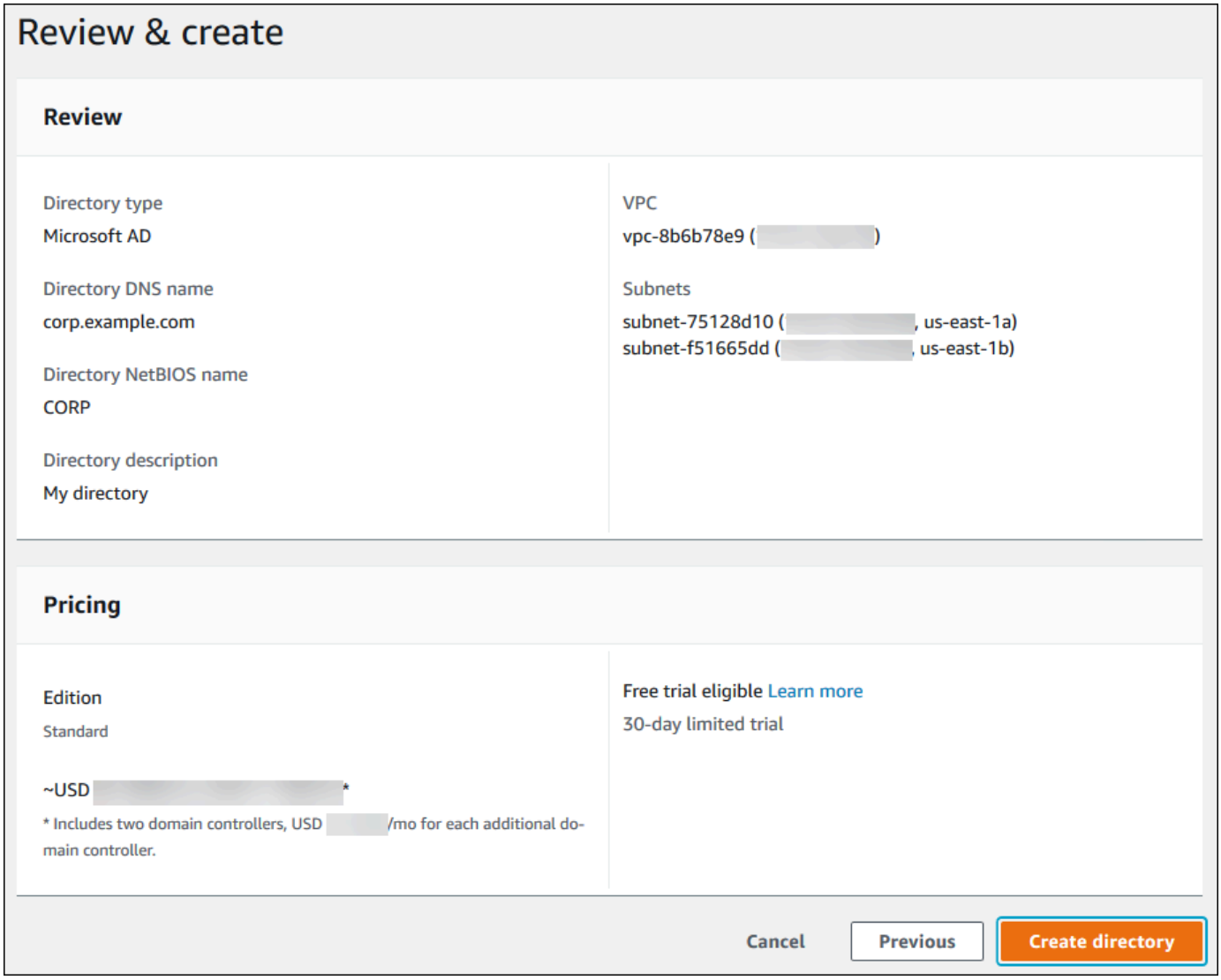
5. Choose **Next**.
6. Enter the following information in the **Networking** section and then choose **Next**:

   VPC

   The VPC for the directory. Create the Oracle DB instance in this same VPC.

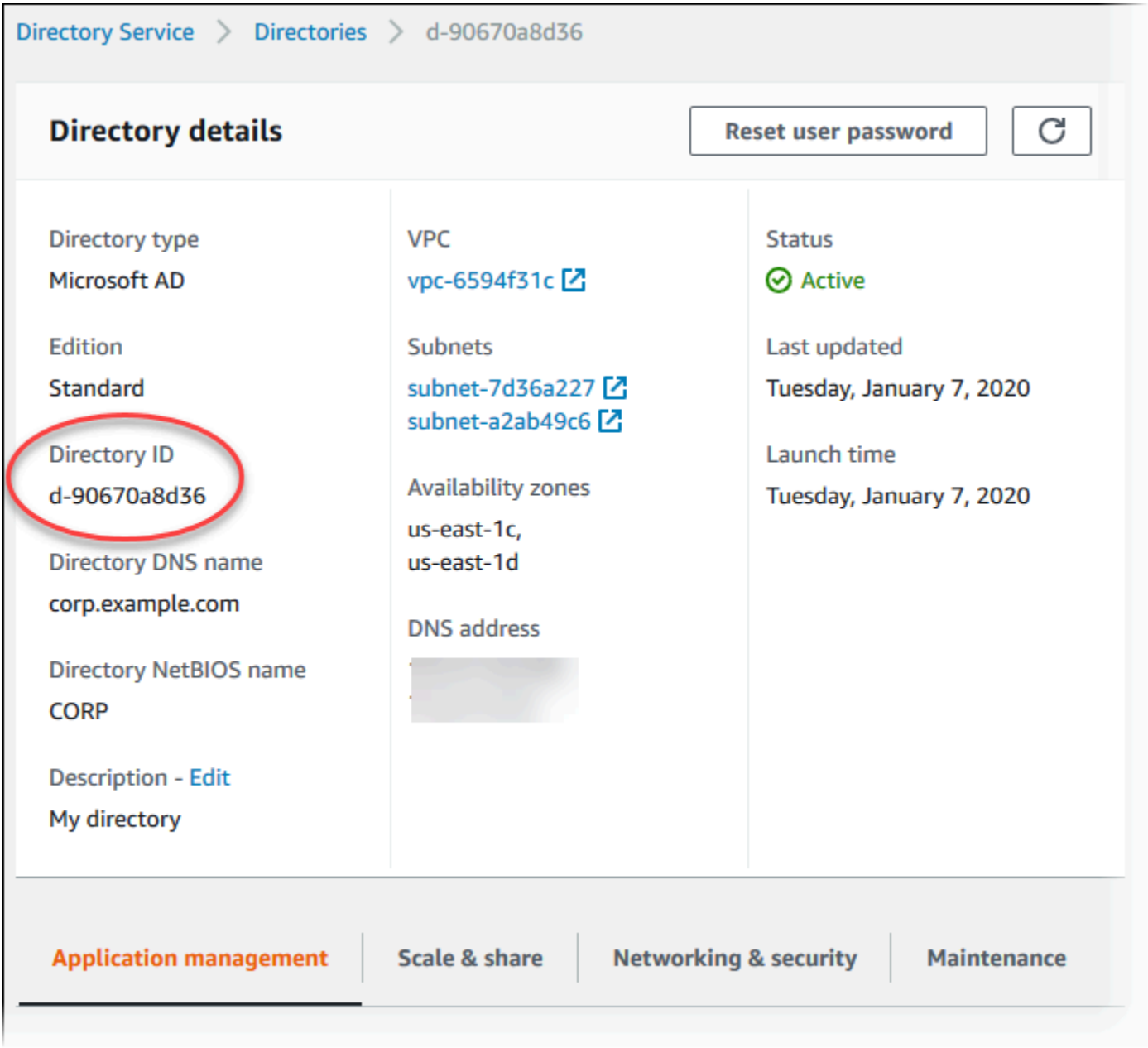
   Subnets

   Subnets for the directory servers. The two subnets must be in different Availability Zones.
7. Review the directory information and make any necessary changes. When the information is correct, choose **Create directory**.



It takes several minutes for the directory to be created. When it has been successfully created, the **Status** value changes to **Active**.

To see information about your directory, choose the directory name in the directory listing. Note the **Directory ID** value because you need this value when you create or modify your Oracle DB instance.



Step 2: Create a trust

If you plan to use AWS Managed Microsoft AD only, move on to Step 3: Configure IAM permissions for Amazon RDS.

To enable Kerberos authentication using your self-managed Active Directory, you must create a forest trust relationship between your self-managed Active Directory and the AWS Managed Microsoft AD created in the previous step. The trust can be one-way, where the AWS Managed Microsoft AD trusts the self-managed Active Directory. The trust can also be two-way, where both Active Directories trust each other. For more information about setting up forest trusts using

Directory Service, see When to create a trust relationship in the *Directory Service Administration Guide*.

Step 3: Configure IAM permissions for Amazon RDS

To call Directory Service for you, Amazon RDS requires an IAM role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`. This role allows Amazon RDS to make calls to the Directory Service.

> **Note**
>
> For the role to allow access, the AWS Security Token Service (AWS STS) endpoint must be activated in the correct AWS Region for your AWS account. AWS STS endpoints are active by default in all AWS Regions, and you can use them without any further actions. For more information, see Activating and deactivating AWS STS in an AWS Region in the *IAM User Guide*.

Creating an IAM role

When you create a DB instance using the AWS Management Console, and the console user has the `iam:CreateRole` permission, the console creates `rds-directoryservice-kerberos-access-role` automatically. Otherwise, you must create the IAM role manually. When you create an IAM role manually, choose `Directory Service`, and attach the AWS managed policy `AmazonRDSDirectoryServiceAccess` to it.

For more information about creating IAM roles for a service, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Note**
>
> The IAM role used for Windows Authentication for RDS for Microsoft SQL Server can't be used for RDS for Oracle.

Creating an IAM trust policy manually

Optionally, you can create resource policies with the required permissions instead of using the managed IAM policy `AmazonRDSDirectoryServiceAccess`. Specify both `directoryservice.rds.amazonaws.com` and `rds.amazonaws.com` as principals.

To limit the permissions that Amazon RDS gives another service for a specific resource, we recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies. The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of an Amazon RDS resource. For more information, see Preventing cross-service confused deputy problems.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Amazon RDS to prevent the confused deputy problem.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

For opt-in Regions, you must also include a service principal for that Region in the form of `directoryservice.rds.region_name.amazonaws.com`. For example, in the Africa (Cape Town) Region, use the following trust policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "directoryservice.rds.af-south-1.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:af-south-1:123456789012:db:mydbinstance"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

The role must also have the following IAM policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
```

```
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

Step 4: Create and configure users

You can create users with the Active Directory Users and Computers tool, which is one of the Active Directory Domain Services and Active Directory Lightweight Directory Services tools. In this case, *users* are individual people or entities that have access to your directory.

To create users in an Directory Service directory, you must be connected to a Windows-based Amazon EC2 instance that is a member of the Directory Service directory. At the same time, you must be logged in as a user that has privileges to create users. For more information about creating users in your Microsoft Active Directory, see Manage users and groups in AWS Managed Microsoft AD in the *Directory Service Administration Guide*.

Step 5: Enable cross-VPC traffic between the directory and the DB instance

If you plan to locate the directory and the DB instance in the same VPC, skip this step and move on to Step 6: Create or modify an Oracle DB instance.

If you plan to locate the directory and the DB instance in different AWS accounts or VPCs, configure cross-VPC traffic using VPC peering or AWS Transit Gateway. The following procedure enables traffic between VPCs using VPC peering. Follow the instructions in What is VPC peering? in the *Amazon Virtual Private Cloud Peering Guide*.

To enable cross-VPC traffic using VPC peering

1. Set up appropriate VPC routing rules to ensure that network traffic can flow both ways.
2. Ensure that the DB instance's security group can receive inbound traffic from the directory's security group. For more information, see Best practices for AWS Managed Microsoft AD in the *Directory Service Administration Guide*.
3. Ensure that there is no network access control list (ACL) rule to block traffic.

If a different AWS account owns the directory, you must share the directory.

To share the directory between AWS accounts

1. Start sharing the directory with the AWS account that the DB instance will be created in by following the instructions in Tutorial: Sharing your AWS Managed Microsoft AD directory for seamless EC2 Domain-join in the *Directory Service Administration Guide*.
2. Sign in to the Directory Service console using the account for the DB instance, and ensure that the domain has the SHARED status before proceeding.
3. While signed into the Directory Service console using the account for the DB instance, note the **Directory ID** value. You use this directory ID to join the DB instance to the domain.

Step 6: Create or modify an Oracle DB instance

Create or modify an Oracle DB instance for use with your directory. You can use the console, CLI, or RDS API to associate a DB instance with a directory. You can do this in one of the following ways:

- Create a new Oracle DB instance using the console, the create-db-instance CLI command, or the CreateDBInstance RDS API operation.

  For instructions, see Creating an Amazon RDS DB instance.
- Modify an existing Oracle DB instance using the console, the modify-db-instance CLI command, or the ModifyDBInstance RDS API operation.

  For instructions, see Modifying an Amazon RDS DB instance.
- Restore an Oracle DB instance from a DB snapshot using the console, the restore-db-instance-from-db-snapshot CLI command, or the RestoreDBInstanceFromDBSnapshot RDS API operation.

  For instructions, see Restoring to a DB instance.
- Restore an Oracle DB instance to a point-in-time using the console, the restore-db-instance-to-point-in-time CLI command, or the RestoreDBInstanceToPointInTime RDS API operation.

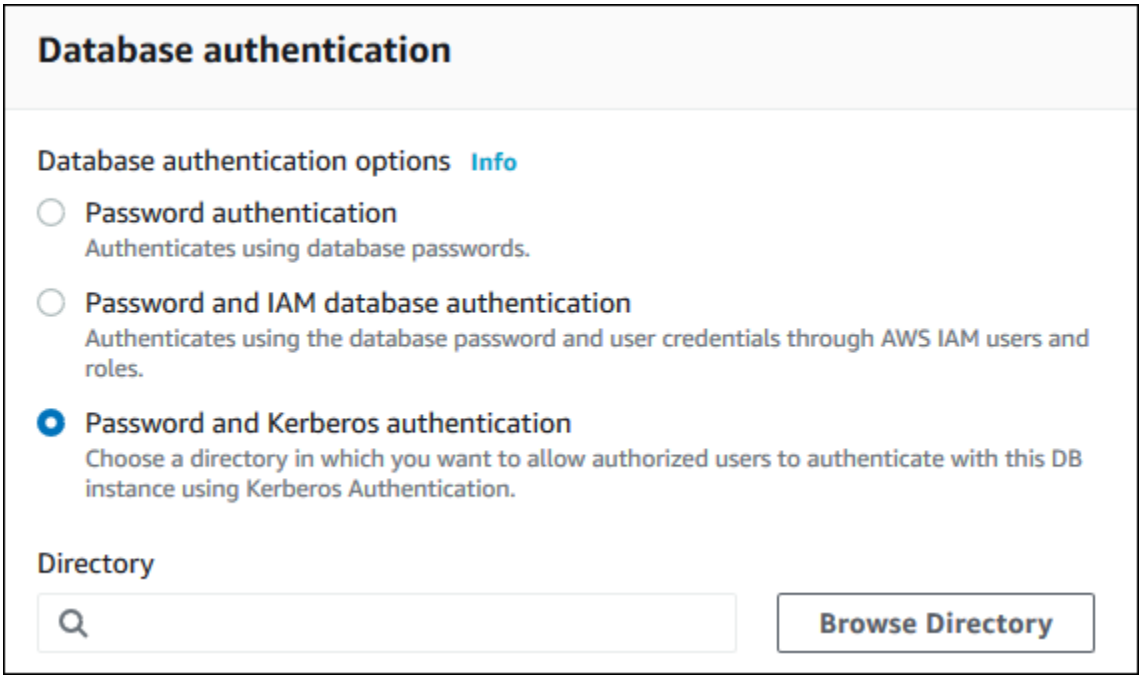
  For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

Kerberos authentication is only supported for Oracle DB instances in a VPC. The DB instance can be in the same VPC as the directory, or in a different VPC. When you create or modify the DB instance, do the following:

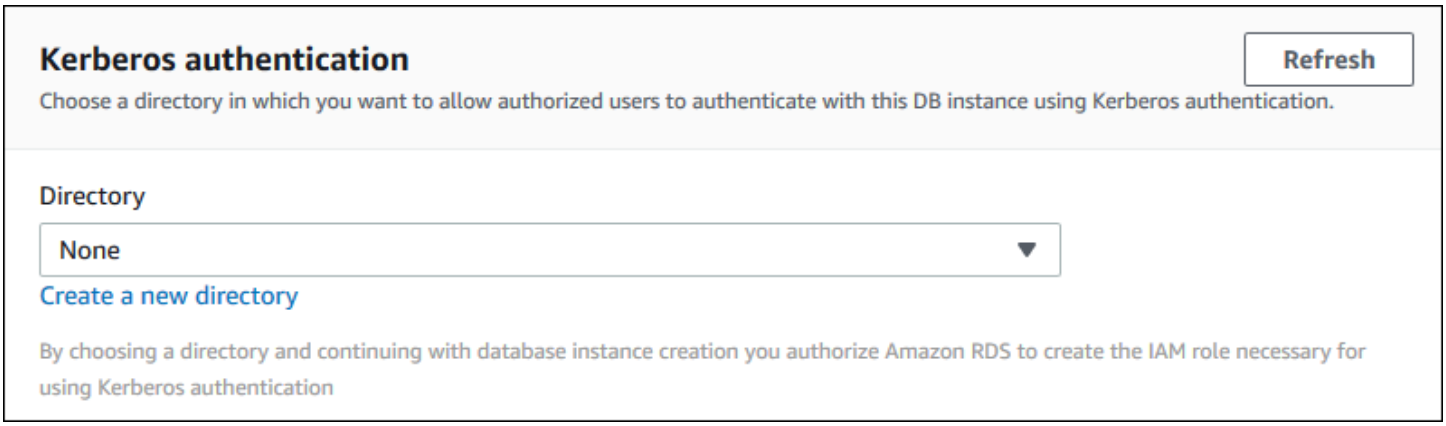
- Provide the domain identifier (d-* identifier) that was generated when you created your directory.

- Provide the name of the IAM role that you created.
- Ensure that the DB instance security group can receive inbound traffic from the directory security group and send outbound traffic to the directory.

When you use the console to create a DB instance, choose **Password and Kerberos authentication** in the **Database authentication** section. Choose **Browse Directory** and then select the directory, or choose **Create a new directory**.



When you use the console to modify or restore a DB instance, choose the directory in the **Kerberos authentication** section, or choose **Create a new directory**.



When you use the AWS CLI, the following parameters are required for the DB instance to be able to use the directory that you created:

- For the `--domain` parameter, use the domain identifier ("d-*" identifier) generated when you created the directory.
- For the `--domain-iam-role-name` parameter, use the role you created that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

For example, the following CLI command modifies a DB instance to use a directory.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --domain d-ID \
    --domain-iam-role-name role-name
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

Important

If you modify a DB instance to enable Kerberos authentication, reboot the DB instance after making the change.

Note

MANAGED_SERVICE_USER is a service account whose name is randomly generated by Directory Service for RDS. During the Kerberos authentication setup, RDS for Oracle creates a user with the same name and assigns it the `CREATE SESSION` privilege. The Oracle DB user is identified externally as *MANAGED_SERVICE_USER@EXAMPLE.COM*, where *EXAMPLE.COM* is the name of your domain. Periodically, RDS uses the credentials provided by the Directory Service to log in to your Oracle database. Afterward, RDS immediately destroys the ticket cache.

Step 7: Create Kerberos authentication Oracle logins

Use the Amazon RDS master user credentials to connect to the Oracle DB instance as you do any other DB instance. The DB instance is joined to the AWS Managed Microsoft AD domain. Thus, you can provision Oracle logins and users from the Microsoft Active Directory users in your domain. To manage database permissions, you grant and revoke standard Oracle permissions to these logins.

To allow a Microsoft Active Directory user to authenticate with Oracle

1. Connect to the Oracle DB instance using your Amazon RDS master user credentials.
2. Create an externally authenticated user in Oracle database.

   In the following example, replace *KRBUSER@CORP.EXAMPLE.COM* with the user name and domain name.

   ```
   CREATE USER "KRBUSER@CORP.EXAMPLE.COM" IDENTIFIED EXTERNALLY;
   GRANT CREATE SESSION TO "KRBUSER@CORP.EXAMPLE.COM";
   ```

   Users (both humans and applications) from your domain can now connect to the Oracle DB instance from a domain joined client machine using Kerberos authentication.

Step 8: Configure an Oracle client

To configure an Oracle client, meet the following requirements:

- Create a configuration file named krb5.conf (Linux) or krb5.ini (Windows) to point to the domain. Configure the Oracle client to use this configuration file.
- Verify that traffic can flow between the client host and Directory Service over DNS port 53 over TCP/UDP, Kerberos ports (88 and 464 for managed Directory Service) over TCP, and LDAP port 389 over TCP.
- Verify that traffic can flow between the client host and the DB instance over the database port.

Following is sample content for AWS Managed Microsoft AD.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
```

```
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = CORP.EXAMPLE.COM
 example.com = CORP.EXAMPLE.COM
```

Following is sample content for on-premise Microsoft AD. In your krb5.conf or krb5.ini file, replace *on-prem-ad-server-name* with the name of your on-premises AD server.

```
[libdefaults]
 default_realm = ONPREM.COM
[realms]
 AWSAD.COM = {
  kdc = awsad.com
  admin_server = awsad.com
 }
 ONPREM.COM = {
  kdc = on-prem-ad-server-name
  admin_server = on-prem-ad-server-name
 }
[domain_realm]
 .awsad.com = AWSAD.COM
 awsad.com= AWSAD.COM
 .onprem.com = ONPREM.COM
 onprem.com= ONPREM.COM
```

Note

After you configure your krb5.ini or krb5.conf file, we recommend that you reboot the server.

The following is sample sqlnet.ora content for a SQL*Plus configuration:

```
SQLNET.AUTHENTICATION_SERVICES=(KERBEROS5PRE,KERBEROS5)
SQLNET.KERBEROS5_CONF=path_to_krb5.conf_file
```

For an example of a SQL Developer configuration, see Document 1609359.1 from Oracle Support.

Managing a DB instance in a domain

You can use the console, the CLI, or the RDS API to manage your DB instance and its relationship with your Microsoft Active Directory. For example, you can associate a Microsoft Active Directory to enable Kerberos authentication. You can also disassociate a Microsoft Active Directory to disable Kerberos authentication. You can also move a DB instance to be externally authenticated by one Microsoft Active Directory to another.

For example, using the CLI, you can do the following:

- To reattempt enabling Kerberos authentication for a failed membership, use the modify-db-instance CLI command and specify the current membership's directory ID for the `--domain` option.
- To disable Kerberos authentication on a DB instance, use the modify-db-instance CLI command and specify `none` for the `--domain` option.
- To move a DB instance from one domain to another, use the modify-db-instance CLI command and specify the domain identifier of the new domain for the `--domain` option.

Viewing the status of domain membership

After you create or modify your DB instance, the DB instance becomes a member of the domain. You can view the status of the domain membership for the DB instance in the console or by running the describe-db-instances CLI command. The status of the DB instance can be one of the following:

- `kerberos-enabled` – The DB instance has Kerberos authentication enabled.
- `enabling-kerberos` – AWS is in the process of enabling Kerberos authentication on this DB instance.
- `pending-enable-kerberos` – Enabling Kerberos authentication is pending on this DB instance.
- `pending-maintenance-enable-kerberos` – AWS will attempt to enable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `pending-disable-kerberos` – Disabling Kerberos authentication is pending on this DB instance.
- `pending-maintenance-disable-kerberos` – AWS will attempt to disable Kerberos authentication on the DB instance during the next scheduled maintenance window.

- `enable-kerberos-failed` – A configuration problem has prevented AWS from enabling Kerberos authentication on the DB instance. Correct the configuration problem before reissuing the command to modify the DB instance.
- `disabling-kerberos` – AWS is in the process of disabling Kerberos authentication on this DB instance.

A request to enable Kerberos authentication can fail because of a network connectivity issue or an incorrect IAM role. If the attempt to enable Kerberos authentication fails when you create or modify a DB instance, make sure that you're using the correct IAM role. Then modify the DB instance to join the domain.

> **Note**
>
> Only Kerberos authentication with Amazon RDS for Oracle sends traffic to the domain's DNS servers. All other DNS requests are treated as outbound network access on your DB instances running Oracle. For more information about outbound network access with Amazon RDS for Oracle, see Setting up a custom DNS server.

Force-rotating Kerberos keys

A secret key is shared between AWS Managed Microsoft AD and Amazon RDS for Oracle DB instance. This key is rotated automatically every 45 days. You can use the following Amazon RDS procedure to force the rotation of this key.

```
SELECT rdsadmin.rdsadmin_kerberos_auth_tasks.rotate_kerberos_keytab AS TASK_ID FROM
 DUAL;
```

> **Note**
>
> In a read replica configuration, this procedure is available only on the source DB instance and not on the read replica.

The `SELECT` statement returns the ID of the task in a `VARCHAR2` data type. You can view the status of an ongoing task in a bdump file. The bdump files are located in the `/rdsdbdata/log/trace` directory. Each bdump file name is in the following format.

```
dbtask-task-id.log
```

You can view the result by displaying the task's output file.

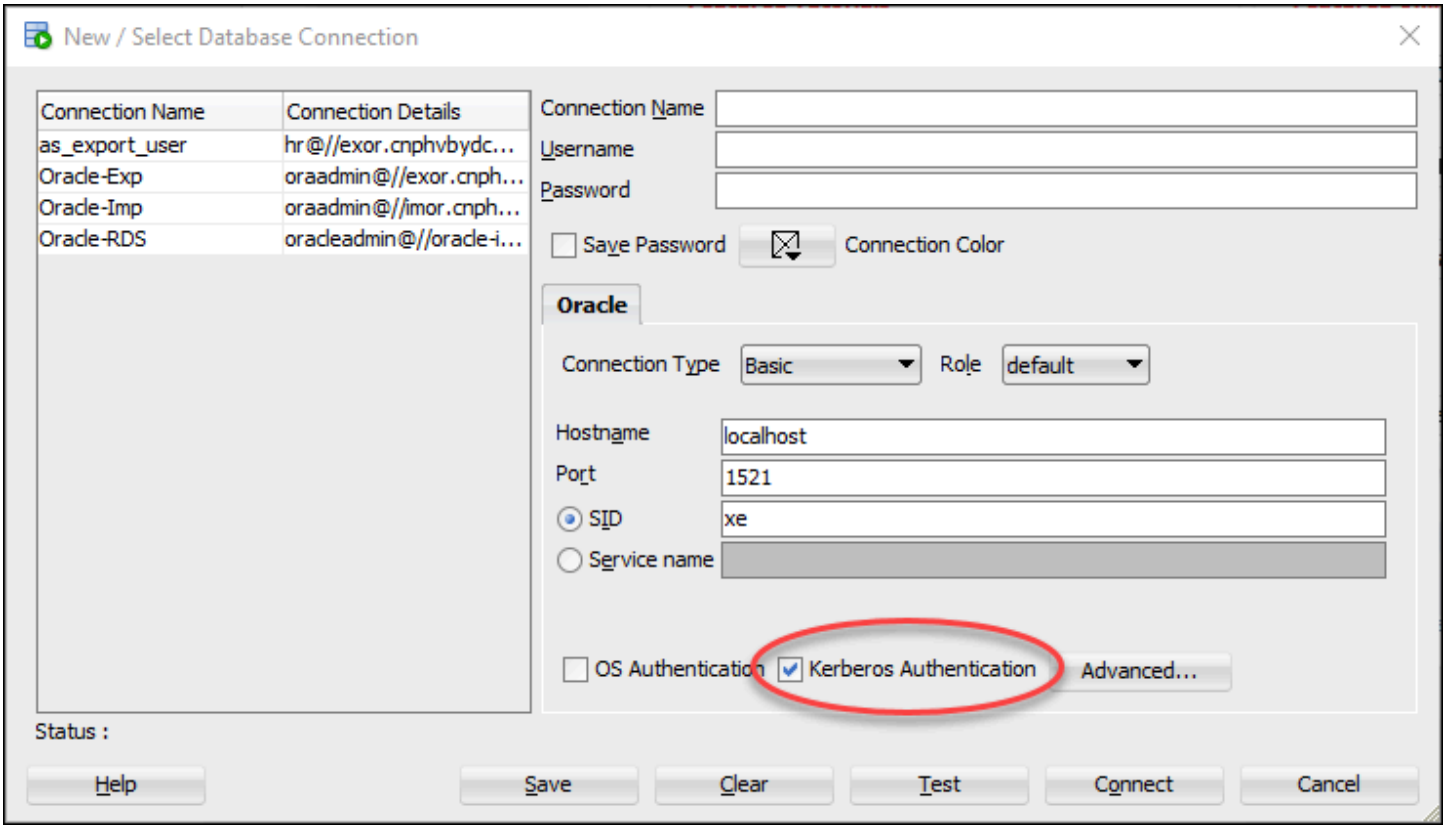
```
SELECT text FROM table(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-task-
id.log'));
```

Replace *task-id* with the task ID returned by the procedure.

> **Note**
>
> Tasks are executed asynchronously.

Connecting to Oracle with Kerberos authentication

This section assumes that you have set up your Oracle client as described in Step 8: Configure an Oracle client. To connect to the Oracle DB with Kerberos authentication, log in using the Kerberos authentication type. For example, after launching Oracle SQL Developer, choose **Kerberos Authentication** as the authentication type, as shown in the following example.



To connect to Oracle with Kerberos authentication with SQL*Plus:

1. At a command prompt, run the following command:

   ```
   kinit username
   ```

   Replace *username* with the user name and, at the prompt, enter the password stored in the Microsoft Active Directory for the user.
2. Open SQL*Plus and connect using the DNS name and port number for the Oracle DB instance.

   For more information about connecting to an Oracle DB instance in SQL*Plus, see Connecting to your DB instance using SQL*Plus.

> **Tip**
>
> If you are using a native Windows cache, you can also set the `SQLNET.KERBEROS5_CC_NAME` parameter to `OSMSFT://` or `MSLSA` in the sqlnet.ora file to use the credentials stored in the Microsoft Active Directory.

Configuring UTL_HTTP access using certificates and an Oracle wallet

Amazon RDS supports outbound network access on your RDS for Oracle DB instances. To connect your DB instance to the network, you can use the following PL/SQL packages:

`UTL_HTTP`

This package makes HTTP calls from SQL and PL/SQL. You can use it to access data on the Internet over HTTP. For more information, see UTL_HTTP in the Oracle documentation.

`UTL_TCP`

This package provides TCP/IP client-side access functionality in PL/SQL. This package is useful to PL/SQL applications that use Internet protocols and email. For more information, see UTL_TCP in the Oracle documentation.

`UTL_SMTP`

This package provides interfaces to the SMTP commands that enable a client to dispatch emails to an SMTP server. For more information, see UTL_SMTP in the Oracle documentation.

By completing the following tasks, you can configure `UTL_HTTP.REQUEST` to work with websites that require client authentication certificates during the SSL handshake. You can also configure password authentication for `UTL_HTTP` access to websites by modifying the Oracle wallet generation commands and the `DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE` procedure. For more information, see DBMS_NETWORK_ACL_ADMIN in the Oracle Database documentation.

> **Note**
>
> You can adapt the following tasks for `UTL_SMTP`, which enables you to send emails over SSL/TLS (including Amazon Simple Email Service).

Topics

- Considerations when configuring UTL_HTTP access
- Step 1: Get the root certificate for a website
- Step 2: Create an Oracle wallet
- Step 3: Download your Oracle wallet to your RDS for Oracle instance
- Step 4: Grant user permissions for the Oracle wallet
- Step 5: Configure access to a website from your DB instance
- Step 6: Test connections from your DB instance to a website

Considerations when configuring UTL_HTTP access

Before configuring access, consider the following:

- You can use SMTP with the UTL_MAIL option. For more information, see Oracle UTL_MAIL.
- The Domain Name Server (DNS) name of the remote host can be any of the following:
  - Publicly resolvable.
  - The endpoint of an Amazon RDS DB instance.
  - Resolvable through a custom DNS server. For more information, see Setting up a custom DNS server.
  - The private DNS name of an Amazon EC2 instance in the same VPC or a peered VPC. In this case, make sure that the name is resolvable through a custom DNS server. Alternatively, to use the DNS provided by Amazon, you can enable the `enableDnsSupport` attribute in the

VPC settings and enable DNS resolution support for the VPC peering connection. For more information, see DNS support in your VPC and Modifying your VPC peering connection.

- To connect securely to remote SSL/TLS resources, we recommend that you create and upload customized Oracle wallets. By using the Amazon S3 integration with Amazon RDS for Oracle feature, you can download a wallet from Amazon S3 into Oracle DB instances. For information about Amazon S3 integration for Oracle, see Amazon S3 integration.

- You can establish database links between Oracle DB instances over an SSL/TLS endpoint if the Oracle SSL option is configured for each instance. No further configuration is required. For more information, see Oracle Secure Sockets Layer.

Step 1: Get the root certificate for a website

For the RDS for Oracle DB instance to make secure connections to a website, add the root CA certificate. Amazon RDS uses the root certificate to sign the website certificate to the Oracle wallet.

You can get the root certificate in various ways. For example, you can do the following:

1. Use a web server to visit the website secured by the certificate.
2. Download the root certificate that was used for signing.

For AWS services, root certificates typically reside in the Amazon trust services repository.

Step 2: Create an Oracle wallet

Create an Oracle wallet that contains both the web server certificates and the client authentication certificates. The RDS Oracle instance uses the web server certificate to establish a secure connection to the website. The website needs the client certificate to authenticate the Oracle database user.

You might want to configure secure connections without using client certificates for authentication. In this case, you can skip the Java keystore steps in the following procedure.

To create an Oracle wallet

1. Place the root and client certificates in a single directory, and then change into this directory.
2. Convert the .p12 client certificate to the Java keystore.

> **Note**
>
> If you're not using client certificates for authentication, you can skip this step.

The following example converts the client certificate named *client_certificate.p12* to the Java keystore named *client_keystore.jks*. The keystore is then included in the Oracle wallet. The keystore password is *P12PASSWORD*.

```
orapki wallet pkcs12_to_jks -wallet ./client_certificate.p12 -
jksKeyStoreLoc ./client_keystore.jks -jksKeyStorepwd P12PASSWORD
```

3. Create a directory for your Oracle wallet that is different from the certificate directory.

   The following example creates the directory `/tmp/wallet`.

   ```
   mkdir -p /tmp/wallet
   ```

4. Create an Oracle wallet in your wallet directory.

   The following example sets the Oracle wallet password to *P12PASSWORD*, which is the same password used by the Java keystore in a previous step. Using the same password is convenient, but not necessary. The `-auto_login` parameter turns on the automatic login feature, so that you don't need to specify a password every time you want to access it.

   > **Note**
   >
   > Specify a password other than the prompt shown here as a security best practice.

   ```
   orapki wallet create -wallet /tmp/wallet -pwd P12PASSWORD -auto_login
   ```

5. Add the Java keystore to your Oracle wallet.

   > **Note**
   >
   > If you're not using client certificates for authentication, you can skip this step.

The following example adds the keystore *client_keystore.jks* to the Oracle wallet named */tmp/wallet*. In this example, you specify the same password for the Java keystore and the Oracle wallet.

```
orapki wallet jks_to_pkcs12 -wallet /tmp/wallet -pwd P12PASSWORD -
keystore ./client_keystore.jks -jkspwd P12PASSWORD
```

6. Add the root certificate for your target website to the Oracle wallet.

   The following example adds a certificate named *Root_CA.cer*.

   ```
   orapki wallet add -wallet /tmp/wallet -trusted_cert -cert ./Root_CA.cer -
   pwd P12PASSWORD
   ```

7. Add any intermediate certificates.

   The following example adds a certificate named *Intermediate.cer*. Repeat this step as many times as need to load all intermediate certificates.

   ```
   orapki wallet add -wallet /tmp/wallet -trusted_cert -cert ./Intermediate.cer -
   pwd P12PASSWORD
   ```

8. Confirm that your newly created Oracle wallet has the required certificates.

   ```
   orapki wallet display -wallet /tmp/wallet -pwd P12PASSWORD
   ```

Step 3: Download your Oracle wallet to your RDS for Oracle instance

In this step, you upload your Oracle wallet to Amazon S3, and then download the wallet from Amazon S3 to your RDS for Oracle instance.

To download your Oracle wallet to your RDS for Oracle DB instance

1. Complete the prerequisites for Amazon S3 integration with Oracle, and add the `S3_INTEGRATION` option to your Oracle DB instance. Ensure that the IAM role for the option has access to the Amazon S3 bucket you are using.

   For more information, see Amazon S3 integration.

2. Log in to your DB instance as the master user, and then create an Oracle directory to hold the Oracle wallet.

   The following example creates an Oracle directory named *WALLET_DIR*.

   ```
   EXEC rdsadmin.rdsadmin_util.create_directory('WALLET_DIR');
   ```

   For more information, see Creating and dropping directories in the main data storage space.
3. Upload the Oracle wallet to your Amazon S3 bucket.

   You can use any supported upload technique.
4. If you're re-uploading an Oracle wallet, delete the existing wallet. Otherwise, skip to the next step.

   The following example removes the existing wallet, which is named *cwallet.sso*.

   ```
   EXEC UTL_FILE.FREMOVE ('WALLET_DIR','cwallet.sso');
   ```
5. Download the Oracle wallet from your Amazon S3 bucket to the Oracle DB instance.

   The following example downloads the wallet named *cwallet.sso* from the Amazon S3 bucket named *my_s3_bucket* to the DB instance directory named *WALLET_DIR*.

   ```
   SELECT rdsadmin.rdsadmin_s3_tasks.download_from_s3(
         p_bucket_name    =>  'my_s3_bucket',
         p_s3_prefix      =>  'cwallet.sso',
         p_directory_name =>  'WALLET_DIR')
      AS TASK_ID FROM DUAL;
   ```
6. (Optional) Download a password-protected Oracle wallet.

   Download this wallet only if you want to require a password for every use of the wallet. The following example downloads password-protected wallet *ewallet.p12*.

   ```
   SELECT rdsadmin.rdsadmin_s3_tasks.download_from_s3(
         p_bucket_name    =>  'my_s3_bucket',
         p_s3_prefix      =>  'ewallet.p12',
         p_directory_name =>  'WALLET_DIR')
      AS TASK_ID FROM DUAL;
   ```
7. Check the status of your DB task.

Substitute the task ID returned from the preceding steps for *dbtask-1234567890123-4567.log* in the following example.

```
SELECT TEXT FROM
 TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-1234567890123-4567.log'));
```

8. Check the contents of the directory that you're using to store the Oracle wallet.

```
SELECT * FROM TABLE(rdsadmin.rds_file_util.listdir(p_directory => 'WALLET_DIR'));
```

For more information, see Listing files in a DB instance directory.

Step 4: Grant user permissions for the Oracle wallet

You can either create a new database user or configure an existing user. In either case, you must configure the user to access the Oracle wallet for secure connections and client authentication using certificates.

To grant user permissions for the Oracle wallet

1. Log in your RDS for Oracle DB instance as the master user.
2. If you don't want to configure an existing database user, create a new user. Otherwise, skip to the next step.

   The following example creates a database user named *my-user*.

```
CREATE USER my-user IDENTIFIED BY my-user-pwd;
GRANT CONNECT TO my-user;
```

3. Grant permission to your database user on the directory containing your Oracle wallet.

   The following example grants read access to user *my-user* on directory *WALLET_DIR*.

```
GRANT READ ON DIRECTORY WALLET_DIR TO my-user;
```

4. Grant permission to your database user to use the UTL_HTTP package.

   The following PL/SQL program grants UTL_HTTP access to user *my-user*.

```
BEGIN
  rdsadmin.rdsadmin_util.grant_sys_object('UTL_HTTP', UPPER('my-user'));
  END;
/
```

5. Grant permission to your database user to use the UTL_FILE package.

   The following PL/SQL program grants UTL_FILE access to user *my-user*.

```
BEGIN
  rdsadmin.rdsadmin_util.grant_sys_object('UTL_FILE', UPPER('my-user'));
  END;
/
```

Step 5: Configure access to a website from your DB instance

In this step, you configure your Oracle database user so that it can connect to your target website using UTL_HTTP, your uploaded Oracle Wallet, and the client certificate. For more information, see Configuring Access Control to an Oracle Wallet in the Oracle Database documentation.

To configure access to a website from your RDS for Oracle DB instance

1. Log in your RDS for Oracle DB instance as the master user.
2. Create a Host Access Control Entry (ACE) for your user and the target website on a secure port.

   The following example configures *my-user* to access *secret.encrypted-website.com* on secure port 443.

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host       => 'secret.encrypted-website.com',
    lower_port => 443,
    upper_port => 443,
    ace        => xs$ace_type(privilege_list => xs$name_list('http'),
                              principal_name => 'my-user',
                              principal_type => xs_acl.ptype_db));
                           -- If the program unit results in PLS-00201, set
                           -- the principal_type parameter to 2 as follows:
                           -- principal_type => 2));
END;
```

```
/
```

Important

The preceding program unit can result in the following error: `PLS-00201: identifier 'XS_ACL' must be declared`. If this error is returned, replace the line that assigns a value to `principal_type` with the following line, and then rerun the program unit:

```
principal_type => 2));
```

For more information about constants in the PL/SQL package XS_ACL, see *Real Application Security Administrator's and Developer's Guide* in the Oracle Database documentation.

For more information, see Configuring Access Control for External Network Services in the Oracle Database documentation.

3. (Optional) Create an ACE for your user and target website on the standard port.

   You might need to use the standard port if some web pages are served from the standard web server port (80) instead of the secure port (443).

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host       => 'secret.encrypted-website.com',
    lower_port => 80,
    upper_port => 80,
    ace        => xs$ace_type(privilege_list => xs$name_list('http'),
                              principal_name => 'my-user',
                              principal_type => xs_acl.ptype_db));
                           -- If the program unit results in PLS-00201, set
                           -- the principal_type parameter to 2 as follows:
                           -- principal_type => 2));
END;
/
```

4. Confirm that the access control entries exist.

```
SET LINESIZE 150
COLUMN HOST FORMAT A40
COLUMN ACL FORMAT A50

SELECT HOST, LOWER_PORT, UPPER_PORT, ACL
  FROM DBA_NETWORK_ACLS
ORDER BY HOST;
```

5. Grant permission to your database user to use the UTL_HTTP package.

   The following PL/SQL program grants UTL_HTTP access to user *my-user*.

```
BEGIN
  rdsadmin.rdsadmin_util.grant_sys_object('UTL_HTTP', UPPER('my-user'));
  END;
/
```

6. Confirm that related access control lists exist.

```
SET LINESIZE 150
COLUMN ACL FORMAT A50
COLUMN PRINCIPAL FORMAT A20
COLUMN PRIVILEGE FORMAT A10

SELECT ACL, PRINCIPAL, PRIVILEGE, IS_GRANT,
       TO_CHAR(START_DATE, 'DD-MON-YYYY') AS START_DATE,
       TO_CHAR(END_DATE, 'DD-MON-YYYY') AS END_DATE
  FROM DBA_NETWORK_ACL_PRIVILEGES
ORDER BY ACL, PRINCIPAL, PRIVILEGE;
```

7. Grant permission to your database user to use certificates for client authentication and your Oracle wallet for connections.

   Note

   If you're not using client certificates for authentication, you can skip this step.

```
DECLARE
  l_wallet_path all_directories.directory_path%type;
BEGIN
```

```
  SELECT DIRECTORY_PATH
    INTO l_wallet_path
    FROM ALL_DIRECTORIES
   WHERE UPPER(DIRECTORY_NAME)='WALLET_DIR';
  DBMS_NETWORK_ACL_ADMIN.APPEND_WALLET_ACE(
    wallet_path => 'file:/' || l_wallet_path,
    ace         =>  xs$ace_type(privilege_list => xs
$name_list('use_client_certificates'),
                                principal_name => 'my-user',
                                principal_type => xs_acl.ptype_db));
END;
/
```

Step 6: Test connections from your DB instance to a website

In this step, you configure your database user so that it can connect to the website using UTL_HTTP, your uploaded Oracle Wallet, and the client certificate.

To configure access to a website from your RDS for Oracle DB instance

1. Log in your RDS for Oracle DB instance as a database user with UTL_HTTP permissions.
2. Confirm that a connection to your target website can resolve the host address.

   The following example gets the host address from *secret.encrypted-website.com*.

   ```
   SELECT UTL_INADDR.GET_HOST_ADDRESS(host => 'secret.encrypted-website.com')
     FROM DUAL;
   ```

3. Test a failed connection.

   The following query fails because UTL_HTTP requires the location of the Oracle wallet with the certificates.

   ```
   SELECT UTL_HTTP.REQUEST('secret.encrypted-website.com') FROM DUAL;
   ```

4. Test website access by using UTL_HTTP.SET_WALLET and selecting from DUAL.

   ```
   DECLARE
     l_wallet_path all_directories.directory_path%type;
   BEGIN
   ```

```
  SELECT DIRECTORY_PATH
    INTO l_wallet_path
    FROM ALL_DIRECTORIES
   WHERE UPPER(DIRECTORY_NAME)='WALLET_DIR';
  UTL_HTTP.SET_WALLET('file:/' || l_wallet_path);
END;
/

SELECT UTL_HTTP.REQUEST('secret.encrypted-website.com') FROM DUAL;
```

5. (Optional) Test website access by storing your query in a variable and using EXECUTE IMMEDIATE.

```
DECLARE
  l_wallet_path all_directories.directory_path%type;
  v_webpage_sql VARCHAR2(1000);
  v_results     VARCHAR2(32767);
BEGIN
  SELECT DIRECTORY_PATH
    INTO l_wallet_path
    FROM ALL_DIRECTORIES
   WHERE UPPER(DIRECTORY_NAME)='WALLET_DIR';
  v_webpage_sql := 'SELECT UTL_HTTP.REQUEST(''secret.encrypted-website.com'', ''''',
 ''file:/' ||l_wallet_path||''''') FROM DUAL';
  DBMS_OUTPUT.PUT_LINE(v_webpage_sql);
  EXECUTE IMMEDIATE v_webpage_sql INTO v_results;
  DBMS_OUTPUT.PUT_LINE(v_results);
END;
/
```

6. (Optional) Find the file system location of your Oracle wallet directory.

```
SELECT * FROM TABLE(rdsadmin.rds_file_util.listdir(p_directory => 'WALLET_DIR'));
```

Use the output from the previous command to make an HTTP request. For example, if the directory is *rdsdbdata/userdirs/01*, run the following query.

```
SELECT UTL_HTTP.REQUEST('https://secret.encrypted-website.com/', '',
 'file://rdsdbdata/userdirs/01')
FROM   DUAL;
```

Working with CDBs in RDS for Oracle

In the Oracle multitenant architecture, a container database (CDB) can include customer-created pluggable databases (PDBs). For more information about CDBs, see Introduction to the Multitenant Architecture in the Oracle Database documentation.

Topics

- Overview of RDS for Oracle CDBs
- Configuring an RDS for Oracle CDB
- Backing up and restoring a CDB
- Converting an RDS for Oracle non-CDB to a CDB
- Converting the single-tenant configuration to multi-tenant
- Adding an RDS for Oracle tenant database to your CDB instance
- Modifying an RDS for Oracle tenant database
- Deleting an RDS for Oracle tenant database from your CDB
- Viewing tenant database details
- Upgrading your CDB

Overview of RDS for Oracle CDBs

You can create an RDS for Oracle DB instance as a container database (CDB) when you run Oracle Database 19c or higher. Starting with Oracle Database 21c, all databases are CDBs. A CDB differs from a non-CDB because it can contain pluggable databases (PDBs), which are called *tenant databases* in RDS for Oracle. A PDB is a portable collection of schemas and objects that appears to an application as a separate database.

You create your initial tenant database (PDB) when you create your CDB instance. In RDS for Oracle, your client application interacts with a PDB rather than the CDB. Your experience with a PDB is mostly identical to your experience with a non-CDB.

Topics

- Multi-tenant configuration of the CDB architecture
- Single-tenant configuration of the CDB architecture
- Creation and conversion options for CDBs

- User accounts and privileges in a CDB
- Parameter group families in a CDB
- Limitations of RDS for Oracle CDBs

Multi-tenant configuration of the CDB architecture

RDS for Oracle supports the *multi-tenant configuration* of the Oracle multitenant architecture, also called the *CDB architecture*. In this configuration, your RDS for Oracle CDB instance can contain 1–30 tenant databases, depending on the database edition and any required option licenses. In Oracle database, a tenant database is a PDB. Your DB instance must use Oracle database release 19.0.0.0.ru-2022-01.rur-2022.r1 or higher.

> **Note**
>
> The Amazon RDS configuration is called "multi-tenant" rather than "multitenant" because it is a capability of Amazon RDS, not just the Oracle DB engine. Similarly, the RDS term "tenant" refers to any tenant in an RDS configuration, not just Oracle PDBs. In the RDS documentation, the unhyphenated term "Oracle multitenant" refers exclusively to the Oracle database CDB architecture, which is compatible with both on-premises and RDS deployments.

You can configure the following settings:

- Tenant database name
- Tenant database master username
- Tenant database master password (optionally integrated with Secrets Manager)
- Tenant database character set
- Tenant database national character set

The tenant database character set can be different from the CDB character set. The same applies to the national character set. After you create your initial tenant database, you can create, modify, or delete tenant databases using RDS APIs. The CDB name defaults to RDSCDB and can't be changed. For more information, see Settings for DB instances and Modifying an RDS for Oracle tenant database.

Single-tenant configuration of the CDB architecture

RDS for Oracle supports a legacy configuration of the Oracle multitenant architecture called the *single-tenant configuration*. In this configuration, an RDS for Oracle CDB instance can contain only one tenant (PDB). You can't create more PDBs later.

Creation and conversion options for CDBs

Oracle Database 21c supports only CDBs, whereas Oracle Database 19c supports both CDBs and non-CDBs. All RDS for Oracle CDB instances support both the multi-tenant and single-tenant configurations.

Creation, conversion, and upgrade options for the Oracle database architecture

The following table shows the different architecture options for creating and upgrading RDS for Oracle databases.

Release	Database creation options	Architecture conversion options	Major version upgrade targets
Oracle Database 21c	CDB architecture only	N/A	N/A
Oracle Database 19c	CDB or non-CDB architecture	Non-CDB to CDB architecture (April 2021 RU or higher)	Oracle Database 21c CDB

As shown in the preceding table, you can't directly upgrade a non-CDB to a CDB in a new major database version. But you can convert an Oracle Database 19c non-CDB to an Oracle Database 19c CDB, and then upgrade the Oracle Database 19c CDB to an Oracle Database 21c CDB. For more information, see Converting an RDS for Oracle non-CDB to a CDB.

Conversion options for CDB architecture configurations

The following table shows the different options for converting the architecture configuration of an RDS for Oracle DB instance.

Current architecture and configuration	Conversion to the single-tenant configuration of the CDB architecture	Conversion to the multi-tenant configuration of the CDB architecture	Conversion to the non-CDB architecture
Non-CDB	Supported	Supported*	N/A
CDB using the single-tenant configuration	N/A	Supported	Not supported
CDB using the multi-tenant configuration	Not supported	N/A	Not supported

* You can't convert a non-CDB to the multi-tenant configuration in a single operation. When you convert a non-CDB to a CDB, the CDB is in the single-tenant configuration. You can then convert the single-tenant to the multi-tenant configuration in a separate operation.

User accounts and privileges in a CDB

In the Oracle multitenant architecture, all user accounts are either *common users* or *local users*. A CDB common user is a database user whose single identity and password are known in the CDB root and in every existing and future PDB. In contrast, a local user exists only in a single PDB.

The RDS master user is a local user account in the PDB, which you name when you create your DB instance. If you create new user accounts, these users will also be local users residing in the PDB. You can't use any user accounts to create new PDBs or modify the state of the existing PDB.

The `rdsadmin` user is a common user account. You can run RDS for Oracle packages that exist in this account, but you can't log in as `rdsadmin`. For more information, see About Common Users and Local Users in the Oracle documentation.

For master users in both the multi-tenant and single-tenant configurations, you can use credentials that are self-managed or managed by AWS Secrets Manager. In the single-tenant configuration, you use instance-level CLI commands such as `create-db-instance` for managed master passwords. In the multi-tenant configuration, you use tenant database commands such as `create-tenant-database` for managed master passwords. For more information about Secrets

Manager integration, see Managing the master user password for an RDS for Oracle tenant database with Secrets Manager.

Parameter group families in a CDB

CDBs have their own parameter group families and default parameter values. The CDB parameter group families are as follows:

- oracle-ee-cdb-21
- oracle-se2-cdb-21
- oracle-ee-cdb-19
- oracle-se2-cdb-19

Limitations of RDS for Oracle CDBs

RDS for Oracle supports a subset of features available in an on-premises CDB.

CDB limitations

The following limitations apply to RDS for Oracle at the CDB level:

- You can't connect to a CDB. You always connect to the tenant database (PDB) rather than the CDB. Specify the endpoint for the PDB just as for a non-CDB. The only difference is that you specify *pdb_name* for the database name, where *pdb_name* is the name you chose for your PDB.
- You can't convert a CDB in the multi-tenant configuration to a CDB in the single-tenant conversion. Conversion to the multi-tenant configuration is one-way and irreversible.
- You can't enable or convert to the multi-tenant configuration if your DB instance uses an Oracle database release lower than 19.0.0.0.ru-2022-01.rur-2022.r1.
- You can't use Database Activity Streams in a CDB.
- You can't enable auditing from within CDB$R00T. You must enable auditing within each PDB individually.

Tenant database (PDB) limitations

The following limitations apply to tenant databases in the RDS for Oracle multi-tenant configuration:

- You can't defer tenant database operations to the maintenance window. All changes occur immediately.
- You can't add a tenant database to a CDB that uses the single-tenant configuration.
- You can't add or modify multiple tenant databases in a single operation. You can only add or modify them one at a time.
- You can't modify a tenant database to be named `CDB$ROOT` or `PDB$SEED`.
- You can't delete a tenant database if it is the only tenant in the CDB.
- Not all DB instance class types have sufficient resources to support multiple PDBs in an RDS for Oracle CDB instance. An increased PDB count affects the performance and stability of the smaller instance classes and increases the time of most instance-level operations, for example, database upgrades.
- You cannot rename a PDB using `rdsadmin.rdsadmin_util.rename_global_name`, You must use the `modify-tenant-database` API instead.
- You can't use multiple AWS accounts to create PDBs in the same CDB. PDBs must be owned by the same account as the DB instance that the PDBs are hosted on.
- All PDBs in a CDB use the same endpoint and database listener.
- The following operations aren't supported at the PDB level but are supported at the CDB level:
  - Backup and recovery
  - Database upgrades
  - Maintenance actions
- The following features aren't supported at the PDB level but are supported at the CDB level:
  - Option groups (options are installed on all PDBs on your CDB instance)
  - Parameter groups (all parameters are derived from the parameter group associated with your CDB instance)
- PDB-level operations that are supported in the on-premises CDB architecture but aren't supported in an RDS for Oracle CDB include the following:

> **Note**
>
> The following list is not exhaustive.

  - Application PDBs

- Proxy PDBs
- Starting and stopping a PDB
- Unplugging and plugging in PDBs

  To move data into or out of your CDB, use the same techniques as for a non-CDB. For more information about migrating data, see Importing data into Oracle on Amazon RDS.
- Setting options at the PDB level

  The PDB inherits options settings from the CDB option group. For more information about setting options, see Parameter groups for Amazon RDS. For best practices, see Working with DB parameter groups.
- Configuring parameters in a PDB

  The PDB inherits parameter settings from the CDB. For more information about setting option, see Adding options to Oracle DB instances.
- Configuring different listeners for PDBs in the same CDB
- Oracle Flashback features

Configuring an RDS for Oracle CDB

Configuring a CDB is similar to configuring a non-CDB.

Topics

- Creating an RDS for Oracle CDB instance
- Connecting to a PDB in your RDS for Oracle CDB

Creating an RDS for Oracle CDB instance

In RDS for Oracle, creating a CDB instance is almost identical to creating a non-CDB instance. The difference is that you choose the Oracle multitenant architecture when creating your DB instance and also choose an architecture configuration: multi-tenant or single-tenant. If you create tags when you create a CDB in the multi-tenant configuration, RDS propagates the tags to the initial tenant database. To create a CDB, use the AWS Management Console, the AWS CLI, or the RDS API.

Console

To create a CDB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the CDB instance.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.
5. In **Choose a database creation method**, select **Standard Create**.
6. In **Engine options**, choose **Oracle**.
7. For **Database management type**, choose **Amazon RDS**.
8. For **Architecture settings**, choose **Oracle multitenant architecture**.
9. For **Architecture configuration**, do either of the following:
   - Choose **Multi-tenant configuration** and proceed to the next step.
   - Choose **Single-tenant configuration** and skip to Step 11.
10. (Multi-tenant configuration) For **Tenant database settings**, make the following changes:
    - For **Tenant database name**, enter the name of your initial PDB. The PDB name must be different from the CDB name, which defaults to RDSCDB.
    - For **Tenant database master username**, enter the master username of your PDB. You can't use the tenant database master username to log in to the CDB itself.
    - For **Credentials management**, choose either of the following credentials management options:
      - **Managed in AWS Secrets Manager**

        The managed password is for the initial tenant database rather than for the instance. In **Select the encryption key**, choose either a KMS key that Secrets Manager creates or a key that you have created.

> **Note**
>
> We recommend AWS Secrets Manager as the most secure technique for managing credentials. Additional charges apply. For more information, see Password management with Amazon RDS and AWS Secrets Manager.

- **Self managed**

  To specify a password, clear the **Auto generate a password** check box if it is selected. Enter the same password in **Master password** and **Confirm master password**.

- For **Tenant database character set**, choose a character set for the PDB. You can choose a tenant database character set that is different from the CDB character set.

  The default PDB character set is **AL32UTF8**. If you choose a nondefault PDB character set, CDB creation might be slower.

> **Note**
>
> You can't specify multiple tenant databases in the create operation. The CDB has one PDB when it is created. You can add PDBs to an existing CDB in a separate operation.

11. (Single-tenant configuration) Choose the settings that you want based on the options listed in Settings for DB instances:

    - In the **Settings** section, open **Credential Settings**. Then do the following:

      i. For **Master username**, enter the name for a local user in your PDB. You can't use the master username to log in to the CDB root.

      ii. For **Credentials management**, choose either of the following credentials management options:

      - **Managed in AWS Secrets Manager**

        In **Select the encryption key**, choose either a KMS key that Secrets Manager creates or a key that you have created.

> **Note**
>
> We recommend AWS Secrets Manager as the most secure technique for managing credentials. Additional charges apply. For more information, see Password management with Amazon RDS and AWS Secrets Manager.

- **Self managed**

  To specify a password, clear the **Auto generate a password** check box if it is selected. Enter the same password in **Master password** and **Confirm master password**.

12. For the remaining sections, specify your DB instance settings. For information about each setting, see Settings for DB instances.
13. Choose **Create database**.

AWS CLI

To create a CDB in the multi-tenant configuration, use the create-db-instance command with the following parameters:

- `--db-instance-identifier`
- `--db-instance-class`
- `--engine { oracle-ee-cdb | oracle-se2-cdb }`
- `--master-username`
- `--master-user-password` or `--manage-master-user-password`
- `--multi-tenant` (for the single-tenant configuration, either don't specify `multi-tenant` or specify `--no-multi-tenant`)
- `--allocated-storage`
- `--backup-retention-period`

For information about each setting, see Settings for DB instances.

This following example creates an RDS for Oracle DB instance named *`my-cdb-inst`* in the multi-tenant configuration. If you specify `--no-multi-tenant` or don't specify `--multi-tenant`,

the default CDB configuration is single-tenant. The engine is `oracle-ee-cdb`: a command that specifies `oracle-ee` and `--multi-tenant` fails with an error. The initial tenant database is named *mypdb*.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --engine oracle-ee-cdb \
    --db-instance-identifier my-cdb-inst \
    --multi-tenant \
    --db-name mypdb \
    --allocated-storage 250 \
    --db-instance-class db.t3.large \
    --master-username pdb_admin \
    --manage-master-user-password \
    --backup-retention-period 3
```

For Windows:

```
aws rds create-db-instance ^
    --engine oracle-ee-cdb ^
    --db-instance-identifier my-cdb-inst ^
    --multi-tenant ^
    --db-name mypdb ^
    --allocated-storage 250 ^
    --db-instance-class db.t3.large ^
    --master-username pdb_admin ^
    --manage-master-user-password \ ^
    --backup-retention-period 3
```

Note

Specify a password other than the prompt shown here as a security best practice.

This command produces output similar to the following. The database name, character set, national character set, master user, and master user secret aren't included in the output. You can view this information by using the CLI command `describe-tenant-databases`.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-cdb-inst",
        "DBInstanceClass": "db.t3.large",
        "MultiTenant": true,
        "Engine": "oracle-ee-cdb",
        "DBResourceId": "db-ABCDEFGJIJKLMNOPQRSTUVWXYZ",
        "DBInstanceStatus": "creating",
        "AllocatedStorage": 250,
        "PreferredBackupWindow": "04:59-05:29",
        "BackupRetentionPeriod": 3,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-0a1bcd2e",
                "Status": "active"
            }
        ],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.oracle-ee-cdb-19",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-1234567a",
            "SubnetGroupStatus": "Complete",
            ...
```

RDS API

To create a DB instance by using the Amazon RDS API, call the CreateDBInstance operation.

For information about each setting, see Settings for DB instances.

Connecting to a PDB in your RDS for Oracle CDB

You can use a utility like SQL*Plus to connect to a PDB. To download Oracle Instant Client, which includes a standalone version of SQL*Plus, see Oracle Instant Client Downloads.

To connect SQL*Plus to your PDB, you need the following information:

- PDB name
- Database user name and password
- Endpoint for your DB instance
- Port number

For information about finding the preceding information, see Finding the endpoint of your RDS for Oracle DB instance.

Example To connect to your PDB using SQL*Plus

In the following examples, substitute your master user for *master_user_name*. Also, substitute the endpoint for your DB instance, and then include the port number and the Oracle SID. The SID value is the name of the PDB that you specified when you created your DB instance, and not the DB instance identifier.

For Linux, macOS, or Unix:

```
sqlplus 'master_user_name@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=endpoint)
(PORT=port))(CONNECT_DATA=(SID=pdb_name)))'
```

For Windows:

```
sqlplus master_user_name@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=endpoint)
(PORT=port))(CONNECT_DATA=(SID=pdb_name)))
```

You should see output similar to the following.

```
SQL*Plus: Release 19.0.0.0.0 Production on Mon Aug 21 09:42:20 2021
```

After you enter the password for the user, the SQL prompt appears.

```
SQL>
```

> **Note**
>
> The shorter format connection string (Easy connect or EZCONNECT), such as `sqlplus username/password@LONGER-THAN-63-CHARS-RDS-ENDPOINT-`

HERE:1521/*database-identifier*, might encounter a maximum character limit and should not be used to connect.

Backing up and restoring a CDB

You can back up and restore your CDB using either RDS DB snapshots or Recovery Manager (RMAN).

Backing up and restoring a CDB using DB snapshots

DB snapshots work similarly in the CDB and non-CDB architectures. The principal differences are as follows:

- When you restore a DB snapshot of a CDB, you can't rename the CDB. The CDB is named RDSCDB and can't be changed.
- When you restore a DB snapshot of a CDB, you can't rename PDBs. You can modify the PDB name by using the modify-tenant-database command.
- To find tenant databases in a snapshot, use the CLI command describe-db-snapshot-tenant-databases.
- You can't directly interact with the tenant databases in a CDB snapshot that uses the multi-tenant architecture configuration. If you restore the DB snapshot, you restore all its tenant databases.
- RDS for Oracle implicitly copies tags on a tenant database to the tenant database in a DB snapshot. When you restore a tenant database, the tags appear in the restored database.
- If you restore a DB snapshot and specify new tags using the `--tags` parameter, the new tags overwrite all existing tags.
- If you take a DB snapshot of a CDB instance that has tags, and you specify `--copy-tags-to-snapshot`, RDS for Oracle copies tags from the tenant databases to the tenant databases in the snapshot.

For more information, see Oracle Database considerations.

Backing up and restoring a CDB using RMAN

To learn how to back up and restore a CDB or individual tenant database using RMAN, see Performing common RMAN tasks for Oracle DB instances.

Converting an RDS for Oracle non-CDB to a CDB

You can change the architecture of an Oracle database from the non-CDB architecture to the Oracle multitenant architecture, also called the *CDB architecture*, with the `modify-db-instance` command. In most cases, this technique is preferable to creating a new CDB and importing data. The conversion operation incurs downtime.

When you upgrade your database engine version, you can't change the database architecture in the same operation. Therefore, to upgrade an Oracle Database 19c non-CDB to an Oracle Database 21c CDB, you first need to convert the non-CDB to a CDB in one step, and then upgrade the 19c CDB to a 21c CDB in a separate step.

The non-CDB conversion operation has the following requirements:

- You must specify `oracle-ee-cdb` or `oracle-se2-cdb` for the DB engine type. These are the only supported values.
- Your DB engine must use Oracle Database 19c with an April 2021 or later release update (RU).

The operation has the following limitations:

- You can't convert a CDB to a non-CDB. You can only convert a non-CDB to a CDB.
- You can't convert a non-CDB to the multi-tenant configuration in a single `modify-db-instance` call. After you convert a non-CDB to a CDB, your CDB is in the single-tenant configuration. To convert the single-tenant configuration to the multi-tenant configuration, run `modify-db-instance` again. For more information, see Converting the single-tenant configuration to multi-tenant.
- You can't convert a primary or replica database that has Oracle Data Guard enabled. To convert a non-CDB that has read replicas, first delete all read replicas.
- You can't upgrade the DB engine version and convert a non-CDB to a CDB in the same operation.

Before converting your non-CDB, consider the following:

- The considerations for option and parameter groups are the same as for upgrading the DB engine. For more information, see Considerations for Oracle database upgrades.
- You can convert existing non-CDB instances that uses managed master passwords to single-tenant instances in a single operation. The single-tenant instances inherit the managed passwords.

- If your DB instance has the OEMAGENT option installed, a best practice is to remove this option before you convert your non-CDB. After your non-CDB is converted to a CDB, reinstall the option. For more information, see Oracle Management Agent for Enterprise Manager Cloud Control.
- During the conversion process, RDS resets the online redo log size to the default 128M.

Console

To convert a non-CDB to a CDB

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region where your DB instance resides.
3. In the navigation pane, choose **Databases**, and then choose the non-CDB instance that you want to convert to a CDB instance.
4. Choose **Modify**.
5. For **Architecture settings**, select **Oracle multitenant architecture**. After conversion, your CDB will be in the single-tenant configuration.
6. (Optional) For **DB parameter group**, choose a new parameter group for your CDB instance. The same parameter group considerations apply when converting a DB instance as when upgrading a DB instance. For more information, see Parameter group considerations.
7. (Optional) For **Option group**, choose a new option group for your CDB instance. The same option group considerations apply when converting a DB instance as when upgrading a DB instance. For more information, see Option group considerations.
8. (Optional) For **Credentials management**, choose **Managed in AWS Secrets Manager** or **Self-managed**. For more information, see Managing the master user password for a DB instance with Secrets Manager.
9. When all the changes are as you want them, choose **Continue** and check the summary of modifications.
10. (Optional) Choose **Apply immediately** to apply the changes immediately. Choosing this option can cause downtime in some cases. For more information, see Using the schedule modifications setting.
11. On the confirmation page, review your changes. If they are correct, choose **Modify DB instance**.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To convert the non-CDB on your DB instance to a CDB in the single-tenant configuration, set `--engine` to `oracle-ee-cdb` or `oracle-se2-cdb` in the AWS CLI command modify-db-instance. For more information, see Settings for DB instances.

The following example converts the DB instance named *my-non-cdb* and specifies a custom option group and parameter group. The command also enables password management with Secrets Manager.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my-non-cdb \
    --engine oracle-ee-cdb \
    --option-group-name custom-option-group \
    --db-parameter-group-name custom-parameter-group \
    --manage-master-user-password
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my-non-cdb ^
    --engine oracle-ee-cdb ^
    --option-group-name custom-option-group ^
    --db-parameter-group-name custom-parameter-group ^
    --manage-master-user-password
```

RDS API

To convert a non-CDB to a CDB, specify `Engine` in the RDS API operation ModifyDBInstance.

Converting the single-tenant configuration to multi-tenant

You can modify the architecture of an RDS for Oracle CDB from the single-tenant configuration to the multi-tenant configuration. Before and after the conversion, your CDB contains a single tenant

database (PDB). Tags for the DB instance propagate to the initial tenant database created during the conversion.

Before you begin, make sure that your IAM policy has permission to create a tenant database. During the conversion, RDS for Oracle migrates the following metadata to the new tenant database:

- The master username
- The managed master password (if the source CDB integrates with Secrets Manager)
- The database name
- The character set
- The national character set

Before the conversion, you view the preceding information by using the `describe-db-instances` command. After the conversion, you view the information by using the `describe-tenant-database` command.

The conversion from single-tenant to multi-tenant has the following limitations:

- You can't later convert back to the single-tenant configuration after you convert to the multi-tenant configuration. The conversion is irreversible.
- You can't convert a primary or replica database that has Oracle Data Guard enabled.
- You can't upgrade the DB engine version and convert to the multi-tenant configuration in the same operation.
- You can't enable or disable managed master user passwords during the conversion.

Console

To convert a CDB using the single-tenant configuration to the multi-tenant configuration

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region where your DB instance resides.
3. In the navigation pane, choose **Databases**, and then choose the non-CDB instance that you want to convert to a CDB instance.

4. Choose **Modify**.
5. For **Architecture settings**, select **Oracle multitenant architecture**.
6. For **Architecture configuration**, select **Multi-tenant configuration**.
7. (Optional) For **DB parameter group**, choose a new parameter group for your CDB instance. The same parameter group considerations apply when converting a DB instance as when upgrading a DB instance.
8. (Optional) For **Option group**, choose a new option group for your CDB instance. The same option group considerations apply when converting a DB instance as when upgrading a DB instance.
9. When all the changes are as you want them, choose **Continue** and check the summary of modifications.
10. Choose **Apply immediately**. This option is required when you switch to a multi-tenant configuration. Note that this option can cause downtime in some cases.
11. On the confirmation page, review your changes. If they are correct, choose **Modify DB instance**.

    Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To convert a CDB using the single-tenant configuration to the multi-tenant configuration, specify `--multi-tenant` in the AWS CLI command modify-db-instance.

The following example converts the DB instance named `my-st-cdb` from the single-tenant configuration to the multi-tenant configuration. The `--apply-immediately` option is required.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance --region us-east-1\
    --db-instance-identifier my-st-cdb \
    --multi-tenant \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance --region us-east-1 ^
```

```
    --db-instance-identifier my-st-cdb ^
    --multi-tenant ^
    --apply-immediately
```

The output looks something like the following.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-st-cdb",
        "DBInstanceClass": "db.r5.large",
        "MultiTenant": false,
        "Engine": "oracle-ee-cdb",
        "DBResourceId": "db-AB1CDE2FGHIJK34LMNOPRLXTXU",
        "DBInstanceStatus": "modifying",
        "MasterUsername": "admin",
        "DBName": "ORCL",
        ...
        "EngineVersion": "19.0.0.0.ru-2022-01.rur-2022-01.r1",
        "AutoMinorVersionUpgrade": true,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "bring-your-own-license",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "default:oracle-ee-cdb-19",
                "Status": "in-sync"
            }
        ],
        ...
        "PendingModifiedValues": {
            "MultiTenant": "true"
        }
    }
}
```

Adding an RDS for Oracle tenant database to your CDB instance

In the RDS for Oracle multi-tenant configuration, a tenant database is a PDB. To add a tenant database, make sure you meet the following prerequisites:

- Your CDB has the multi-tenant configuration enabled. For more information, see Multi-tenant configuration of the CDB architecture.
- You have the necessary IAM permissions to create the tenant database.

You can add a tenant database using the AWS Management Console, the AWS CLI, or the RDS API. You can't add multiple tenant databases in a single operation: you must add them one at a time. If the CDB has backup retention enabled, Amazon RDS backs up the DB instance before and after it adds a new tenant database. If the CDB has read replicas, you can only add a tenant database to the primary DB instance; Amazon RDS automatically creates the tenant database on the replicas. Replication health is also validated, ensuring all replicas are available and replication lag is less than 5 minutes before the tenant is created.

Console

To add a tenant database to your DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the tenant database.
3. In the navigation pane, choose **Databases**.
4. Choose the CDB instance to which you want to add a tenant database. Your DB instance must use the multi-tenant configuration of the CDB architecture.
5. Choose **Actions** and then **Add tenant database**.
6. For **Tenant database settings**, do the following:

   - For **Tenant database name**, enter the name of your new PDB.
   - For **Tenant database master username**, enter the name of the master user for your PDB.
   - Choose either of the following credentials management options:
     - **Managed in AWS Secrets Manager**

       In **Select the encryption key**, choose either a KMS key that Secrets Manager creates or a key that you have created.

       > **Note**
       >
       > We recommend AWS Secrets Manager as the most secure technique for managing credentials. Additional charges apply. AWS Secrets Manager is not supported for instances using read replicas. For more information, see Password management with Amazon RDS and AWS Secrets Manager.

- **Self managed**

  To specify a password, clear the **Auto generate a password** check box if it is selected. Enter the same password in **Master password** and **Confirm master password**.

- Under **Additional configuration**, enter the name of your PDB for **Initial database name**. You can't name the CDB, which has the default name RDSCDB.

- For **Tenant database character set**, choose a character set for the PDB. The default is **AL32UTF8**. You can choose a PDB character set that is different from the CDB character set. If the instance has read replicas, tenants cannot be created with a custom character set. You can create your tenants with a custom character set before creating a read replica if needed.
- For **Tenant database national character set**, choose a national character set for the PDB. The default is **AL32UTF8**. The national character set specifies the encoding only for columns that use the NCHAR data type (NCHAR, NVARCHAR2, and NCLOB) and doesn't affect database metadata.

For more information about the preceding settings, see Settings for DB instances.

7. Choose **Add tenant**.

AWS CLI

To add a tenant database to your CDB with the AWS CLI, use the command create-tenant-database with the following required parameters:

- `--db-instance-identifier`
- `--tenant-db-name`
- `--master-username`
- `--master-user-password`

This following example creates a tenant database named *mypdb2* in the RDS for Oracle CDB instance named *my-cdb-inst*. The PDB character set is UTF-16.

Example

For Linux, macOS, or Unix:

```
aws rds create-tenant-database --region us-east-1 \
```

```
    --db-instance-identifier my-cdb-inst \
    --tenant-db-name mypdb2 \
    --master-username mypdb2-admin \
    --master-user-password mypdb2-pwd \
    --character-set-name UTF-16
```

For Windows:

```
aws rds create-tenant-database --region us-east-1 \
    --db-instance-identifier my-cdb-inst ^
    --tenant-db-name mypdb2 ^
    --master-username mypdb2-admin ^
    --master-user-password mypdb2-pwd ^
    --character-set-name UTF-16
```

The output looks similar to the following.

```
...}
    "TenantDatabase" :
         {
            "DbiResourceId" : "db-abc123",
            "TenantDatabaseResourceId" : "tdb-bac567",
            "TenantDatabaseArn" : "arn:aws:rds:us-east-1:123456789012:db:my-cdb-
inst:mypdb2",
            "DBInstanceIdentifier" : "my-cdb-inst",
            "TenantDBName" : "mypdb2",
            "Status" : "creating",
            "MasterUsername" : "mypdb2",
            "CharacterSetName" : "UTF-16",
            ...
        }
}...
```

Modifying an RDS for Oracle tenant database

You can modify only the PDB name and the master user password of a tenant database in your CDB. Note the following requirements and limitations:

- To modify the settings of a tenant database in your DB instance, the tenant database must exist.
- You can't modify multiple tenant databases in a single operation. You can only modify one tenant database at a time.

- You can't change the name of a tenant database to `CDB$ROOT` or `PDB$SEED`.
- If your DB instance has read replicas, you can only modify tenants on the primary DB instance. Replication health is also validated, ensuring the replicas are available and replication lag is less than 5 minutes before the tenant is modified.

You can modify PDBs using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To modify the PDB name or master password of a tenant database

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the tenant database.
3. In the navigation pane, choose **Databases**.
4. Choose the tenant database whose database name or master user password you want to modify.
5. Choose **Modify**.
6. For **Tenant database settings**, do any of the following:
   - For **Tenant database name**, enter the new name of your new PDB.
   - For **Tenant database master password**, enter a new password.
7. Choose **Modify tenant**.

AWS CLI

To modify a tenant database using the AWS CLI, call the modify-tenant-database command with the following parameters:

- `--db-instance-identifier` *value*
- `--tenant-db-name` *value*
- `[--new-tenant-db-name` *value*`]`
- `[--master-user-password` *value*`]`

The following example renames tenant database pdb1 to pdb-hr on DB instance my-cdb-inst.

Example

For Linux, macOS, or Unix:

```
aws rds modify-tenant-database --region us-east-1 \
    --db-instance-identifier my-cdb-inst \
    --tenant-db-name pdb1 \
    --new-tenant-db-name pdb-hr
```

For Windows:

```
aws rds modify-tenant-database --region us-east-1 ^
    --db-instance-identifier my-cdb-inst ^
    --tenant-db-name pdb1 ^
    --new-tenant-db-name pdb-hr
```

This command produces output similar to the following.

```
{
    "TenantDatabase" : {
        "DbiResourceId" : "db-abc123",
        "TenantDatabaseResourceId" : "tdb-bac567",
        "TenantDatabaseArn" : "arn:aws:rds:us-east-1:123456789012:db:my-cdb-inst:pdb1",
        "DBInstanceIdentifier" : "my-cdb-inst",
        "TenantDBName" : "pdb1",
        "Status" : "modifying",
        "MasterUsername" : "tenant-admin-user"
        "Port" : "6555",
        "CharacterSetName" : "UTF-16",
        "MaxAllocatedStorage" : "1000",
        "ParameterGroups": [
            {
                "ParameterGroupName": "pdb1-params",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "pdb1-options",
                "Status": "in-sync"
```

```
            }
        ],
        "PendingModifiedValues": {
            "TenantDBName": "pdb-hr"
        }
    }
}
```

Deleting an RDS for Oracle tenant database from your CDB

You can delete a tenant database (PDB) using the AWS Management Console, the AWS CLI, or the RDS API. Consider the following prerequisites and limitations:

- The tenant database and DB instance must exist.
- For the deletion to succeed, one of the following situations must exist:
  - The tenant database and DB instance are available.

    > **Note**
    >
    > You can take a final snapshot, but only if the tenant database and DB instance were in an available state before you issued the `delete-tenant-database` command. This snapshot will only be taken on the primary instance if the DB instance has read replicas.

  - The tenant database is being created.
  - The DB instance is modifying the tenant database.
  - If the DB instance has read replicas these constraints apply to all replicas.
- You can't delete multiple tenant databases in a single operation.
- You can't delete a tenant database if it is the only tenant in the CDB.
- You can't delete a tenant database on a read replica, you can only delete a tenant on the primary DB instance. Replication health is also validated, ensuring the replication lag is less than 5 minutes before the tenant is deleted.

Console

To delete a tenant database

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the tenant database that you want to delete.
3. For **Actions**, choose **Delete**.
4. To create a final DB snapshot for the DB instance, choose **Create final snapshot?**.
5. If you chose to create a final snapshot, enter the **Final snapshot name**.
6. Enter **`delete me`** in the box.
7. Choose **Delete**.

AWS CLI

To delete a tenant database using the AWS CLI, call the delete-tenant-database command with the following parameters:

- `--db-instance-identifier` *value*
- `--tenant-db-name` *value*
- `[--skip-final-snapshot | --no-skip-final-snapshot]`
- `[--final-snapshot-identifier` *value*`]`

This following example deletes the tenant database named *pdb-test* from the CDB named *my-cdb-inst*. By default, the operation creates a final snapshot.

Example

For Linux, macOS, or Unix:

```
aws rds delete-tenant-database --region us-east-1 \
    --db-instance-identifier my-cdb-inst \
    --tenant-db-name pdb-test \
    --final-snapshot-identifier final-snap-pdb-test
```

For Windows:

```
aws rds delete-tenant-database --region us-east-1 ^
    --db-instance-identifier my-cdb-inst ^
    --tenant-db-name pdb-test ^
    --final-snapshot-identifier final-snap-pdb-test
```

This command produces output similar to the following.

```
{
    "TenantDatabase" : {
        "DbiResourceId" : "db-abc123",
        "TenantDatabaseResourceId" : "tdb-bac456",
        "TenantDatabaseArn" : "arn:aws:rds:us-east-1:123456789012:db:my-cdb-inst:pdb-
test",
        "DBInstanceIdentifier" : "my-cdb-inst",
        "TenantDBName" : "pdb-test",
        "Status" : "deleting",
        "MasterUsername" : "pdb-test-admin"
        "Port" : "6555",
        "CharacterSetName" : "UTF-16",
        "MaxAllocatedStorage" : "1000",
        "ParameterGroups": [
            {
                "ParameterGroupName": "tenant-1-params",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "tenant-1-options",
                "Status": "in-sync"
            }
        ]
    }
}
```

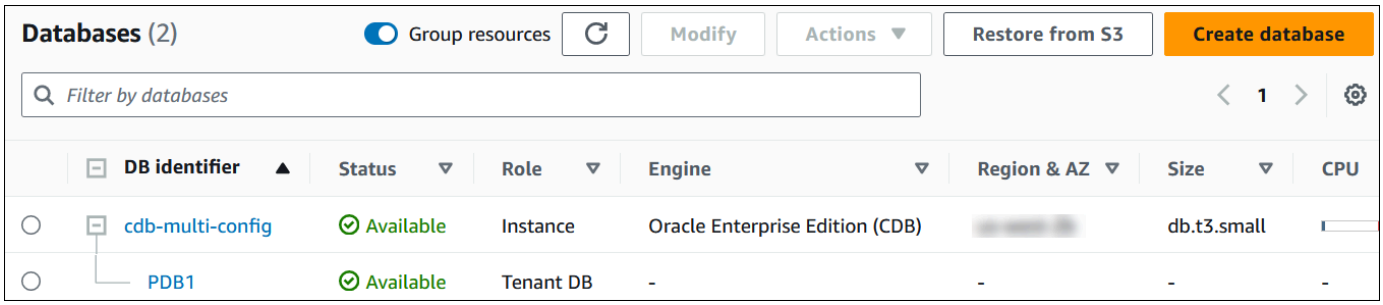
Viewing tenant database details

You can view details about a tenant database in the same way that you can for a non-CDB or CDB.

Console

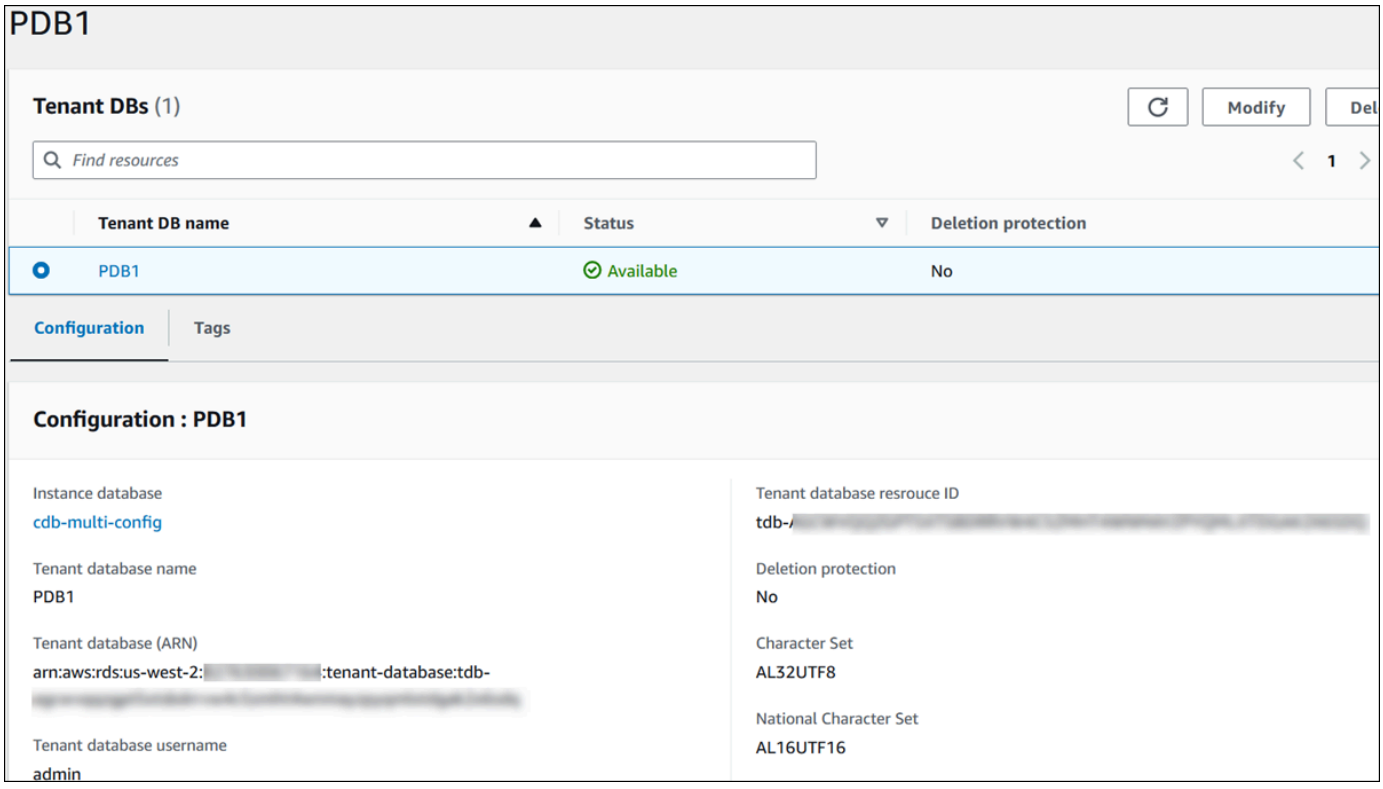
To view details about a tenant database

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region where your DB instance resides.
3. In the navigation pane, choose **Databases**.



In the preceding image, the sole tenant database (PDB) appears as a child of the DB instance.

4. Choose the name of a tenant database.



AWS CLI

To see details about your PDBs, use the AWS CLI command describe-tenant-databases.

This following example describes all tenant databases in the specified Region.

Example

For Linux, macOS, or Unix:

```
aws rds describe-tenant-databases --region us-east-1
```

For Windows:

```
aws rds describe-tenant-databases --region us-east-1
```

This command produces output similar to the following.

```
    "TenantDatabases" : [
         {
            "DBInstanceIdentifier" : "my-cdb-inst",
            "TenantDBName" : "pdb-test",
            "Status" : "available",
            "MasterUsername" : "pdb-test-admin",
            "DbiResourceId" : "db-abc123",
            "TenantDatabaseResourceId" : "tdb-bac456",
            "TenantDatabaseArn" : "arn:aws:rds:us-east-1:123456789012:db:my-cdb-
inst:pdb-test",
            "CharacterSetName": "AL32UTF8",
            "NcharCharacterSetName": "AL16UTF16",
            "DeletionProtection": false,
            "PendingModifiedValues": {
                 "MasterUserPassword": "****"
            },
            "TagList": []
         },
         {

            "DBInstanceIdentifier" : "my-cdb-inst2",
            "TenantDBName" : "pdb-dev",
            "Status" : "modifying",
            "MasterUsername" : "masterrdsuser"
```

```
            "DbiResourceId" : "db-xyz789",
            "TenantDatabaseResourceId" : "tdb-ghp890",
            "TenantDatabaseArn" : "arn:aws:rds:us-east-1:123456789012:db:my-cdb-
inst2:pdb-dev",
            "CharacterSetName": "AL32UTF8",
            "NcharCharacterSetName": "AL16UTF16",
            "DeletionProtection": false,
            "PendingModifiedValues": {
                 "MasterUserPassword": "****"
            },
            "TagList": []
         },
         ... other truncated data
```

The following example describes the tenant databases on DB instance `my-cdb-inst` in the specified Region.

Example

For Linux, macOS, or Unix:

```
aws rds describe-tenant-databases --region us-east-1 \
    --db-instance-identifier my-cdb-inst
```

For Windows:

```
aws rds describe-tenant-databases --region us-east-1 ^
    --db-instance-identifier my-cdb-inst
```

This command produces output similar to the following.

```
{
    "TenantDatabase": {
        "TenantDatabaseCreateTime": "2023-10-19T23:55:30.046Z",
        "DBInstanceIdentifier": "my-cdb-inst",
        "TenantDBName": "pdb-hr",
        "Status": "creating",
        "MasterUsername": "tenant-admin-user",
        "DbiResourceId": "db-abc123",
        "TenantDatabaseResourceId": "tdb-bac567",
        "TenantDatabaseARN": "arn:aws:rds:us-west-2:579508833180:pdb-hr:tdb-
abcdefghi1jklmno2p3qrst4uvw5xy6zabc7defghi8jklmn90op",
```

```
        "CharacterSetName": "AL32UTF8",
        "NcharCharacterSetName": "AL16UTF16",
        "DeletionProtection": false,
        "PendingModifiedValues": {
            "MasterUserPassword": "****"
        },
        "TagList": [
            {
                "Key": "TEST",
                "Value": "testValue"
            }
        ]
    }
}
```

The following example describes tenant database pdb1 on DB instance `my-cdb-inst` in the US East (N. Virginia) Region.

Example

For Linux, macOS, or Unix:

```
aws rds describe-tenant-databases --region us-east-1 \
--db-instance-identifier my-cdb-inst \
--tenant-db-name pdb1
```

For Windows:

```
aws rds describe-tenant-databases --region us-east-1 ^
--db-instance-identifier my-cdb-inst ^
--tenant-db-name pdb1
```

This command produces output similar to the following.

```
{
    "TenantDatabases" : [
        {
            "DbiResourceId" : "db-abc123",
            "TenantDatabaseResourceId" : "tdb-bac567",
            "TenantDatabaseArn" : "arn:aws:rds:us-east-1:123456789012:db:my-cdb-
inst:pdb1"
            "DBInstanceIdentifier" : "my-cdb-inst",
```

```
            "TenantDBName" : "pdb1",
            "Status" : "ACTIVE",
            "MasterUsername" : "masterawsuser"
            "Port" : "1234",
            "CharacterSetName": "UTF-8",
            "ParameterGroups": [
                {
                    "ParameterGroupName": "tenant-custom-pg",
                    "ParameterApplyStatus": "in-sync"
                }
            ],
            {
            "OptionGroupMemberships": [
                {
                    "OptionGroupName": "tenant-custom-og",
                    "Status": "in-sync"
                }
            ]
        }
    ]
}
```

Upgrading your CDB

You can upgrade a CDB to a different Oracle Database release. For example, you can upgrade an Oracle Database 19c CDB to an Oracle Database 21c CDB. You can't change the database architecture during an upgrade. Thus, you can't upgrade a non-CDB to a CDB or upgrade a CDB to a non-CDB.

The procedure for upgrading a CDB to a CDB is the same as for upgrading a non-CDB to a non-CDB. For more information, see Upgrading the RDS for Oracle DB engine.

Administering your RDS for Oracle DB instance

Following are the common management tasks that you perform with an RDS for Oracle DB instance. Some tasks are the same for all RDS DB instances. Other tasks are specific to RDS for Oracle.

The following tasks are common to all RDS databases, but Oracle Database has special considerations. For example, you connect to an Oracle database using the Oracle clients SQL*Plus and SQL Developer.

Task area	Relevant documentation
Instance classes, storage, and PIOPS If you are creating a production instance, learn how instance classes, storage types, and Provisioned IOPS work in Amazon RDS.	RDS for Oracle DB instance classes Amazon RDS storage types
Multi-AZ deployments A production DB instance should use Multi-AZ deployments. Multi-AZ deployments provide increased availability, data durability, and fault tolerance for DB instances.	Configuring and managing a Multi-AZ deployment for Amazon RDS
Amazon VPC If your AWS account has a default virtual private cloud (VPC), then your DB instance is automatically created inside the default VPC. If your account doesn't have a default VPC, and you want the DB instance in a VPC, create the VPC and subnet groups before you create the instance.	Working with a DB instance in a VPC
Security groups By default, DB instances use a firewall that prevents access. Make sure that you create a security group with the correct IP addresses and network configuration to access the DB instance.	Controlling access with security groups
Parameter groups	Parameter groups for Amazon RDS

Task area	Relevant documentation
If your DB instance is going to require specific database parameters, create a parameter group before you create the DB instance.	
Option groups If your DB instance requires specific database options, create an option group before you create the DB instance.	Adding options to Oracle DB instances
Connecting to your DB instance After creating a security group and associating it to a DB instance, you can connect to the DB instance using any standard SQL client application such as Oracle SQL*Plus.	Connecting to your Oracle DB instance
Backup and restore You can configure your DB instance to take automated backups, or take manual snapshots, and then restore instances from the backups or snapshots.	Backing up, restoring, and exporting data
Monitoring You can monitor an Oracle DB instance by using CloudWatch Amazon RDS metrics, events, and enhanced monitoring.	Viewing metrics in the Amazon RDS console Viewing Amazon RDS events
Log files You can access the log files for your Oracle DB instance.	Monitoring Amazon RDS log files

Following, you can find a description for Amazon RDS–specific implementations of common DBA tasks for RDS Oracle. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. Also, RDS restricts access to certain system procedures and tables that require advanced privileges. In many of the tasks, you run the `rdsadmin` package, which is an Amazon RDS–specific tool that enables you to administer your database.

The following are common DBA tasks for DB instances running Oracle:

- System tasks

Disconnecting a session	Amazon RDS method: `rdsadmin.rdsadmin_util.disconnect` Oracle method: `alter system disconnect session`
Terminating a session	Amazon RDS method: `rdsadmin.rdsadmin_util.kill` Oracle method: `alter system kill session`
Canceling a SQL statement in a session	Amazon RDS method: `rdsadmin.rdsadmin_util.cancel` Oracle method: `alter system cancel sql`
Enabling and disabling restricted sessions	Amazon RDS method: `rdsadmin.rdsadmin_util.restricted_session` Oracle method: `alter system enable restricted session`
Flushing the shared pool	Amazon RDS method: `rdsadmin.rdsadmin_util.flush_shared_pool` Oracle method: `alter system flush shared_pool`
Flushing the buffer cache	Amazon RDS method: `rdsadmin.rdsadmin_util.flush_buffer_cache` Oracle method: `alter system flush buffer_cache`
Granting SELECT or EXECUTE privileges to SYS objects	Amazon RDS method: `rdsadmin.rdsadmin_util.grant_sys_object` Oracle method: `grant`
Revoking SELECT or EXECUTE privileges on SYS objects	Amazon RDS method: `rdsadmin.rdsadmin_util.revoke_sys_object` Oracle method: `revoke`

Managing RDS_X$ views for Oracle DB instances	Amazon RDS method: `rdsadmin.rdsadmin_util.create_sys_x$_view` Oracle method: `CREATE VIEW`
Granting privileges to non-master users	Amazon RDS method: `grant`
Creating custom functions to verify passwords	Amazon RDS method: `rdsadmin.rdsadmin_password_verify.create_verify_function` Amazon RDS method: `rdsadmin.rdsadmin_password_verify.create_passthrough_verify_fcn`
Setting up a custom DNS server	—
Listing allowed system diagnostic events	Amazon RDS method: `rdsadmin.rdsadmin_util.list_allowed_system_events` Oracle method: —
Setting system diagnostic events	Amazon RDS method: `rdsadmin.rdsadmin_util.set_allowed_system_events` Oracle method: `ALTER SYSTEM SET EVENTS 'set_event_clause'`
Listing system diagnostic events that are set	Amazon RDS method: `rdsadmin.rdsadmin_util.list_set_system_events` Oracle method: `ALTER SESSION SET EVENTS 'IMMEDIATE EVENTDUMP(SYSTEM)'`
Unsetting system diagnostic events	Amazon RDS method: `rdsadmin.rdsadmin_util.unset_system_event` Oracle method: `ALTER SYSTEM SET EVENTS 'unset_event_clause'`

- Database tasks

Changing the global name of a database	Amazon RDS method: `rdsadmin.rdsadmin_util.rename_global_name` Oracle method: `alter database rename`
Creating and sizing tablespaces in RDS for Oracle	Amazon RDS method: `create tablespace` Oracle method: `alter database`
Setting the default tablespace in RDS for Oracle	Amazon RDS method: `rdsadmin.rdsadmin_util.alter_default_tablespace` Oracle method: `alter database default tablespace`
Setting the default temporary tablespace in RDS for Oracle	Amazon RDS method: `rdsadmin.rdsadmin_util.alter_default_temp_tablespace` Oracle method: `alter database default temporary tablespace`
Creating a temporary tablespace on the instance store	Amazon RDS method: `rdsadmin.rdsadmin_util.create_inst_store_tmp_tblspace` Oracle method: `create temporary tablespace`
Checkpointing a database	Amazon RDS method: `rdsadmin.rdsadmin_util.checkpoint` Oracle method: `alter system checkpoint`
Setting distributed recovery	Amazon RDS method: `rdsadmin.rdsadmin_util.enable_distr_recovery` Oracle method: `alter system enable distributed recovery`

Setting the database time zone	Amazon RDS method: `rdsadmin.rdsadmin_util.alter_db_time_zone` Oracle method: `alter database set time_zone`
Working with external tables in RDS for Oracle	—
Generating performance reports with Automatic Workload Repository (AWR)	Amazon RDS method: `rdsadmin.rdsadmin_diagnostic_util` procedures Oracle method: `dbms_workload_repository` package
Adjusting database links for use with DB instances in a VPC	—
Setting the default edition for a DB instance	Amazon RDS method: `rdsadmin.rdsadmin_util.alter_default_edition` Oracle method: `alter database default edition`
Enabling auditing for the SYS.AUD$ table	Amazon RDS method: `rdsadmin.rdsadmin_master_util.audit_all_sys_aud_table` Oracle method: `audit`
Disabling auditing for the SYS.AUD$ table	Amazon RDS method: `rdsadmin.rdsadmin_master_util.noaudit_all_sys_aud_table` Oracle method: `noaudit`
Cleaning up interrupted online index builds	Amazon RDS method: `rdsadmin.rdsadmin_dbms_repair.online_index_clean` Oracle method: `dbms_repair.online_index_clean`

Skipping corrupt blocks	Amazon RDS method: Several `rdsadmin.rdsadmin_dbms_repair` procedures Oracle method: `dbms_repair` package
Resizing tablespaces, data files, and tempfiles in RDS for Oracle	Amazon RDS method: `rdsadmin.rdsadmin_util.resize_temp_tablespace`, `rdsadmin.rdsadmin_util.resize_tempfile`, or `rdsadmin.rdsadmin_util.autoextend_tempfile` procedures `rdsadmin.rdsadmin_util.resize_datafile` or `rdsadmin.rdsadmin_util.autoextend_datafile` procedure Oracle method: —
Purging the recycle bin	Amazon RDS method: `EXEC rdsadmin.rdsadmin_util.purge_dba_recyclebin` Oracle method: `purge dba_recyclebin`
Setting the default displayed values for full redaction	Amazon RDS method: `EXEC rdsadmin.rdsadmin_util.dbms_redact_upd_full_rdct_val` Oracle method: `exec dbms_redact.UPDATE_FULL_REDACTION_VALUES`

- Log tasks

Setting force logging	Amazon RDS method: `rdsadmin.rdsadmin_util.force_logging` Oracle method: `alter database force logging`

Setting supplemental logging	Amazon RDS method: `rdsadmin.rdsadmin_util.alter_supplemental_logging` Oracle method: `alter database add supplemental log`
Switching online log files	Amazon RDS method: `rdsadmin.rdsadmin_util.switch_logfile` Oracle method: `alter system switch logfile`
Adding online redo logs	Amazon RDS method: `rdsadmin.rdsadmin_util.add_logfile`
Dropping online redo logs	Amazon RDS method: `rdsadmin.rdsadmin_util.drop_logfile`
Resizing online redo logs	—
Retaining archived redo logs	Amazon RDS method: `rdsadmin.rdsadmin_util.set_configuration`

Downloading archived redo logs from Amazon S3	Amazon RDS method: `rdsadmin.rdsadmin_archive_log_download.download_log_with_seqnum` Amazon RDS method: `rdsadmin.rdsadmin_archive_log_download.download_logs_in_seqnum_range`
Accessing online and archived redo logs	Amazon RDS method: `rdsadmin.rdsadmin_master_util.create_archivelog_dir` Amazon RDS method: `rdsadmin.rdsadmin_master_util.create_onlinelog_dir`

- RMAN tasks

Validating database files in RDS for Oracle	Amazon RDS method: `rdsadmin_rman_util. procedure` Oracle method: `RMAN VALIDATE`

Enabling and disabling block change tracking	Amazon RDS method: `rdsadmin_rman_util . procedure` Oracle method: `ALTER DATABASE`
Crosschecking archived redo logs	Amazon RDS method: `rdsadmin_rman_util .crosscheck_archivelog` Oracle method: `RMAN BACKUP`
Backing up archived redo log files	Amazon RDS method: `rdsadmin_rman_util . procedure` Oracle method: `RMAN BACKUP`
Performing a full database backup	Amazon RDS method: `rdsadmin_rman_util .backup_database_full` Oracle method: `RMAN BACKUP`
Performing an incremental database backup	Amazon RDS method: `rdsadmin_rman_util .backup_database_incremental` Oracle method: `RMAN BACKUP`

Backing up a tablespace	Amazon RDS method: `rdsadmin_rman_util.backup_database_tablespace` Oracle method: `RMAN BACKUP`

- Oracle Scheduler tasks

Modifying DBMS_SCHEDULER jobs	Amazon RDS method: `dbms_scheduler.set_attribute` Oracle method: `dbms_scheduler.set_attribute`
Modifying AutoTask maintenance windows	Amazon RDS method: `dbms_scheduler.set_attribute` Oracle method: `dbms_scheduler.set_attribute`
Setting the time zone for Oracle Scheduler jobs	Amazon RDS method: `dbms_scheduler.set_scheduler_attribute` Oracle method: `dbms_scheduler.set_scheduler_attribute`

Turning off Oracle Scheduler jobs owned by SYS	Amazon RDS method: `rdsadmin.rdsadmin_dbms_scheduler.disable` Oracle method: `dbms_scheduler.disable`
Turning on Oracle Scheduler jobs owned by SYS	Amazon RDS method: `rdsadmin.rdsadmin_dbms_scheduler.enable` Oracle method: `dbms_scheduler.enable`
Modifying the Oracle Scheduler repeat interval for jobs of CALENDAR type	Amazon RDS method: `rdsadmin.rdsadmin_dbms_scheduler.set_attribute` Oracle method: `dbms_scheduler.set_attribute`
Modifying the Oracle Scheduler repeat interval for jobs of NAMED type	Amazon RDS method: `rdsadmin.rdsadmin_dbms_scheduler.set_attribute` Oracle method: `dbms_scheduler.set_attribute`

Turning off autocommit for Oracle Scheduler job creation	Amazon RDS method: `rdsadmin.rdsadmin_dbms_scheduler.set_no_commit_flag` Oracle method: `dbms_isched.set_no_commit_flag`

- Diagnosing problems

Listing incidents	Amazon RDS method: `rdsadmin.rdsadmin_adrci_util.list_adrci_incidents` Oracle method: ADRCI command `show incident`
Listing problems	Amazon RDS method: `rdsadmin.rdsadmin_adrci_util.list_adrci_problem` Oracle method: ADRCI command `show problem`
Creating incident packages	Amazon RDS method: `rdsadmin.rdsadmin_adrci_util.create_adrci_package` Oracle method: ADRCI command `ips create package`

Showing trace files	Amazon RDS method: `rdsadmin.rdsadmin_adrci_util.show_adrci_tracefile` Oracle method: ADRCI command `show tracefile`

- Other tasks

Creating and dropping directories in the main data storage space	Amazon RDS method: `rdsadmin.rdsadmin_util.create_directory` Oracle method: `CREATE DIRECTORY` Amazon RDS method: `rdsadmin.rdsadmin_util.drop_directory` Oracle method: `DROP DIRECTORY`
Listing files in a DB instance directory	Amazon RDS method: `rdsadmin.rds_file_util.listdir` Oracle method: —

Reading files in a DB instance directory	Amazon RDS method: `rdsadmin.rds_file_util.read_text_file` Oracle method: —
Accessing Opatch files	Amazon RDS method: `rdsadmin.rds_file_util.read_text_file` or `rdsadmin.tracefile_listing` Oracle method: `opatch`
Setting parameters for advisor tasks	Amazon RDS method: `rdsadmin.rdsadmin_util.advisor_task_set_parameter` Oracle method: Various stored package procedures
Disabling AUTO_STATS_ADVISOR_TASK	Amazon RDS method: `rdsadmin.rdsadmin_util.advisor_task_drop` Oracle method: —
Re-enabling AUTO_STATS_ADVISOR_TASK	Amazon RDS method: `rdsadmin.rdsadmin_util.dbms_stats_init` Oracle method: —

You can also use Amazon RDS procedures for Amazon S3 integration with Oracle and for running OEM Management Agent database tasks. For more information, see Amazon S3 integration and Performing database tasks with the Management Agent.

Performing common system tasks for Oracle DB instances

Following, you can find how to perform certain common DBA tasks related to the system on your Amazon RDS DB instances running Oracle. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances, and restricts access to certain system procedures and tables that require advanced privileges.

Topics

- Disconnecting a session
- Terminating a session
- Canceling a SQL statement in a session
- Enabling and disabling restricted sessions
- Flushing the shared pool
- Granting SELECT or EXECUTE privileges to SYS objects
- Revoking SELECT or EXECUTE privileges on SYS objects
- Managing RDS_X$ views for Oracle DB instances
- Granting privileges to non-master users
- Creating custom functions to verify passwords
- Setting up a custom DNS server
- Setting and unsetting system diagnostic events

Disconnecting a session

To disconnect the current session by ending the dedicated server process, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.disconnect`. The `disconnect` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`sid`	number	—	Yes	The session identifier.

Parameter name	Data type	Default	Required	Description
serial	number	—	Yes	The serial number of the session.
method	varchar	'IMMEDIATE'	No	Valid values are 'IMMEDIATE' or 'POST_TRANSACTION'.

The following example disconnects a session.

```
begin
    rdsadmin.rdsadmin_util.disconnect(
        sid    => sid,
        serial => serial_number);
end;
/
```

To get the session identifier and the session serial number, query the V$SESSION view. The following example gets all sessions for the user AWSUSER.

```
SELECT SID, SERIAL#, STATUS FROM V$SESSION WHERE USERNAME = 'AWSUSER';
```

The database must be open to use this method. For more information about disconnecting a session, see ALTER SYSTEM in the Oracle documentation.

Terminating a session

To terminate a session, use the Amazon RDS procedure rdsadmin.rdsadmin_util.kill. The kill procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
sid	number	—	Yes	The session identifier.
serial	number	—	Yes	The serial number of the session.

Parameter name	Data type	Default	Required	Description
method	varchar	null	No	Valid values are 'IMMEDIATE' or 'PROCESS' . If you specify IMMEDIATE , it has the same effect as running the following statement: `ALTER SYSTEM KILL SESSION 'sid,serial#' IMMEDIATE` If you specify PROCESS, you terminate the processes associated with a session. Only specify PROCESS if terminating the session using IMMEDIATE was unsuccessful.

To get the session identifier and the session serial number, query the V$SESSION view. The following example gets all sessions for the user *AWSUSER*.

```
SELECT SID, SERIAL#, STATUS FROM V$SESSION WHERE USERNAME = 'AWSUSER';
```

The following example terminates a session.

```
BEGIN
    rdsadmin.rdsadmin_util.kill(
        sid    => sid,
        serial => serial_number,
        method => 'IMMEDIATE');
END;
/
```

The following example terminates the processes associated with a session.

```
BEGIN
    rdsadmin.rdsadmin_util.kill(
        sid    => sid,
        serial => serial_number,
        method => 'PROCESS');
END;
/
```

Canceling a SQL statement in a session

To cancel a SQL statement in a session, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.cancel`.

> **Note**
>
> This procedure is supported for Oracle Database 19c (19.0.0) and all higher major and minor versions of RDS for Oracle.

The `cancel` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`sid`	number	—	Yes	The session identifier.
`serial`	number	—	Yes	The serial number of the session.
`sql_id`	varchar2	null	No	The SQL identifier of the SQL statement.

The following example cancels a SQL statement in a session.

```
begin
    rdsadmin.rdsadmin_util.cancel(
        sid    => sid,
```

```
        serial => serial_number,
        sql_id => sql_id);
end;
/
```

To get the session identifier, the session serial number, and the SQL identifier of a SQL statement, query the V$SESSION view. The following example gets all sessions and SQL identifiers for the user AWSUSER.

```
select SID, SERIAL#, SQL_ID, STATUS from V$SESSION where USERNAME = 'AWSUSER';
```

Enabling and disabling restricted sessions

To enable and disable restricted sessions, use the Amazon RDS procedure rdsadmin.rdsadmin_util.restricted_session. The restricted_session procedure has the following parameters.

Parameter name	Data type	Default	Yes	Description
p_enable	boolean	true	No	Set to true to enable restricted sessions, false to disable restricted sessions.

The following example shows how to enable and disable restricted sessions.

```
/* Verify that the database is currently unrestricted. */

SELECT LOGINS FROM V$INSTANCE;

LOGINS
-------
ALLOWED

/* Enable restricted sessions */

EXEC rdsadmin.rdsadmin_util.restricted_session(p_enable => true);
```

```
/* Verify that the database is now restricted. */

SELECT LOGINS FROM V$INSTANCE;

LOGINS
----------
RESTRICTED


/* Disable restricted sessions */

EXEC rdsadmin.rdsadmin_util.restricted_session(p_enable => false);


/* Verify that the database is now unrestricted again. */

SELECT LOGINS FROM V$INSTANCE;

LOGINS
-------
ALLOWED
```

Flushing the shared pool

To flush the shared pool, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.flush_shared_pool`. The `flush_shared_pool` procedure has no parameters.

The following example flushes the shared pool.

```
EXEC rdsadmin.rdsadmin_util.flush_shared_pool;
```

Flushing the buffer cache

To flush the buffer cache, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.flush_buffer_cache`. The `flush_buffer_cache` procedure has no parameters.

The following example flushes the buffer cache.

```
EXEC rdsadmin.rdsadmin_util.flush_buffer_cache;
```

Flushing the database smart flash cache

To flush the database smart flash cache, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.flush_flash_cache`. The `flush_flash_cache` procedure has no parameters. The following example flushes the database smart flash cache.

```
EXEC rdsadmin.rdsadmin_util.flush_flash_cache;
```

For more information about using the database smart flash cache with RDS for Oracle, see Storing temporary data in an RDS for Oracle instance store.

Granting SELECT or EXECUTE privileges to SYS objects

Usually you transfer privileges by using roles, which can contain many objects. To grant privileges to a single object, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.grant_sys_object`. The procedure grants only privileges that the master user has already been granted through a role or direct grant.

The `grant_sys_object` procedure has the following parameters.

Important

For all parameter values, use uppercase unless you created the user with a case-sensitive identifier. For example, if you run `CREATE USER myuser` or `CREATE USER MYUSER`, the data dictionary stores `MYUSER`. However, if you use double quotes in `CREATE USER "MyUser"`, the data dictionary stores `MyUser`.

Parameter name	Data type	Default	Required	Description
p_obj_name	varchar2	—	Yes	The name of the object to grant privileges for. The object can be a directory, function, package, procedure, sequence, table, or view. Object names must be spelled exactly as they appear in DBA_OBJEC

Parameter name	Data type	Default	Required	Description
				TS . Most system objects are defined in uppercase , so we recommend that you try that first.
p_grantee	varchar2	—	Yes	The name of the object to grant privileges to. The object can be a schema or a role.
p_privilege	varchar2	null	Yes	—
p_grant_option	boolean	false	No	Set to true to use the with grant option.

The following example grants select privileges on an object named V_$SESSION to a user named USER1.

```
begin
    rdsadmin.rdsadmin_util.grant_sys_object(
        p_obj_name  => 'V_$SESSION',
        p_grantee   => 'USER1',
        p_privilege => 'SELECT');
end;
/
```

The following example grants select privileges on an object named V_$SESSION to a user named USER1 with the grant option.

```
begin
    rdsadmin.rdsadmin_util.grant_sys_object(
        p_obj_name     => 'V_$SESSION',
        p_grantee      => 'USER1',
        p_privilege    => 'SELECT',
        p_grant_option => true);
end;
/
```

To be able to grant privileges on an object, your account must have those privileges granted to it directly with the grant option, or via a role granted using `with admin option`. In the most common case, you may want to grant SELECT on a DBA view that has been granted to the SELECT_CATALOG_ROLE role. If that role isn't already directly granted to your user using `with admin option`, then you can't transfer the privilege. If you have the DBA privilege, then you can grant the role directly to another user.

The following example grants the SELECT_CATALOG_ROLE and EXECUTE_CATALOG_ROLE to USER1. Since the `with admin option` is used, USER1 can now grant access to SYS objects that have been granted to SELECT_CATALOG_ROLE.

```
GRANT SELECT_CATALOG_ROLE TO USER1 WITH ADMIN OPTION;
GRANT EXECUTE_CATALOG_ROLE to USER1 WITH ADMIN OPTION;
```

Objects already granted to PUBLIC do not need to be re-granted. If you use the `grant_sys_object` procedure to re-grant access, the procedure call succeeds.

Revoking SELECT or EXECUTE privileges on SYS objects

To revoke privileges on a single object, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.revoke_sys_object`. The procedure only revokes privileges that the master account has already been granted through a role or direct grant.

The `revoke_sys_object` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_obj_name	varchar2	—	Yes	The name of the object to revoke privileges for. The object can be a directory, function, package, procedure, sequence, table, or view. Object names must be spelled exactly as they appear in DBA_OBJECTS . Most system objects are defined in upper

Parameter name	Data type	Default	Required	Description
				case, so we recommend you try that first.
p_revokee	varchar2	—	Yes	The name of the object to revoke privileges for. The object can be a schema or a role.
p_privilege	varchar2	null	Yes	—

The following example revokes select privileges on an object named V_$SESSION from a user named USER1.

```
begin
    rdsadmin.rdsadmin_util.revoke_sys_object(
        p_obj_name  => 'V_$SESSION',
        p_revokee   => 'USER1',
        p_privilege => 'SELECT');
end;
/
```

Managing RDS_X$ views for Oracle DB instances

You might need to access SYS.X$ fixed tables, which are only accessible by SYS. To create SYS.RDS_X$ views on eligible X$ tables, use the procedures in the rdsadmin.rdsadmin_util package. Your master user is automatically granted the privilege SELECT ... WITH GRANT OPTION on the RDS_X$ views.

The rdsadmin.rdsadmin_util procedures are available in the following cases:

- Existing DB instances that have never been upgraded and use the following releases:
  - 21.0.0.0.ru-2023-10.rur-2023-10.r1 and higher 21c releases
  - 19.0.0.0.ru-2023-10.rur-2023-10.r1 and higher 19c releases
- Any new DB instance that you create
- Any existing DB instance that you have upgraded

> **Important**
>
> Internally, the `rdsadmin.rdsadmin_util` package creates views on X$ tables. The X$ tables are internal system objects that aren't described in the Oracle Database documentation. We recommend that you test specific views in your non-production database and only create views in your production database under the guidance of Oracle Support.

List X$ fixed tables eligible for use in RDS_X$ views

To list X$ tables that are eligible for use in RDS_X$ views, use the RDS procedure `rdsadmin.rdsadmin_util.list_allowed_sys_x$_views`. This procedure accepts no parameters. The following statements lists all eligible X$ tables (sample output included).

```
SQL> SET SERVEROUTPUT ON
SQL> SELECT * FROM TABLE(rdsadmin.rdsadmin_util.list_allowed_sys_x$_views);

'X$BH'
'X$K2GTE'
'X$KCBWBPD'
'X$KCBWDS'
'X$KGLLK'
'X$KGLOB'
'X$KGLPN'
'X$KSLHOT'
'X$KSMSP'
'X$KSPPCV'
'X$KSPPI'
'X$KSPPSV'
'X$KSQEQ'
'X$KSQRS'
'X$KTUXE'
'X$KQRFP'
```

The list of eligible X$ tables can change over time. To make sure that your list of eligible X$ fixed tables is current, rerun `list_allowed_sys_x$_views` periodically.

Creating SYS.RDS_X$ views

To create an RDS_X$ view on an eligible X$ table, use the RDS procedure `rdsadmin.rdsadmin_util.create_sys_x$_view`. You can only create views for the tables listed in the output of `rdsadmin.rdsadmin_util.list_allowed_sys_x$_views`. The `create_sys_x$_view` procedure accepts the following parameters.

Parameter name	Data type	Default	Required	Description
`p_x$_tbl`	varchar2	Null	Yes	A valid X$ table name. The value must be one of the X$ tables reported by `list_allowed_sys_x$_views` .
`p_force_creation`	Boolean	FALSE	No	A value indicating whether to force creation of an RDS_X$ view that already exists for an X$ table. By default, RDS won't create a view if it already exists. To force creation, set this parameter to TRUE.

The following example creates the SYS.RDS_X$KGLOB view on the table X$KGLOB. The format for the view name is RDS_*X$tablename*.

```
SQL> SET SERVEROUTPUT ON
SQL> EXEC rdsadmin.rdsadmin_util.create_sys_x$_view('X$KGLOB');

PL/SQL procedure successfully completed.
```

The following data dictionary query lists the view SYS.RDS_X$KGLOB and shows its status. Your master user is automatically granted the privilege `SELECT ... WITH GRANT OPTION` on this view.

```
SQL> SET SERVEROUTPUT ON
```

```
SQL> COL OWNER FORMAT A30
SQL> COL OBJECT_NAME FORMAT A30
SQL> COL STATUS FORMAT A30
SQL> SET LINESIZE 200
SQL> SELECT OWNER, OBJECT_NAME, STATUS
FROM DBA_OBJECTS
WHERE OWNER = 'SYS' AND OBJECT_NAME = 'RDS_X$KGLOB';

OWNER                          OBJECT_NAME                    STATUS
------------------------------ ------------------------------
 ------------------------------
SYS                            RDS_X$KGLOB                    VALID
```

Important

X$ tables aren't guaranteed to stay the same before and after an upgrade. RDS for Oracle drops and recreates the RDS_X$ views on X$ tables during an engine upgrade. Then it grants the `SELECT ... WITH GRANT OPTION` privilege to the master user. After an upgrade, grant privileges to database users as needed on the corresponding RDS_X$ views.

Listing SYS.RDS_X$ views

To list existing RDS_X$ views, use the RDS procedure `rdsadmin.rdsadmin_util.list_created_sys_x$_views`. The procedure lists only views that were created by the procedure `create_sys_x$_view`. The following example lists X$ tables that have corresponding RDS_X$ views (sample output included).

```
SQL> SET SERVEROUTPUT ON
SQL> COL XD_TBL_NAME FORMAT A30
SQL> COL STATUS FORMAT A30
SQL> SET LINESIZE 200
SQL> SELECT * FROM TABLE(rdsadmin.rdsadmin_util.list_created_sys_x$_views);

XD_TBL_NAME                    STATUS
------------------------------ ------------------------------
X$BH                           VALID
X$K2GTE                        VALID
X$KCBWBPD                      VALID

3 rows selected.
```

Dropping RDS_X$ views

To drop a SYS.RDS_X$ view, use the RDS procedure `rdsadmin.rdsadmin_util.drop_sys_x$_view`. You can only drop views listed in the output of `rdsadmin.rdsadmin_util.list_allowed_sys_x$_views`. The `drop_sys_x$_view` procedure accepts the following parameter.

Parameter name	Data type	Default	Required	Description
p_x$_tbl	varchar2	Null	Yes	A valid X$ fixed table name. The value must be one of the X$ fixed tables reported by `list_created_sys_x$_views .`

The following example drops the RDS_X$KGLOB view, which was created on the table X$KGLOB.

```
SQL> SET SERVEROUTPUT ON
SQL> EXEC rdsadmin.rdsadmin_util.drop_sys_x$_view('X$KGLOB');

PL/SQL procedure successfully completed.
```

The following example shows that the view SYS.RDS_X$KGLOB has been dropped (sample output included).

```
SQL> SET SERVEROUTPUT ON
SQL> COL OWNER FORMAT A30
SQL> COL OBJECT_NAME FORMAT A30
SQL> COL STATUS FORMAT A30
SQL> SET LINESIZE 200
SQL> SELECT OWNER, OBJECT_NAME, STATUS
FROM DBA_OBJECTS
WHERE OWNER = 'SYS' AND OBJECT_NAME = 'RDS_X$KGLOB';

no rows selected
```

Granting privileges to non-master users

You can grant select privileges for many objects in the SYS schema by using the SELECT_CATALOG_ROLE role. The SELECT_CATALOG_ROLE role gives users SELECT privileges on data dictionary views. The following example grants the role SELECT_CATALOG_ROLE to a user named user1.

```
GRANT SELECT_CATALOG_ROLE TO user1;
```

You can grant EXECUTE privileges for many objects in the SYS schema by using the EXECUTE_CATALOG_ROLE role. The EXECUTE_CATALOG_ROLE role gives users EXECUTE privileges for packages and procedures in the data dictionary. The following example grants the role EXECUTE_CATALOG_ROLE to a user named *user1*.

```
GRANT EXECUTE_CATALOG_ROLE TO user1;
```

The following example gets the permissions that the roles SELECT_CATALOG_ROLE and EXECUTE_CATALOG_ROLE allow.

```
  SELECT *
    FROM ROLE_TAB_PRIVS
   WHERE ROLE IN ('SELECT_CATALOG_ROLE','EXECUTE_CATALOG_ROLE')
ORDER BY ROLE, TABLE_NAME ASC;
```

The following example creates a non-master user named user1, grants the CREATE SESSION privilege, and grants the SELECT privilege on a database named *sh.sales*.

```
CREATE USER user1 IDENTIFIED BY PASSWORD;
GRANT CREATE SESSION TO user1;
GRANT SELECT ON sh.sales TO user1;
```

Creating custom functions to verify passwords

You can create a custom password verification function in the following ways:

- To use standard verification logic, and to store your function in the SYS schema, use the create_verify_function procedure.
- To use custom verification logic, or to avoid storing your function in the SYS schema, use the create_passthrough_verify_fcn procedure.

The create_verify_function procedure

You can create a custom function to verify passwords by using the Amazon RDS procedure `rdsadmin.rdsadmin_password_verify.create_verify_function`. The `create_verify_function` procedure is supported for all versions of RDS for Oracle.

The `create_verify_function` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_verify_function_name`	varchar2	—	Yes	The name for your custom function. This function is created for you in the SYS schema. You assign this function to user profiles.
`p_min_length`	number	8	No	The minimum number of characters required.
`p_max_length`	number	256	No	The maximum number of characters allowed.
`p_min_letters`	number	1	No	The minimum number of letters required.
`p_min_uppercase`	number	0	No	The minimum number of uppercase letters required.
`p_min_lowercase`	number	0	No	The minimum number of lowercase letters required.
`p_min_digits`	number	1	No	The minimum number of digits required.

Parameter name	Data type	Default	Required	Description
p_min_special	number	0	No	The minimum number of special characters required.
p_min_different_chars	number	3	No	The minimum number of different characters required between the old and new password.
p_disallow_username	boolean	true	No	Set to true to disallow the user name in the password.
p_disallow_reverse	boolean	true	No	Set to true to disallow the reverse of the user name in the password.
p_disallow_db_name	boolean	true	No	Set to true to disallow the database or server name in the password.
p_disallow_simple_strings	boolean	true	No	Set to true to disallow simple strings as the password.
p_disallow_whitespace	boolean	false	No	Set to true to disallow white space characters in the password.
p_disallow_at_sign	boolean	false	No	Set to true to disallow the @ character in the password.

You can create multiple password verification functions.

There are restrictions on the name of your custom function. Your custom function can't have the same name as an existing system object. The name can be no more than 30 characters long. Also, the name must include one of the following strings: PASSWORD, VERIFY, COMPLEXITY, ENFORCE, or STRENGTH.

The following example creates a function named CUSTOM_PASSWORD_FUNCTION. The function requires that a password has at least 12 characters, 2 uppercase characters, 1 digit, and 1 special character, and that the password disallows the @ character.

```
begin
    rdsadmin.rdsadmin_password_verify.create_verify_function(
        p_verify_function_name => 'CUSTOM_PASSWORD_FUNCTION',
        p_min_length           => 12,
        p_min_uppercase        => 2,
        p_min_digits           => 1,
        p_min_special          => 1,
        p_disallow_at_sign     => true);
end;
/
```

To see the text of your verification function, query DBA_SOURCE. The following example gets the text of a custom password function named CUSTOM_PASSWORD_FUNCTION.

```
COL TEXT FORMAT a150

  SELECT TEXT
    FROM DBA_SOURCE
   WHERE OWNER = 'SYS'
     AND NAME = 'CUSTOM_PASSWORD_FUNCTION'
ORDER BY LINE;
```

To associate your verification function with a user profile, use ALTER PROFILE. The following example associates a verification PL/SQL function named CUSTOM_PASSWORD_FUNCTION with the DEFAULT user profile. PASSWORD_VERIFY_FUNCTION is the Oracle profile resource name.

```
ALTER PROFILE DEFAULT LIMIT PASSWORD_VERIFY_FUNCTION CUSTOM_PASSWORD_FUNCTION;
```

To see which user profiles are associated with which verification functions, query DBA_PROFILES. The following example gets the profiles that are associated with the custom verification function named CUSTOM_PASSWORD_FUNCTION.

```
SELECT * FROM DBA_PROFILES WHERE RESOURCE_NAME = 'PASSWORD_VERIFY_FUNCTION' AND LIMIT =
 'CUSTOM_PASSWORD_FUNCTION';


PROFILE                    RESOURCE_NAME                    RESOURCE  LIMIT
-------------------------  -------------------------------  --------
 ------------------------
DEFAULT                    PASSWORD_VERIFY_FUNCTION         PASSWORD
 CUSTOM_PASSWORD_FUNCTION
```

The following example gets all profiles and the password verification functions that they are associated with.

```
SELECT * FROM DBA_PROFILES WHERE RESOURCE_NAME = 'PASSWORD_VERIFY_FUNCTION';

PROFILE                    RESOURCE_NAME                    RESOURCE  LIMIT
-------------------------  -------------------------------  --------
 ------------------------
DEFAULT                    PASSWORD_VERIFY_FUNCTION         PASSWORD
 CUSTOM_PASSWORD_FUNCTION
RDSADMIN                   PASSWORD_VERIFY_FUNCTION         PASSWORD  NULL
```

The create_passthrough_verify_fcn procedure

The `create_passthrough_verify_fcn` procedure is supported for all versions of RDS for Oracle.

You can create a custom function to verify passwords by using the Amazon RDS procedure `rdsadmin.rdsadmin_password_verify.create_passthrough_verify_fcn`. The `create_passthrough_verify_fcn` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_verify_function_name`	varchar2	—	Yes	The name for your custom verification function. This is a wrapper function that is created for you in the SYS schema, and

Parameter name	Data type	Default	Required	Description
				it doesn't contain any verification logic. You assign this function to user profiles.
p_target_owner	varchar2	—	Yes	The schema owner for your custom verification function.
p_target_function_name	varchar2	—	Yes	The name of your existing custom function that contains the verification logic. Your custom function must return a boolean. Your function should return `true` if the password is valid and `false` if the password is invalid.

The following example creates a password verification function that uses the logic from the function named PASSWORD_LOGIC_EXTRA_STRONG.

```
begin
    rdsadmin.rdsadmin_password_verify.create_passthrough_verify_fcn(
        p_verify_function_name => 'CUSTOM_PASSWORD_FUNCTION',
        p_target_owner         => 'TEST_USER',
        p_target_function_name => 'PASSWORD_LOGIC_EXTRA_STRONG');
end;
/
```

To associate the verification function with a user profile, use `alter profile`. The following example associates the verification function with the DEFAULT user profile.

```
ALTER PROFILE DEFAULT LIMIT PASSWORD_VERIFY_FUNCTION CUSTOM_PASSWORD_FUNCTION;
```

Setting up a custom DNS server

Amazon RDS supports outbound network access on your DB instances running Oracle. For more information about outbound network access, including prerequisites, see Configuring UTL_HTTP access using certificates and an Oracle wallet.

Amazon RDS Oracle allows Domain Name Service (DNS) resolution from a custom DNS server owned by the customer. You can resolve only fully qualified domain names from your Amazon RDS DB instance through your custom DNS server.

After you set up your custom DNS name server, it takes up to 30 minutes to propagate the changes to your DB instance. After the changes are propagated to your DB instance, all outbound network traffic requiring a DNS lookup queries your DNS server over port 53.

To set up a custom DNS server for your Amazon RDS for Oracle DB instance, do the following:

- From the DHCP options set attached to your virtual private cloud (VPC), set the `domain-name-servers` option to the IP address of your DNS name server. For more information, see DHCP options sets.

  > **Note**
  >
  > The `domain-name-servers` option accepts up to four values, but your Amazon RDS DB instance uses only the first value.

- Ensure that your DNS server can resolve all lookup queries, including public DNS names, Amazon EC2 private DNS names, and customer-specific DNS names. If the outbound network traffic contains any DNS lookups that your DNS server can't handle, your DNS server must have appropriate upstream DNS providers configured.
- Configure your DNS server to produce User Datagram Protocol (UDP) responses of 512 bytes or less.
- Configure your DNS server to produce Transmission Control Protocol (TCP) responses of 1024 bytes or less.
- Configure your DNS server to allow inbound traffic from your Amazon RDS DB instances over port 53. If your DNS server is in an Amazon VPC, the VPC must have a security group that contains inbound rules that permit UDP and TCP traffic on port 53. If your DNS server is not in an Amazon VPC, it must have appropriate firewall allow-listing to permit UDP and TCP inbound traffic on port 53.

For more information, see Security groups for your VPC and Adding and removing rules.

- Configure the VPC of your Amazon RDS DB instance to allow outbound traffic over port 53. Your VPC must have a security group that contains outbound rules that allow UDP and TCP traffic on port 53.

  For more information, see Security groups for your VPC and Adding and removing rules.

- The routing path between the Amazon RDS DB instance and the DNS server has to be configured correctly to allow DNS traffic.
  - If the Amazon RDS DB instance and the DNS server are not in the same VPC, a peering connection has to be set up between them. For more information, see What is VPC peering?

Setting and unsetting system diagnostic events

To set and unset diagnostic events at the session level, you can use the Oracle SQL statement `ALTER SESSION SET EVENTS`. However, to set events at the system level you can't use Oracle SQL. Instead, use the system event procedures in the `rdsadmin.rdsadmin_util` package. The system event procedures are available in the following engine versions:

- All Oracle Database 21c versions
- 19.0.0.0.ru-2020-10.rur-2020-10.r1 and higher Oracle Database 19c versions

  For more information, see Version 19.0.0.0.ru-2020-10.rur-2020-10.r1 in the *Amazon RDS for Oracle Release Notes*

> **Important**
>
> Internally, the `rdsadmin.rdsadmin_util` package sets events by using the `ALTER SYSTEM SET EVENTS` statement. This `ALTER SYSTEM` statement isn't documented in the Oracle Database documentation. Some system diagnostic events can generate large amounts of tracing information, cause contention, or affect database availability. We recommend that you test specific diagnostic events in your nonproduction database, and only set events in your production database under guidance of Oracle Support.

Listing allowed system diagnostic events

To list the system events that you can set, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.list_allowed_system_events`. This procedure accepts no parameters.

The following example lists all system events that you can set.

```
SET SERVEROUTPUT ON
EXEC rdsadmin.rdsadmin_util.list_allowed_system_events;
```

The following sample output lists event numbers and their descriptions. Use the Amazon RDS procedures `set_system_event` to set these events and `unset_system_event` to unset them.

```
604   - error occurred at recursive SQL level
942   - table or view does not exist
1401  - inserted value too large for column
1403  - no data found
1410  - invalid ROWID
1422  - exact fetch returns more than requested number of rows
1426  - numeric overflow
1427  - single-row subquery returns more than one row
1476  - divisor is equal to zero
1483  - invalid length for DATE or NUMBER bind variable
1489  - result of string concatenation is too long
1652  - unable to extend temp segment by  in tablespace
1858  - a non-numeric character was found where a numeric was expected
4031  - unable to allocate  bytes of shared memory ("","","","")
6502  - PL/SQL: numeric or value error
10027 - Specify Deadlock Trace Information to be Dumped
10046 - enable SQL statement timing
10053 - CBO Enable optimizer trace
10173 - Dynamic Sampling time-out error
10442 - enable trace of kst for ORA-01555 diagnostics
12008 - error in materialized view refresh path
12012 - error on auto execute of job
12504 - TNS:listener was not given the SERVICE_NAME in CONNECT_DATA
14400 - inserted partition key does not map to any partition
31693 - Table data object  failed to load/unload and is being skipped due to error:
```

> **Note**
>
> The list of the allowed system events can change over time. To make sure that you have the most recent list of eligible events, use `rdsadmin.rdsadmin_util.list_allowed_system_events`.

Setting system diagnostic events

To set a system event, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.set_system_event`. You can only set events listed in the output of `rdsadmin.rdsadmin_util.list_allowed_system_events`. The `set_system_event` procedure accepts the following parameters.

Parameter name	Data type	Default	Required	Description
`p_event`	number	—	Yes	The system event number. The value must be one of the event numbers reported by `list_allowed_system_events` .
`p_level`	number	—	Yes	The event level. See the Oracle Database documentation or Oracle Support for descriptions of different level values.

The procedure `set_system_event` constructs and runs the required `ALTER SYSTEM SET EVENTS` statements according to the following principles:

- The event type (`context` or `errorstack`) is determined automatically.
- A statement in the form `ALTER SYSTEM SET EVENTS 'event LEVEL event_level'` sets the context events. This notation is equivalent to `ALTER SYSTEM SET EVENTS 'event TRACE NAME CONTEXT FOREVER, LEVEL event_level'`.

- A statement in the form `ALTER SYSTEM SET EVENTS 'event ERRORSTACK (event_level)'` sets the error stack events. This notation is equivalent to `ALTER SYSTEM SET EVENTS 'event TRACE NAME ERRORSTACK LEVEL event_level'`.

The following example sets event 942 at level 3, and event 10442 at level 10. Sample output is included.

```
SQL> SET SERVEROUTPUT ON
SQL> EXEC rdsadmin.rdsadmin_util.set_system_event(942,3);
Setting system event 942 with: alter system set events '942 errorstack (3)'

PL/SQL procedure successfully completed.

SQL> EXEC rdsadmin.rdsadmin_util.set_system_event(10442,10);
Setting system event 10442 with: alter system set events '10442 level 10'

PL/SQL procedure successfully completed.
```

Listing system diagnostic events that are set

To list the system events that are currently set, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.list_set_system_events`. This procedure reports only events set at system level by `set_system_event`.

The following example lists the active system events.

```
SET SERVEROUTPUT ON
EXEC rdsadmin.rdsadmin_util.list_set_system_events;
```

The following sample output shows the list of events, the event type, the level at which the events are currently set, and the time when the event was set.

```
942 errorstack (3) - set at 2020-11-03 11:42:27
10442 level 10 - set at 2020-11-03 11:42:41

PL/SQL procedure successfully completed.
```

Unsetting system diagnostic events

To unset a system event, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.unset_system_event`. You can only unset events listed in the output of `rdsadmin.rdsadmin_util.list_allowed_system_events`. The `unset_system_event` procedure accepts the following parameter.

Parameter name	Data type	Default	Required	Description
`p_event`	number	—	Yes	The system event number. The value must be one of the event numbers reported by `list_allowed_system_events` .

The following example unsets events 942 and 10442. Sample output is included.

```
SQL> SET SERVEROUTPUT ON
SQL> EXEC rdsadmin.rdsadmin_util.unset_system_event(942);
Unsetting system event 942 with: alter system set events '942 off'

PL/SQL procedure successfully completed.

SQL> EXEC rdsadmin.rdsadmin_util.unset_system_event(10442);
Unsetting system event 10442 with: alter system set events '10442 off'

PL/SQL procedure successfully completed.
```

Performing common database tasks for Oracle DB instances

Following, you can find how to perform certain common DBA tasks related to databases on your Amazon RDS DB instances running Oracle. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. Amazon RDS also restricts access to some system procedures and tables that require advanced privileges.

Topics

- Changing the global name of a database

- Working with tablespaces in RDS for Oracle
- Working with tempfiles in RDS for Oracle
- Resizing tablespaces, data files, and tempfiles in RDS for Oracle
- Moving data between storage volumes in RDS for Oracle
- Working with external tables in RDS for Oracle

Changing the global name of a database

To change the global name of a database, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.rename_global_name`. The `rename_global_name` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_new_global_name	varchar2	—	Yes	The new global name for the database.

The database must be open for the name change to occur. For more information about changing the global name of a database, see ALTER DATABASE in the Oracle documentation.

The following example changes the global name of a database to `new_global_name`.

```
EXEC rdsadmin.rdsadmin_util.rename_global_name(p_new_global_name => 'new_global_name');
```

Working with tablespaces in RDS for Oracle

You can use tablespaces with RDS for Oracle, which is a logical storage unit that stores the database's data.

> **Important**
>
> If your DB instance has replicas, we recommend using parameter group settings instead of session-level changes to manage default file locations. Session-level changes to default file locations in the primary instance are not automatically reflected in the replicas. Using parameter group settings ensures consistent file locations across your primary and replica instances.

Topics

- Specifying database file locations in RDS for Oracle
- Creating and sizing tablespaces in RDS for Oracle
- Creating tablespaces on additional storage volumes in RDS for Oracle
- Setting the default tablespace in RDS for Oracle
- Setting the default temporary tablespace in RDS for Oracle
- Creating a temporary tablespace on the instance store

Specifying database file locations in RDS for Oracle

RDS for Oracle uses Oracle Managed Files (OMF) to name database files. When you create database files the database derives the setting based on the current setting of the `DB_CREATE_FILE_DEST` initialization parameter.

The default value of the `DB_CREATE_FILE_DEST` initialization parameter is `/rdsdbdata/db` for standalone databases and `/rdsdbdata/db/pdb` for containerized (CDB/MT) architecture. If your DB instance has additional storage volumes, then you can set `DB_CREATE_FILE_DEST` to your volume locations. For example, if your instance has a volume mounted on `/rdsdbdata/db`, you can set `DB_CREATE_FILE_DEST` to this value.

You can modify the `DB_CREATE_FILE_DEST` parameter at either the session level or Oracle database instance level.

Modifying DB_CREATE_FILE_SET at the instance level

To modify the parameter at the instance level, update the parameter in the parameter group assigned to your DB instance and apply it. For more information, see RDS for Oracle initialization parameters and Modifying parameters in a DB parameter group in Amazon RDS.

Modifying DB_CREATE_FILE_DEST at the session level

You can modify the parameter at the session level by executing an `ALTER SESSION` statement. This approach is useful when you want to create database files in a specific location for a particular session without affecting the entire instance.

The following example shows how to check the current parameter value and modify it for the session:

```
SHOW PARAMETER db_create_file_dest

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_create_file_dest                  string      /rdsdbdata/db

ALTER SESSION SET db_create_file_dest = '/rdsdbdata2/db';

Session altered.

SHOW PARAMETER db_create_file_dest

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_create_file_dest                  string      /rdsdbdata2/db
```

Creating and sizing tablespaces in RDS for Oracle

When you create tablespaces, the database creates the data files in the storage volume specified by the DB_CREATE_FILE_DEST initialization parameter at the time of creation. By default, if you don't specify a data file size, tablespaces are created with the default of AUTOEXTEND ON, and no maximum size. In the following example, the tablespace *users1* is autoextensible.

```
CREATE TABLESPACE users1;
```

Because of these default settings, tablespaces can grow to consume all allocated storage. We recommend that you specify an appropriate maximum size on permanent and temporary tablespaces, and that you carefully monitor space usage.

The following example creates a tablespace named *users2* with a starting size of 1 gigabyte. Because a data file size is specified, but AUTOEXTEND ON isn't specified, the tablespace isn't autoextensible.

```
CREATE TABLESPACE users2 DATAFILE SIZE 1G;
```

The following example creates a tablespace named *users3* with a starting size of 1 gigabyte, autoextend turned on, and a maximum size of 10 gigabytes.

```
CREATE TABLESPACE users3 DATAFILE SIZE 1G AUTOEXTEND ON MAXSIZE 10G;
```

The following example creates a temporary tablespace named *temp01*.

```
CREATE TEMPORARY TABLESPACE temp01;
```

You can resize a bigfile tablespace by using ALTER TABLESPACE. You can specify the size in kilobytes (K), megabytes (M), gigabytes (G), or terabytes (T). The following example resizes a bigfile tablespace named *users_bf* to 200 MB.

```
ALTER TABLESPACE users_bf RESIZE 200M;
```

The following example adds an additional data file to a smallfile tablespace named *users_sf*.

```
ALTER TABLESPACE users_sf ADD DATAFILE SIZE 100000M AUTOEXTEND ON NEXT 250m
 MAXSIZE UNLIMITED;
```

Creating tablespaces on additional storage volumes in RDS for Oracle

To create a tablespace on an additional storage volume, modify the DB_CREATE_FILE_DEST parameter to the volume location. The following example sets the file location to /rdsdbdata2/db.

```
ALTER SESSION SET db_create_file_dest = '/rdsdbdata2/db';

Session altered.
```

In the following example, you create a tablespace on the additional volume /rdsdbdata2/db.

```
CREATE TABLESPACE new_tablespace DATAFILE SIZE 10G;

Tablespace created.

SELECT tablespace_name,file_id,file_name FROM dba_data_files
WHERE tablespace_name = 'NEW_TABLESPACE';

TABLESPACE_NAME              FILE_ID FILE_NAME
-------------------------- ----------
 --------------------------------------------------------------------------------
NEW_TABLESPACE                     7 /rdsdbdata2/db/ORCL_A/datafile/
o1_mf_newtable_a123b4c5_.dbf
```

To create a smallfile tablespace and spread its data files across different storage volumes, add data files to the tablespace after you create it. In the following example, you create a tablespace with the data files in the default location of `/rdsdbdata/db`. Then you set the default destination to `/rdsdbdata/db2`. When you add a data file to your newly created tablespace, the database stores the file in `/rdsdbdata/db2`.

```
ALTER SESSION SET db_create_file_dest = '/rdsdbdata/db';

Session altered.

CREATE SMALLFILE TABLESPACE smalltbs DATAFILE SIZE 10G;

Tablespace created.

SELECT tablespace_name,file_id,file_name FROM dba_data_files
WHERE tablespace_name = 'SMALLTBS';

TABLESPACE_NAME              FILE_ID FILE_NAME
------------------------- ----------
 --------------------------------------------------------------------------------
SMALLTBS                           8 /rdsdbdata/db/ORCL_A/datafile/
o1_mf_smalltbs_n563yryk_.dbf

ALTER SESSION SET db_create_file_dest = '/rdsdbdata2/db';

Session altered.

ALTER TABLESPACE smalltbs ADD DATAFILE SIZE 10G;

Tablespace altered.

SELECT tablespace_name,file_id,file_name FROM dba_data_files
WHERE tablespace_name = 'SMALLTBS';

TABLESPACE_NAME              FILE_ID FILE_NAME
------------------------- ----------
 --------------------------------------------------------------------------------
SMALLTBS                           8 /rdsdbdata/db/ORCL_A/datafile/
o1_mf_smalltbs_n563yryk_.dbf
SMALLTBS                           9 /rdsdbdata2/db/ORCL_A/datafile/
o1_mf_smalltbs_n564004g_.dbf
```

Setting the default tablespace in RDS for Oracle

To set the default tablespace, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.alter_default_tablespace`. The `alter_default_tablespace` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`tablespace_name`	varchar	—	Yes	The name of the default tablespace.

The following example sets the default tablespace to *`users2`*:

```
EXEC rdsadmin.rdsadmin_util.alter_default_tablespace(tablespace_name => 'users2');
```

Setting the default temporary tablespace in RDS for Oracle

To set the default temporary tablespace, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.alter_default_temp_tablespace`. The `alter_default_temp_tablespace` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`tablespace_name`	varchar	—	Yes	The name of the default temporary tablespace.

The following example sets the default temporary tablespace to *`temp01`*.

```
EXEC rdsadmin.rdsadmin_util.alter_default_temp_tablespace(tablespace_name => 'temp01');
```

Creating a temporary tablespace on the instance store

To create a temporary tablespace on the instance store, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.create_inst_store_tmp_tblspace`. The `create_inst_store_tmp_tblspace` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_tablespace_name	varchar	—	Yes	The name of the temporary tablespace.

The following example creates the temporary tablespace *temp01* in the instance store.

```
EXEC rdsadmin.rdsadmin_util.create_inst_store_tmp_tblspace(p_tablespace_name =>
 'temp01');
```

Important

When you run rdsadmin_util.create_inst_store_tmp_tblspace, the newly created temporary tablespace is not automatically set as the default temporary tablespace. To set it as the default, see Setting the default temporary tablespace in RDS for Oracle.

For more information, see Storing temporary data in an RDS for Oracle instance store.

Working with tempfiles in RDS for Oracle

Adding a tempfile to the instance store on a read replica

When you create a temporary tablespace on a primary DB instance, the read replica doesn't create tempfiles. Assume that an empty temporary tablespace exists on your read replica for either of the following reasons:

- You dropped a tempfile from the tablespace on your read replica. For more information, see Dropping tempfiles on a read replica.
- You created a new temporary tablespace on the primary DB instance. In this case, RDS for Oracle synchronizes the metadata to the read replica.

You can add a tempfile to the empty temporary tablespace, and store the tempfile in the instance store. To create a tempfile in the instance store, use the Amazon RDS procedure rdsadmin.rdsadmin_util.add_inst_store_tempfile. You can use this procedure only on a read replica. The procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_tablespace_name	varchar	—	Yes	The name of the temporary tablespace on your read replica.

In the following example, the empty temporary tablespace *temp01* exists on your read replica. Run the following command to create a tempfile for this tablespace, and store it in the instance store.

```
EXEC rdsadmin.rdsadmin_util.add_inst_store_tempfile(p_tablespace_name => 'temp01');
```

For more information, see Storing temporary data in an RDS for Oracle instance store.

Dropping tempfiles on a read replica

You can't drop an existing temporary tablespace on a read replica. You can change the tempfile storage on a read replica from Amazon EBS to the instance store, or from the instance store to Amazon EBS. To achieve these goals, do the following:

1. Drop the current tempfiles in the temporary tablespace on the read replica.
2. Create new tempfiles on different storage.

To drop the tempfiles, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.drop_replica_tempfiles`. You can use this procedure only on read replicas. The `drop_replica_tempfiles` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_tablespace_name	varchar	—	Yes	The name of the temporary tablespace on your read replica.

Assume that a temporary tablespace named *temp01* resides in the instance store on your read replica. Drop all tempfiles in this tablespace by running the following command.

```
EXEC rdsadmin.rdsadmin_util.drop_replica_tempfiles(p_tablespace_name => 'temp01');
```

For more information, see Storing temporary data in an RDS for Oracle instance store.

Resizing tablespaces, data files, and tempfiles in RDS for Oracle

By default, Oracle tablespaces are created with auto-extend turned on and no maximum size. Because of these default settings, tablespaces can sometimes grow too large. We recommend that you specify an appropriate maximum size on permanent and temporary tablespaces, and that you carefully monitor space usage.

Resizing permanent tablespaces

To resize a permanent tablespace in an RDS for Oracle DB instance, use any of the following Amazon RDS procedures:

- `rdsadmin.rdsadmin_util.resize_datafile`
- `rdsadmin.rdsadmin_util.autoextend_datafile`

The `resize_datafile` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_data_file_id`	number	—	Yes	The identifier of the data file to resize.
`p_size`	varchar2	—	Yes	The size of the data file. Specify the size in bytes (the default), kilobytes (K), megabytes (M), or gigabytes (G).

The `autoextend_datafile` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_data_file_id`	number	—	Yes	The identifier of the data file to resize.

Parameter name	Data type	Default	Required	Description
p_autoextend_state	varchar2	—	Yes	The state of the autoextension feature. Specify ON to extend the data file automatic ally and OFF to turn off autoextension.
p_next	varchar2	—	No	The size of the next data file increment. Specify the size in bytes (the default), kilobytes (K), megabytes (M), or gigabytes (G).
p_maxsize	varchar2	—	No	The maximum disk space allowed for automatic extension. Specify the size in bytes (the default), kilobytes (K), megabytes (M), or gigabytes (G). You can specify UNLIMITED to remove the file size limit.

The following example resizes data file 4 to 500 MB.

```
EXEC rdsadmin.rdsadmin_util.resize_datafile(4,'500M');
```

The following example turns off autoextension for data file 4. It also turns on autoextension for data file 5, with an increment of 128 MB and no maximum size.

```
EXEC rdsadmin.rdsadmin_util.autoextend_datafile(4,'OFF');
EXEC rdsadmin.rdsadmin_util.autoextend_datafile(5,'ON','128M','UNLIMITED');
```

Resizing temporary tablespaces

To resize a temporary tablespaces in an RDS for Oracle DB instance, including a read replica, use any of the following Amazon RDS procedures:

- `rdsadmin.rdsadmin_util.resize_temp_tablespace`
- `rdsadmin.rdsadmin_util.resize_tempfile`
- `rdsadmin.rdsadmin_util.autoextend_tempfile`

The `resize_temp_tablespace` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_temp_tablespace_name`	varchar2	—	Yes	The name of the temporary tablespace to resize.
`p_size`	varchar2	—	Yes	The size of the tablespace. Specify the size in bytes (the default), kilobytes (K), megabytes (M), or gigabytes (G).

The `resize_tempfile` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_temp_file_id`	number	—	Yes	The identifier of the temp file to resize.
`p_size`	varchar2	—	Yes	The size of the temp file. Specify the size in bytes (the default), kilobytes (K), megabytes (M), or gigabytes (G).

The `autoextend_tempfile` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_temp_file_id`	number	—	Yes	The identifier of the temp file to resize.
`p_autoextend_state`	varchar2	—	Yes	The state of the autoextension feature. Specify `ON` to extend the temp file automatically and `OFF` to turn off autoextension.
`p_next`	varchar2	—	No	The size of the next temp file increment. Specify the size in bytes (the default), kilobytes (K), megabytes (M), or gigabytes (G).
`p_maxsize`	varchar2	—	No	The maximum disk space allowed for automatic extension. Specify the size in bytes (the default), kilobytes (K), megabytes (M), or gigabytes (G). You can specify `UNLIMITED` to remove the file size limit.

The following examples resize a temporary tablespace named `TEMP` to the size of 4 GB.

```
EXEC rdsadmin.rdsadmin_util.resize_temp_tablespace('TEMP','4G');
```

```
EXEC rdsadmin.rdsadmin_util.resize_temp_tablespace('TEMP','4096000000');
```

The following example resizes a temporary tablespace based on the temp file with the file identifier 1 to the size of 2 MB.

```
EXEC rdsadmin.rdsadmin_util.resize_tempfile(1,'2M');
```

The following example turns off autoextension for temp file 1. It also sets the maximum autoextension size of temp file 2 to 10 GB, with an increment of 100 MB.

```
EXEC rdsadmin.rdsadmin_util.autoextend_tempfile(1,'OFF');
EXEC rdsadmin.rdsadmin_util.autoextend_tempfile(2,'ON','100M','10G');
```

For more information about read replicas for Oracle DB instances see Working with read replicas for Amazon RDS for Oracle.

Moving data between storage volumes in RDS for Oracle

You can move data files and database objects between your primary and additional storage volumes. Before you move data, consider the following points:

- The source and target volumes must have sufficient free space.
- Data movement operations consume I/O on both volumes.
- Large data movements can impact database performance.
- If you restore a snapshot, moving data between storage volumes might be slow if it is affected by EBS lazy loading.

Topics

- Moving data files between volumes in RDS for Oracle
- Moving table data and indexes between volumes in RDS for Oracle
- Managing LOB storage using additional volumes

Moving data files between volumes in RDS for Oracle

To move data files between storage volumes, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.move_datafile`. Note the following requirements:

- You must use Oracle Enterprise Edition to run the `move_datafile` procedure.
- You can't move tablespace `SYSTEM` and `RDSADMIN`.

The `move_datafile` procedure has the following parameters.

Parameter name	Data type	Required	Description
`p_data_file_id`	number	Yes	The ID of the data file to be moved.
`p_location`	varchar2	Yes	The storage volume to which you want to move the data file.

The following example moves a tablespace from the default volume `rdsdbdata` to the additional volume `rdsdbdata2`.

```
SQL> SELECT tablespace_name,file_id,file_name FROM dba_data_files
 WHERE tablespace_name = 'MYNEWTABLESPACE';

TABLESPACE_NAME              FILE_ID FILE_NAME
------------------------- ----------
 --------------------------------------------------------------------------------
MYNEWTABLESPACE                    6 /rdsdbdata/db/ORCL_A/datafile/
o1_mf_mynewtab_n123abcd_.dbf

EXECUTE rdsadmin.rdsadmin_util.move_datafile( 6, 'rdsdbdata2');

PL/SQL procedure successfully completed.

SQL> SELECT tablespace_name,file_id,file_name FROM dba_data_files
  WHERE tablespace_name = 'MYNEWTABLESPACE';

TABLESPACE_NAME              FILE_ID FILE_NAME
------------------------- ----------
 --------------------------------------------------------------------------------
MYNEWTABLESPACE                    6 /rdsdbdata2/db/ORCL_A/datafile/
o1_mf_mynewtab_n356efgh_.dbf
```

Moving table data and indexes between volumes in RDS for Oracle

You can optimize database storage by creating tablespaces on additional storage volumes. Then you can move objects such as tables, indexes, and partitions to these tablespaces using standard

Oracle SQL. This approach is valuable for performance tuning when your database contains data with different access patterns. For example, you could store frequently accessed operational data on high-performance storage volumes while moving rarely accessed historical data to lower-cost storage volumes.

In the following example, you create a new tablespace on high-performance volume `rdsdbdata2`. Then you move a table to your additional storage volume while the table is online. You also move the index to the same volume. Moving tables and rebuilding indexes while online requires Oracle Enterprise Edition.

```
ALTER SESSION SET db_create_file_dest = '/rdsdbdata2/db';
CREATE TABLESPACE perf_tbs DATAFILE SIZE 10G;

ALTER TABLE employees
  MOVE TABLESPACE perf_tbs ONLINE;

ALTER INDEX employees_idx
  REBUILD ONLINE TABLESPACE perf_tbs;
```

In the following example, you create a tablespace on a low-cost volume. Then you move a table partition to your low-cost storage volume using an online operation.

```
ALTER SESSION SET db_create_file_dest = '/rdsdbdata3/db';
CREATE TABLESPACE hist_tbs DATAFILE SIZE 10G;

ALTER TABLE orders
  MOVE PARTITION orders_2022
  TABLESPACE hist_tbs ONLINE;
```

In the following example, you query active sessions long operations.

```
SELECT sid,opname,sofar,totalwork,time_remaining,elapsed_seconds
  FROM v$session_longops
  WHERE time_remaining > 0;
```

You can check your tablespaces usage with the following query.

```
SELECT tablespace_name, used_percent
  FROM dba_tablespace_usage_metrics
  ORDER BY used_percent DESC;
```

Managing LOB storage using additional volumes

Your database might contains tables with BLOB or CLOB objects that consume substantial storage but are infrequently accessed. To optimize storage, you can relocate these LOB segments to a tablespace on an additional storage volume.

In the following example, you create a tablespace for LOB data on a low-cost volume that is intended for low-access data. Then you create a table that stores data on this volume.

```
ALTER SESSION SET db_create_file_dest = '/rdsdbdata3/db';
CREATE TABLESPACE lob_data DATAFILE SIZE 5G AUTOEXTEND ON NEXT 1G;

CREATE TABLE documents (
    doc_id NUMBER PRIMARY KEY,
    doc_date DATE,
    doc_content CLOB
) TABLESPACE user_data
LOB(doc_content) STORE AS (TABLESPACE lob_data);
```

Working with external tables in RDS for Oracle

Oracle external tables are tables with data that is not in the database. Instead, the data is in external files that the database can access. By using external tables, you can access data without loading it into the database. For more information about external tables, see Managing external tables in the Oracle documentation.

With Amazon RDS, you can store external table files in directory objects. You can create a directory object, or you can use one that is predefined in the Oracle database, such as the DATA_PUMP_DIR directory. For information about creating directory objects, see Creating and dropping directories in the main data storage space. You can query the ALL_DIRECTORIES view to list the directory objects for your Amazon RDS Oracle DB instance.

> **Note**
>
> Directory objects point to the main data storage space (Amazon EBS volume) used by your instance. The space used—along with data files, redo logs, audit, trace, and other files—counts against allocated storage.

You can move an external data file from one Oracle database to another by using the DBMS_FILE_TRANSFER package or the UTL_FILE package. The external data file is moved from a directory on the source database to the specified directory on the destination database. For information about using DBMS_FILE_TRANSFER, see Importing using Oracle Data Pump.

After you move the external data file, you can create an external table with it. The following example creates an external table that uses the emp_xt_file1.txt file in the USER_DIR1 directory.

```
CREATE TABLE emp_xt (
  emp_id      NUMBER,
  first_name  VARCHAR2(50),
  last_name   VARCHAR2(50),
  user_name   VARCHAR2(20)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY USER_DIR1
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    FIELDS TERMINATED BY ','
    MISSING FIELD VALUES ARE NULL
    (emp_id,first_name,last_name,user_name)
  )
  LOCATION ('emp_xt_file1.txt')
)
PARALLEL
REJECT LIMIT UNLIMITED;
```

Suppose that you want to move data that is in an Amazon RDS Oracle DB instance into an external data file. In this case, you can populate the external data file by creating an external table and selecting the data from the table in the database. For example, the following SQL statement creates the orders_xt external table by querying the orders table in the database.

```
CREATE TABLE orders_xt
  ORGANIZATION EXTERNAL
   (
     TYPE ORACLE_DATAPUMP
     DEFAULT DIRECTORY DATA_PUMP_DIR
     LOCATION ('orders_xt.dmp')
   )
```

```
   AS SELECT * FROM orders;
```

In this example, the data is populated in the `orders_xt.dmp` file in the DATA_PUMP_DIR directory.

Checkpointing a database

To checkpoint the database, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.checkpoint`. The `checkpoint` procedure has no parameters.

The following example checkpoints the database.

```
EXEC rdsadmin.rdsadmin_util.checkpoint;
```

Setting distributed recovery

To set distributed recovery, use the Amazon RDS procedures `rdsadmin.rdsadmin_util.enable_distr_recovery` and `disable_distr_recovery`. The procedures have no parameters.

The following example enables distributed recovery.

```
EXEC rdsadmin.rdsadmin_util.enable_distr_recovery;
```

The following example disables distributed recovery.

```
EXEC rdsadmin.rdsadmin_util.disable_distr_recovery;
```

Setting the database time zone

You can set the time zone of your Amazon RDS Oracle database in the following ways:

- The `Timezone` option

  The `Timezone` option changes the time zone at the host level and affects all date columns and values such as SYSDATE. For more information, see Oracle time zone.
- The Amazon RDS procedure `rdsadmin.rdsadmin_util.alter_db_time_zone`

The `alter_db_time_zone` procedure changes the time zone for only certain data types, and doesn't change SYSDATE. There are additional restrictions on setting the time zone listed in the Oracle documentation.

> **Note**
>
> You can also set the default time zone for Oracle Scheduler. For more information, see Setting the time zone for Oracle Scheduler jobs.

The `alter_db_time_zone` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_new_tz	varchar2	—	Yes	The new time zone as a named region or an absolute offset from Coordinated Universal Time (UTC). Valid offsets range from -12:00 to +14:00.

The following example changes the time zone to UTC plus three hours.

```
EXEC rdsadmin.rdsadmin_util.alter_db_time_zone(p_new_tz => '+3:00');
```

The following example changes the time zone to the Africa/Algiers time zone.

```
EXEC rdsadmin.rdsadmin_util.alter_db_time_zone(p_new_tz => 'Africa/Algiers');
```

After you alter the time zone by using the `alter_db_time_zone` procedure, reboot your DB instance for the change to take effect. For more information, see Rebooting a DB instance. For information about upgrading time zones, see Time zone considerations.

Generating performance reports with Automatic Workload Repository (AWR)

To gather performance data and generate reports, Oracle recommends Automatic Workload Repository (AWR). AWR requires Oracle Database Enterprise Edition and a license for the Diagnostics and Tuning packs. To enable AWR, set the `CONTROL_MANAGEMENT_PACK_ACCESS` initialization parameter to either `DIAGNOSTIC` or `DIAGNOSTIC+TUNING`.

Working with AWR reports in RDS

To generate AWR reports, you can run scripts such as `awrrpt.sql`. These scripts are installed on the database host server. In Amazon RDS, you don't have direct access to the host. However, you can get copies of SQL scripts from another installation of Oracle Database.

You can also use AWR by running procedures in the `SYS.DBMS_WORKLOAD_REPOSITORY` PL/SQL package. You can use this package to manage baselines and snapshots, and also to display ASH and AWR reports. For example, to generate an AWR report in text format run the `DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT` procedure. However, you can't reach these AWR reports from the AWS Management Console.

When working with AWR, we recommend using the `rdsadmin.rdsadmin_diagnostic_util` procedures. You can use these procedures to generate the following:

- AWR reports
- Active Session History (ASH) reports
- Automatic Database Diagnostic Monitor (ADDM) reports
- Oracle Data Pump Export dump files of AWR data

The `rdsadmin_diagnostic_util` procedures save the reports to the DB instance file system. You can access these reports from the console. You can also access reports using the `rdsadmin.rds_file_util` procedures, and you can access reports that are copied to Amazon S3 using the S3 Integration option. For more information, see Reading files in a DB instance directory and Amazon S3 integration.

You can use the `rdsadmin_diagnostic_util` procedures in the following Amazon RDS for Oracle DB engine versions:

- All Oracle Database 21c versions

- 19.0.0.0.ru-2020-04.rur-2020-04.r1 and higher Oracle Database 19c versions

For a blog that explains how to work with diagnostic reports in a replication scenario, see Generate AWR reports for Amazon RDS for Oracle read replicas.

Common parameters for the diagnostic utility package

You typically use the following parameters when managing AWR and ADDM with the `rdsadmin_diagnostic_util` package.

Parameter	Data type	Defau	Requi	Description
`begin_snap_id`	NUMBEI	—	Yes	The ID of the beginning snapshot.
`end_snap_id`	NUMBEI	—	Yes	The ID of the ending snapshot.
`dump_directory`	VARCHA	BDUMI	No	The directory to write the report or export file to. If you specify a nondefault directory, the user that runs the `rdsadmin_diagnostic_util` procedures must have write permissions for the directory.
`p_tag`	VARCHA	—	No	A string that can be used to distinguish between backups to indicate the purpose or usage of backups, such as `incremental` or `daily`. You can specify up to 30 characters. Valid characters are `a-z`, `A-Z`, `0-9`, an underscore (_), a dash (-), and a period (.). The tag is not case-sensitive. RMAN always stores tags in uppercase, regardless of the case used when entering them. Tags don't need to be unique, so multiple backups can have the same tag. If you don't specify a tag, RMAN assigns a default tag automatically using the format TAG*YYYYMMDDTHHMMSS*, where *YYYY* is the year, *MM* is the month, *DD* is the day, *HH* is the hour (in 24-hour

Parameter	Data type	Defau	Requi	Description
				format), *MM* is the minutes, and *SS* is the seconds. The date and time indicate when RMAN started the backup. For example, a backup with the default tagTAG20190927T214517 indicates a backup that started on 2019-09-27 at 21:45:17. The p_tag parameter is supported for the following RDS for Oracle DB engine versions: • Oracle Database 21c (21.0.0) • Oracle Database 19c (19.0.0), using 19.0.0.0.ru-2021-10.rur-2021-10.r1 and higher
report_type	VARCHA	HTML	No	The format of the report. Valid values are TEXT and HTML.
dbid	NUMBEF	—	No	A valid database identifier (DBID) shown in the DBA_HIST_DATABASE_INSTANCE view for Oracle. If this parameter is not specified, RDS uses the current DBID, which is shown in the V$DATABASE.DBID view.

You typically use the following parameters when managing ASH with the rdsadmin_diagnostic_util package.

Parameter	Data type	Defaul	Requir	Description
begin_time	DATE	—	Yes	The beginning time of the ASH analysis.
end_time	DATE	—	Yes	The ending time of the ASH analysis.
slot_width	NUMBER	0	No	The duration of the slots (in seconds) used in the "Top Activity" section of the ASH report. If this parameter isn't specified, the time interval between

Parameter	Data type	Defaul	Requir	Description
				`begin_time` and `end_time` uses no more than 10 slots.
`sid`	`NUMBER`	Null	No	The session ID.
`sql_id`	`VARCHAR2`	Null	No	The SQL ID.
`wait_class`	`VARCHAR2`	Null	No	The wait class name.
`service_hash`	`NUMBER`	Null	No	The service name hash.
`module_name`	`VARCHAR2`	Null	No	The module name.
`action_name`	`VARCHAR2`	Null	No	The action name.
`client_id`	`VARCHAR2`	Null	No	The application-specific ID of the database session.
`plsql_entry`	`VARCHAR2`	Null	No	The PL/SQL entry point.

Generating an AWR report

To generate an AWR report, use the `rdsadmin.rdsadmin_diagnostic_util.awr_report` procedure.

The following example generates a AWR report for the snapshot range 101–106. The output text file is named `awrrpt_101_106.txt`. You can access this report from the AWS Management Console.

```
EXEC rdsadmin.rdsadmin_diagnostic_util.awr_report(101,106,'TEXT');
```

The following example generates an HTML report for the snapshot range 63–65. The output HTML file is named `awrrpt_63_65.html`. The procedure writes the report to the nondefault database directory named `AWR_RPT_DUMP`.

```
EXEC rdsadmin.rdsadmin_diagnostic_util.awr_report(63,65,'HTML','AWR_RPT_DUMP');
```

Extracting AWR data into a dump file

To extract AWR data into a dump file, use the `rdsadmin.rdsadmin_diagnostic_util.awr_extract` procedure. You can use this function only at the PDB level.

The following example extracts the snapshot range 101–106. The output dump file is named `awrextract_101_106.dmp`. You can access this file through the console.

```
EXEC rdsadmin.rdsadmin_diagnostic_util.awr_extract(101,106);
```

The following example extracts the snapshot range 63–65. The output dump file is named `awrextract_63_65.dmp`. The file is stored in the nondefault database directory named `AWR_RPT_DUMP`.

```
EXEC rdsadmin.rdsadmin_diagnostic_util.awr_extract(63,65,'AWR_RPT_DUMP');
```

Generating an ADDM report

To generate an ADDM report, use the `rdsadmin.rdsadmin_diagnostic_util.addm_report` procedure.

The following example generates an ADDM report for the snapshot range 101–106. The output text file is named `addmrpt_101_106.txt`. You can access the report through the console.

```
EXEC rdsadmin.rdsadmin_diagnostic_util.addm_report(101,106);
```

The following example generates an ADDM report for the snapshot range 63–65. The output text file is named `addmrpt_63_65.txt`. The file is stored in the nondefault database directory named `ADDM_RPT_DUMP`.

```
EXEC rdsadmin.rdsadmin_diagnostic_util.addm_report(63,65,'ADDM_RPT_DUMP');
```

Generating an ASH report

To generate an ASH report, use the `rdsadmin.rdsadmin_diagnostic_util.ash_report` procedure.

The following example generates an ASH report that includes the data from 14 minutes ago until the current time. The name of the output file uses the format `ashrpt`*`begin_time`*`end_time`*`.txt`*, where *`begin_time`* and *`end_time`* use the format `YYYYMMDDHH24MISS`. You can access the file through the console.

```
BEGIN
    rdsadmin.rdsadmin_diagnostic_util.ash_report(
        begin_time     =>     SYSDATE-14/1440,
        end_time       =>     SYSDATE,
        report_type    =>     'TEXT');
END;
/
```

The following example generates an ASH report that includes the data from November 18, 2019, at 6:07 PM through November 18, 2019, at 6:15 PM. The name of the output HTML report is `ashrpt_20190918180700_20190918181500.html`. The report is stored in the nondefault database directory named `AWR_RPT_DUMP`.

```
BEGIN
    rdsadmin.rdsadmin_diagnostic_util.ash_report(
        begin_time     =>    TO_DATE('2019-09-18 18:07:00', 'YYYY-MM-DD HH24:MI:SS'),
        end_time       =>    TO_DATE('2019-09-18 18:15:00', 'YYYY-MM-DD HH24:MI:SS'),
        report_type    =>    'html',
        dump_directory =>    'AWR_RPT_DUMP');
END;
/
```

Accessing AWR reports from the console or CLI

To access AWR reports or export dump files, you can use the AWS Management Console or AWS CLI. For more information, see Downloading a database log file.

Adjusting database links for use with DB instances in a VPC

To use Oracle database links with Amazon RDS DB instances inside the same virtual private cloud (VPC) or peered VPCs, the two DB instances should have a valid route between them. Verify the

valid route between the DB instances by using your VPC routing tables and network access control list (ACL).

The security group of each DB instance must allow ingress to and egress from the other DB instance. The inbound and outbound rules can refer to security groups from the same VPC or a peered VPC. For more information, see Updating your security groups to reference peered VPC security groups.

If you have configured a custom DNS server using the DHCP Option Sets in your VPC, your custom DNS server must be able to resolve the name of the database link target. For more information, see Setting up a custom DNS server.

For more information about using database links with Oracle Data Pump, see Importing using Oracle Data Pump.

Setting the default edition for a DB instance

You can redefine database objects in a private environment called an edition. You can use edition-based redefinition to upgrade an application's database objects with minimal downtime.

You can set the default edition of an Amazon RDS Oracle DB instance using the Amazon RDS procedure `rdsadmin.rdsadmin_util.alter_default_edition`.

The following example sets the default edition for the Amazon RDS Oracle DB instance to `RELEASE_V1`.

```
EXEC rdsadmin.rdsadmin_util.alter_default_edition('RELEASE_V1');
```

The following example sets the default edition for the Amazon RDS Oracle DB instance back to the Oracle default.

```
EXEC rdsadmin.rdsadmin_util.alter_default_edition('ORA$BASE');
```

For more information about Oracle edition-based redefinition, see About editions and edition-based redefinition in the Oracle documentation.

Enabling auditing for the SYS.AUD$ table

To enable auditing on the database audit trail table SYS.AUD$, use the Amazon RDS procedure `rdsadmin.rdsadmin_master_util.audit_all_sys_aud_table`. The only supported audit property is ALL. You can't audit or not audit individual statements or operations.

Enabling auditing is supported for Oracle DB instances running the following versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

The `audit_all_sys_aud_table` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_by_access	boolean	true	No	Set to `true` to audit BY ACCESS. Set to `false` to audit BY SESSION.

The following query returns the current audit configuration for SYS.AUD$ for a database.

```
SELECT * FROM DBA_OBJ_AUDIT_OPTS WHERE OWNER='SYS' AND OBJECT_NAME='AUD$';
```

The following commands enable audit of ALL on SYS.AUD$ BY ACCESS.

```
EXEC rdsadmin.rdsadmin_master_util.audit_all_sys_aud_table;

EXEC rdsadmin.rdsadmin_master_util.audit_all_sys_aud_table(p_by_access => true);
```

The following command enables audit of ALL on SYS.AUD$ BY SESSION.

```
EXEC rdsadmin.rdsadmin_master_util.audit_all_sys_aud_table(p_by_access => false);
```

For more information, see AUDIT (traditional auditing) in the Oracle documentation.

Disabling auditing for the SYS.AUD$ table

To disable auditing on the database audit trail table SYS.AUD$, use the Amazon RDS procedure `rdsadmin.rdsadmin_master_util.noaudit_all_sys_aud_table`. This procedure takes no parameters.

The following query returns the current audit configuration for SYS.AUD$ for a database:

```
SELECT * FROM DBA_OBJ_AUDIT_OPTS WHERE OWNER='SYS' AND OBJECT_NAME='AUD$';
```

The following command disables audit of ALL on SYS.AUD$.

```
EXEC rdsadmin.rdsadmin_master_util.noaudit_all_sys_aud_table;
```

For more information, see NOAUDIT (traditional auditing) in the Oracle documentation.

Cleaning up interrupted online index builds

To clean up failed online index builds, use the Amazon RDS procedure `rdsadmin.rdsadmin_dbms_repair.online_index_clean`.

The `online_index_clean` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`object_id`	binary_integer	`ALL_INDEX_ID`	No	The object ID of the index. Typically, you can use the object ID from the ORA-08104 error text.
`wait_for_lock`	binary_integer	`rdsadmin.rdsadmin_dbms_repair.lock_wait`	No	Specify `rdsadmin.rdsadmin_dbms_repair.lock_wait` , the default, to try to get a lock on the underlying object and retry until an

Parameter name	Data type	Default	Required	Description
				internal limit is reached if the lock fails. Specify `rdsadmin.rdsadmin_dbms_repair.lock_nowait` to try to get a lock on the underlying object but not retry if the lock fails.

The following example cleans up a failed online index build:

```
declare
  is_clean boolean;
begin
  is_clean := rdsadmin.rdsadmin_dbms_repair.online_index_clean(
    object_id     => 1234567890,
    wait_for_lock => rdsadmin.rdsadmin_dbms_repair.lock_nowait
  );
end;
/
```

For more information, see ONLINE_INDEX_CLEAN function in the Oracle documentation.

Skipping corrupt blocks

To skip corrupt blocks during index and table scans, use the `rdsadmin.rdsadmin_dbms_repair` package.

The following procedures wrap the functionality of the `sys.dbms_repair.admin_table` procedure and take no parameters:

- `rdsadmin.rdsadmin_dbms_repair.create_repair_table`
- `rdsadmin.rdsadmin_dbms_repair.create_orphan_keys_table`
- `rdsadmin.rdsadmin_dbms_repair.drop_repair_table`

- `rdsadmin.rdsadmin_dbms_repair.drop_orphan_keys_table`
- `rdsadmin.rdsadmin_dbms_repair.purge_repair_table`
- `rdsadmin.rdsadmin_dbms_repair.purge_orphan_keys_table`

The following procedures take the same parameters as their counterparts in the `DBMS_REPAIR` package for Oracle databases:

- `rdsadmin.rdsadmin_dbms_repair.check_object`
- `rdsadmin.rdsadmin_dbms_repair.dump_orphan_keys`
- `rdsadmin.rdsadmin_dbms_repair.fix_corrupt_blocks`
- `rdsadmin.rdsadmin_dbms_repair.rebuild_freelists`
- `rdsadmin.rdsadmin_dbms_repair.segment_fix_status`
- `rdsadmin.rdsadmin_dbms_repair.skip_corrupt_blocks`

For more information about handling database corruption, see DBMS_REPAIR in the Oracle documentation.

Example Responding to corrupt blocks

This example shows the basic workflow for responding to corrupt blocks. Your steps will depend on the location and nature of your block corruption.

> **Important**
>
> Before attempting to repair corrupt blocks, review the DBMS_REPAIR documentation carefully.

To skip corrupt blocks during index and table scans

1. Run the following procedures to create repair tables if they don't already exist.

```
EXEC rdsadmin.rdsadmin_dbms_repair.create_repair_table;
EXEC rdsadmin.rdsadmin_dbms_repair.create_orphan_keys_table;
```

2. Run the following procedures to check for existing records and purge them if appropriate.

```
SELECT COUNT(*) FROM SYS.REPAIR_TABLE;
SELECT COUNT(*) FROM SYS.ORPHAN_KEY_TABLE;
SELECT COUNT(*) FROM SYS.DBA_REPAIR_TABLE;
SELECT COUNT(*) FROM SYS.DBA_ORPHAN_KEY_TABLE;

EXEC rdsadmin.rdsadmin_dbms_repair.purge_repair_table;
EXEC rdsadmin.rdsadmin_dbms_repair.purge_orphan_keys_table;
```

3. Run the following procedure to check for corrupt blocks.

```
SET SERVEROUTPUT ON
DECLARE v_num_corrupt INT;
BEGIN
  v_num_corrupt := 0;
  rdsadmin.rdsadmin_dbms_repair.check_object (
    schema_name => '&corruptionOwner',
    object_name => '&corruptionTable',
    corrupt_count =>  v_num_corrupt
  );
  dbms_output.put_line('number corrupt: '||to_char(v_num_corrupt));
END;
/

COL CORRUPT_DESCRIPTION FORMAT a30
COL REPAIR_DESCRIPTION FORMAT a30

SELECT OBJECT_NAME, BLOCK_ID, CORRUPT_TYPE, MARKED_CORRUPT,
       CORRUPT_DESCRIPTION, REPAIR_DESCRIPTION
FROM   SYS.REPAIR_TABLE;

SELECT SKIP_CORRUPT
FROM   DBA_TABLES
WHERE  OWNER = '&corruptionOwner'
AND    TABLE_NAME = '&corruptionTable';
```

4. Use the `skip_corrupt_blocks` procedure to enable or disable corruption skipping for affected tables. Depending on the situation, you may also need to extract data to a new table, and then drop the table containing the corrupt block.

   Run the following procedure to enable corruption skipping for affected tables.

```
begin
```

```
  rdsadmin.rdsadmin_dbms_repair.skip_corrupt_blocks (
    schema_name => '&corruptionOwner',
    object_name => '&corruptionTable',
    object_type => rdsadmin.rdsadmin_dbms_repair.table_object,
    flags => rdsadmin.rdsadmin_dbms_repair.skip_flag);
end;
/
select skip_corrupt from dba_tables where owner = '&corruptionOwner' and table_name
 = '&corruptionTable';
```

Run the following procedure to disable corruption skipping.

```
begin
  rdsadmin.rdsadmin_dbms_repair.skip_corrupt_blocks (
    schema_name => '&corruptionOwner',
    object_name => '&corruptionTable',
    object_type => rdsadmin.rdsadmin_dbms_repair.table_object,
    flags => rdsadmin.rdsadmin_dbms_repair.noskip_flag);
end;
/

select skip_corrupt from dba_tables where owner = '&corruptionOwner' and table_name
 = '&corruptionTable';
```

5. When you have completed all repair work, run the following procedures to drop the repair tables.

```
EXEC rdsadmin.rdsadmin_dbms_repair.drop_repair_table;
EXEC rdsadmin.rdsadmin_dbms_repair.drop_orphan_keys_table;
```

Purging the recycle bin

When you drop a table, your Oracle database doesn't immediately remove its storage space. The database renames the table and places it and any associated objects in a recycle bin. Purging the recycle bin removes these items and releases their storage space.

To purge the entire recycle bin, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.purge_dba_recyclebin`. However, this procedure can't purge the recycle bin of `SYS` and `RDSADMIN` objects. If you need to purge these objects, contact AWS Support.

The following example purges the entire recycle bin.

```
EXEC rdsadmin.rdsadmin_util.purge_dba_recyclebin;
```

Setting the default displayed values for full redaction

To change the default displayed values for full redaction on your Amazon RDS Oracle instance, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.dbms_redact_upd_full_rdct_val`. Note that you create a redaction policy with the `DBMS_REDACT` PL/SQL package, as explained in the Oracle Database documentation. The `dbms_redact_upd_full_rdct_val` procedure specifies the characters to display for different data types affected by an existing policy.

The `dbms_redact_upd_full_rdct_val` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_number_val`	number	Null	No	Modifies the default value for columns of the `NUMBER` data type.
`p_binfloat_val`	binary_float	Null	No	Modifies the default value for columns of the `BINARY_FLOAT` data type.
`p_bindouble_val`	binary_double	Null	No	Modifies the default value for columns of the `BINARY_DOUBLE` data type.
`p_char_val`	char	Null	No	Modifies the default value for columns of the `CHAR` data type.
`p_varchar_val`	varchar2	Null	No	Modifies the default value for columns of the `VARCHAR2` data type.

Parameter name	Data type	Default	Required	Description
p_nchar_val	nchar	Null	No	Modifies the default value for columns of the NCHAR data type.
p_nvarchar_val	nvarchar2	Null	No	Modifies the default value for columns of the NVARCHAR2 data type.
p_date_val	date	Null	No	Modifies the default value for columns of the DATE data type.
p_ts_val	timestamp	Null	No	Modifies the default value for columns of the TIMESTAMP data type.
p_tswtz_val	timestamp with time zone	Null	No	Modifies the default value for columns of the TIMESTAMP WITH TIME ZONE data type.
p_blob_val	blob	Null	No	Modifies the default value for columns of the BLOB data type.
p_clob_val	clob	Null	No	Modifies the default value for columns of the CLOB data type.
p_nclob_val	nclob	Null	No	Modifies the default value for columns of the NCLOB data type.

The following example changes the default redacted value to * for the CHAR data type:

```
EXEC rdsadmin.rdsadmin_util.dbms_redact_upd_full_rdct_val(p_char_val => '*');
```

The following example changes the default redacted values for NUMBER, DATE, and CHAR data types:

```
BEGIN
rdsadmin.rdsadmin_util.dbms_redact_upd_full_rdct_val(
    p_number_val=>1,
    p_date_val=>to_date('1900-01-01','YYYY-MM-DD'),
    p_varchar_val=>'X');
END;
/
```

After you alter the default values for full redaction with the dbms_redact_upd_full_rdct_val procedure, reboot your DB instance for the change to take effect. For more information, see Rebooting a DB instance.

Performing common log-related tasks for Oracle DB instances

Following, you can find how to perform certain common DBA tasks related to logging on your Amazon RDS DB instances running Oracle. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances, and restricts access to certain system procedures and tables that require advanced privileges.

For more information, see Amazon RDS for Oracle database log files.

Topics

- Setting force logging
- Setting supplemental logging
- Switching online log files
- Adding online redo logs
- Dropping online redo logs
- Resizing online redo logs
- Retaining archived redo logs
- Accessing online and archived redo logs
- Downloading archived redo logs from Amazon S3

Setting force logging

In force logging mode, Oracle logs all changes to the database except changes in temporary tablespaces and temporary segments (NOLOGGING clauses are ignored). For more information, see Specifying FORCE LOGGING mode in the Oracle documentation.

To set force logging, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.force_logging`. The `force_logging` procedure has the following parameters.

Parameter name	Data type	Default	Yes	Description
`p_enable`	boolean	true	No	Set to `true` to put the database in force logging mode, `false` to remove the database from force logging mode.

The following example puts the database in force logging mode.

```
EXEC rdsadmin.rdsadmin_util.force_logging(p_enable => true);
```

Setting supplemental logging

If you enable supplemental logging, LogMiner has the necessary information to support chained rows and clustered tables. For more information, see Supplemental logging in the Oracle documentation.

Oracle Database doesn't enable supplemental logging by default. To enable and disable supplemental logging, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.alter_supplemental_logging`. For more information about how Amazon RDS manages the retention of archived redo logs for Oracle DB instances, see Retaining archived redo logs.

The `alter_supplemental_logging` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_action	varchar2	—	Yes	'ADD' to add supplemental logging, 'DROP' to drop supplemental logging.
p_type	varchar2	null	No	The type of supplemental logging. Valid values are 'ALL', 'FOREIGN KEY', 'PRIMARY KEY', 'UNIQUE', or PROCEDURAL .

The following example enables supplemental logging.

```
begin
    rdsadmin.rdsadmin_util.alter_supplemental_logging(
        p_action => 'ADD');
end;
/
```

The following example enables supplemental logging for all fixed-length maximum size columns.

```
begin
    rdsadmin.rdsadmin_util.alter_supplemental_logging(
        p_action => 'ADD',
        p_type   => 'ALL');
end;
/
```

The following example enables supplemental logging for primary key columns.

```
begin
    rdsadmin.rdsadmin_util.alter_supplemental_logging(
        p_action => 'ADD',
        p_type   => 'PRIMARY KEY');
end;
```

```
/
```

Switching online log files

To switch log files, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.switch_logfile`. The `switch_logfile` procedure has no parameters.

The following example switches log files.

```
EXEC rdsadmin.rdsadmin_util.switch_logfile;
```

Adding online redo logs

An Amazon RDS DB instance running Oracle starts with four online redo logs, 128 MB each. To add additional redo logs, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.add_logfile`.

The `add_logfile` procedure has the following parameters.

> **Note**
>
> The parameters are mutually exclusive.

Parameter name	Data type	Default	Required	Description
`bytes`	positive	null	No	The size of the log file in bytes. Use this parameter only if the size of the log is under 2147483648 bytes (2 GiB). Otherwise, RDS issues an error. For log sizes above this byte value, use the `p_size` parameter instead.

Parameter name	Data type	Default	Required	Description
p_size	varchar2	—	Yes	The size of the log file in kilobytes (K), megabytes (M), or gigabytes (G).

The following command adds a 100 MB log file.

```
EXEC rdsadmin.rdsadmin_util.add_logfile(p_size => '100M');
```

Dropping online redo logs

To drop redo logs, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.drop_logfile`. The `drop_logfile` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
grp	positive	—	Yes	The group number of the log.

The following example drops the log with group number 3.

```
EXEC rdsadmin.rdsadmin_util.drop_logfile(grp => 3);
```

You can only drop logs that have a status of unused or inactive. The following example gets the statuses of the logs.

```
SELECT GROUP#, STATUS FROM V$LOG;

GROUP#     STATUS
---------- ----------------
1          CURRENT
2          INACTIVE
3          INACTIVE
4          UNUSED
```

Resizing online redo logs

An Amazon RDS DB instance running Oracle starts with four online redo logs, 128 MB each. The following example shows how you can use Amazon RDS procedures to resize your logs from 128 MB each to 512 MB each.

```
/* Query V$LOG to see the logs.          */
/* You start with 4 logs of 128 MB each. */

SELECT GROUP#, BYTES, STATUS FROM V$LOG;

GROUP#     BYTES      STATUS
---------- ---------- ----------------
1          134217728  INACTIVE
2          134217728  CURRENT
3          134217728  INACTIVE
4          134217728  INACTIVE


/* Add four new logs that are each 512 MB */

EXEC rdsadmin.rdsadmin_util.add_logfile(bytes => 536870912);
EXEC rdsadmin.rdsadmin_util.add_logfile(bytes => 536870912);
EXEC rdsadmin.rdsadmin_util.add_logfile(bytes => 536870912);
EXEC rdsadmin.rdsadmin_util.add_logfile(bytes => 536870912);


/* Query V$LOG to see the logs. */
/* Now there are 8 logs.        */

SELECT GROUP#, BYTES, STATUS FROM V$LOG;

GROUP#     BYTES      STATUS
---------- ---------- ----------------
1          134217728  INACTIVE
2          134217728  CURRENT
3          134217728  INACTIVE
4          134217728  INACTIVE
5          536870912  UNUSED
6          536870912  UNUSED
7          536870912  UNUSED
8          536870912  UNUSED
```

```
/* Drop each inactive log using the group number. */

EXEC rdsadmin.rdsadmin_util.drop_logfile(grp => 1);
EXEC rdsadmin.rdsadmin_util.drop_logfile(grp => 3);
EXEC rdsadmin.rdsadmin_util.drop_logfile(grp => 4);


/* Query V$LOG to see the logs. */
/* Now there are 5 logs.        */

select GROUP#, BYTES, STATUS from V$LOG;

GROUP#     BYTES      STATUS
---------- ---------- ----------------
2          134217728  CURRENT
5          536870912  UNUSED
6          536870912  UNUSED
7          536870912  UNUSED
8          536870912  UNUSED


/* Switch logs so that group 2 is no longer current. */

EXEC rdsadmin.rdsadmin_util.switch_logfile;


/* Query V$LOG to see the logs.        */
/* Now one of the new logs is current. */

SQL>SELECT GROUP#, BYTES, STATUS FROM V$LOG;

GROUP#     BYTES      STATUS
---------- ---------- ----------------
2          134217728  ACTIVE
5          536870912  CURRENT
6          536870912  UNUSED
7          536870912  UNUSED
8          536870912  UNUSED


/* If the status of log 2 is still "ACTIVE", issue a checkpoint to clear it to
 "INACTIVE".  */
```

```
EXEC rdsadmin.rdsadmin_util.checkpoint;


/* Query V$LOG to see the logs.            */
/* Now the final original log is inactive. */

select GROUP#, BYTES, STATUS from V$LOG;

GROUP#     BYTES      STATUS
---------- ---------- ----------------
2          134217728  INACTIVE
5          536870912  CURRENT
6          536870912  UNUSED
7          536870912  UNUSED
8          536870912  UNUSED


# Drop the final inactive log.

EXEC rdsadmin.rdsadmin_util.drop_logfile(grp => 2);


/* Query V$LOG to see the logs.    */
/* Now there are four 512 MB logs. */

SELECT GROUP#, BYTES, STATUS FROM V$LOG;

GROUP#     BYTES      STATUS
---------- ---------- ----------------
5          536870912  CURRENT
6          536870912  UNUSED
7          536870912  UNUSED
8          536870912  UNUSED
```

Retaining archived redo logs

You can retain archived redo logs locally on your DB instance for use with products like Oracle LogMiner (`DBMS_LOGMNR`). After you have retained the redo logs, you can use LogMiner to analyze the logs. For more information, see Using LogMiner to analyze redo log files in the Oracle documentation.

To retain archived redo logs, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.set_configuration`. If you use this procedure on a primary

instance in Oracle Data Guard, RDS changes the archive log retention setting on the primary instance and open read replicas, but not on mounted replicas. RDS retains the latest archive redo logs on mounted replicas for a short period of time. RDS automatically deletes older logs downloaded to mounted replicas.

The `set_configuration` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`name`	varchar	—	Yes	The name of the configuration to update. To change the archived redo log retention hours, set the name to `archivelog retention hours`.
`value`	varchar	—	Yes	The value for the configuration. Set the value the number of hours to retain the logs.

The following example retains 24 hours of redo logs.

```
begin
    rdsadmin.rdsadmin_util.set_configuration(
        name  => 'archivelog retention hours',
        value => '24');
end;
/
commit;
```

Note

The commit is required for the change to take effect.

To view how long archived redo logs are kept for your DB instance, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.show_configuration`.

The following example shows the log retention time.

```
set serveroutput on
EXEC rdsadmin.rdsadmin_util.show_configuration;
```

The output shows the current setting for `archivelog retention hours`. The following output shows that archived redo logs are kept for 48 hours.

```
NAME:archivelog retention hours
VALUE:48
DESCRIPTION:ArchiveLog expiration specifies the duration in hours before archive/redo
 log files are automatically deleted.
```

Because the archived redo logs are retained on your DB instance, ensure that your DB instance has enough allocated storage for the retained logs. To determine how much space your DB instance has used in the last X hours, you can run the following query, replacing X with the number of hours.

```
SELECT SUM(BLOCKS * BLOCK_SIZE) bytes
  FROM V$ARCHIVED_LOG
 WHERE FIRST_TIME >= SYSDATE-(X/24) AND DEST_ID=1;
```

RDS for Oracle only generates archived redo logs when the backup retention period of your DB instance is greater than zero. By default the backup retention period is greater than zero.

When the archived log retention period expires, RDS for Oracle removes the archived redo logs from your DB instance. To support restoring your DB instance to a point in time, Amazon RDS retains the archived redo logs outside of your DB instance based on the backup retention period. To modify the backup retention period, see Modifying an Amazon RDS DB instance.

Note

In some cases, you might be using JDBC on Linux to download archived redo logs and experience long latency times and connection resets. In such cases, the issues might be caused by the default random number generator setting on your Java client. We recommend setting your JDBC drivers to use a nonblocking random number generator.

Accessing online and archived redo logs

You might want to access your online and archived redo log files for mining with external tools such as GoldenGate, Attunity, Informatica, and others. To access these files, do the following:

1. Create directory objects that provide read-only access to the physical file paths.

   Use `rdsadmin.rdsadmin_master_util.create_archivelog_dir` and `rdsadmin.rdsadmin_master_util.create_onlinelog_dir`.
2. Read the files using PL/SQL.

   You can read the files by using PL/SQL. For more information about reading files from directory objects, see Listing files in a DB instance directory and Reading files in a DB instance directory.

Accessing transaction logs is supported for the following releases:

- Oracle Database 21c
- Oracle Database 19c

The following code creates directories that provide read-only access to your online and archived redo log files:

> **Important**
>
> This code also revokes the `DROP ANY DIRECTORY` privilege.

```
EXEC rdsadmin.rdsadmin_master_util.create_archivelog_dir;
EXEC rdsadmin.rdsadmin_master_util.create_onlinelog_dir;
```

The following code drops the directories for your online and archived redo log files.

```
EXEC rdsadmin.rdsadmin_master_util.drop_archivelog_dir;
EXEC rdsadmin.rdsadmin_master_util.drop_onlinelog_dir;
```

The following code grants and revokes the `DROP ANY DIRECTORY` privilege.

```
EXEC rdsadmin.rdsadmin_master_util.revoke_drop_any_directory;
```

```
EXEC rdsadmin.rdsadmin_master_util.grant_drop_any_directory;
```

Downloading archived redo logs from Amazon S3

You can download archived redo logs on your DB instance using the `rdsadmin.rdsadmin_archive_log_download` package. If archived redo logs are no longer on your DB instance, you might want to download them again from Amazon S3. Then you can mine the logs or use them to recover or replicate your database.

> **Note**
>
> You can't download archived redo logs on read replica instances.

Downloading archived redo logs: basic steps

The availability of your archived redo logs depends on the following retention policies:

- Backup retention policy – Logs inside of this policy are available in Amazon S3. Logs outside of this policy are removed.
- Archived log retention policy – Logs inside of this policy are available on your DB instance. Logs outside of this policy are removed.

If logs aren't on your instance but are protected by your backup retention period, use `rdsadmin.rdsadmin_archive_log_download` to download them again. RDS for Oracle saves the logs to the `/rdsdbdata/log/arch` directory on your DB instance.

To download archived redo logs from Amazon S3

1. Configure your retention period to ensure your downloaded archived redo logs are retained for the duration you need them. Make sure to `COMMIT` your change.

   RDS retains your downloaded logs according to the archived log retention policy, starting from the time the logs were downloaded. To learn how to set the retention policy, see Retaining archived redo logs.
2. Wait up to 5 minutes for the archived log retention policy change to take effect.
3. Download the archived redo logs from Amazon S3 using `rdsadmin.rdsadmin_archive_log_download`.

For more information, see Downloading a single archived redo log and Downloading a series of archived redo logs.

> **Note**
>
> RDS automatically checks the available storage before downloading. If the requested logs consume a high percentage of space, you receive an alert.

4. Confirm that the logs were downloaded from Amazon S3 successfully.

   You can view the status of your download task in a bdump file. The bdump files have the path name `/rdsdbdata/log/trace/dbtask-task-id.log`. In the preceding download step, you run a `SELECT` statement that returns the task ID in a `VARCHAR2` data type. For more information, see similar examples in Monitoring the status of a file transfer.

Downloading a single archived redo log

To download a single archived redo log to the `/rdsdbdata/log/arch` directory, use `rdsadmin.rdsadmin_archive_log_download.download_log_with_seqnum`. This procedure has the following parameter.

Parameter name	Data type	Default	Required	Description
`seqnum`	number	—	Yes	The sequence number of the archived redo log.

The following example downloads the log with sequence number 20.

```
SELECT rdsadmin.rdsadmin_archive_log_download.download_log_with_seqnum(seqnum => 20)
       AS TASK_ID
FROM   DUAL;
```

Downloading a series of archived redo logs

To download a series of archived redo logs to the `/rdsdbdata/log/arch` directory, use `download_logs_in_seqnum_range`. Your download is limited to 300 logs per request. The `download_logs_in_seqnum_range` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
start_seq	number	—	Yes	The starting sequence number for the series.
end_seq	number	—	Yes	The ending sequence number for the series.

The following example downloads the logs from sequence 50 to 100.

```
SELECT rdsadmin.rdsadmin_archive_log_download.download_logs_in_seqnum_range(start_seq
 => 50, end_seq => 100)
       AS TASK_ID
FROM   DUAL;
```

Performing common RMAN tasks for Oracle DB instances

In the following section, you can find how you can perform Oracle Recovery Manager (RMAN) DBA tasks on your Amazon RDS DB instances running Oracle. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances. It also restricts access to certain system procedures and tables that require advanced privileges.

Use the Amazon RDS package `rdsadmin.rdsadmin_rman_util` to perform RMAN backups of your Amazon RDS for Oracle database to disk. The `rdsadmin.rdsadmin_rman_util` package supports full and incremental database file backups, tablespace backups, and archived redo log backups.

After an RMAN backup has finished, you can copy the backup files off the Amazon RDS for Oracle DB instance host. You might do this for the purpose of restoring to a non-RDS host or for long-term storage of backups. For example, you can copy the backup files to an Amazon S3 bucket. For more information, see using Amazon S3 integration.

The backup files for RMAN backups remain on the Amazon RDS DB instance host until you remove them manually. You can use the `UTL_FILE.FREMOVE` Oracle procedure to remove files from a directory. For more information, see FREMOVE procedure in the Oracle Database documentation.

You can't use the RMAN to restore RDS for Oracle DB instances. However, you can use RMAN to restore a backup to an on-premises or Amazon EC2 instance. For more information, see the blog article Restore an Amazon RDS for Oracle instance to a self-managed instance.

> **Note**
>
> For backing up and restoring to another Amazon RDS for Oracle DB instance, you can continue to use the Amazon RDS backup and restore features. For more information, see Backing up, restoring, and exporting data.

Topics

- Prerequisites for RMAN backups
- Common parameters for RMAN procedures
- Validating database files in RDS for Oracle
- Enabling and disabling block change tracking
- Crosschecking archived redo logs
- Backing up archived redo log files
- Performing a full database backup
- Performing a full backup of a tenant database
- Performing an incremental database backup
- Performing an incremental backup of a tenant database
- Backing up a tablespace
- Backing up a control file
- Performing block media recovery

Prerequisites for RMAN backups

Before backing up your database using the `rdsadmin.rdsadmin_rman_util` package, make sure that you meet the following prerequisites:

- Make sure that your RDS for Oracle database is in `ARCHIVELOG` mode. To enable this mode, set the backup retention period to a non-zero value.
- When backing up archived redo logs or performing a full or incremental backup that includes archived redo logs, and when backing up the database, make sure that redo log retention is set to a nonzero value. Archived redo logs are required to make database files consistent during recovery. For more information, see Retaining archived redo logs.

- Make sure that your DB instance has sufficient free space to hold the backups. When back up your database, you specify an Oracle directory object as a parameter in the procedure call. RMAN places the files in the specified directory. You can use default directories, such as `DATA_PUMP_DIR`, or create a new directory. For more information, see Creating and dropping directories in the main data storage space.

  You can monitor the current free space in an RDS for Oracle instance using the CloudWatch metric `FreeStorageSpace`. We recommend that your free space exceeds the current size of the database, though RMAN backs up only formatted blocks and supports compression.

Common parameters for RMAN procedures

You can use procedures in the Amazon RDS package `rdsadmin.rdsadmin_rman_util` to perform tasks with RMAN. Several parameters are common to the procedures in the package. The package has the following common parameters.

Parameter name	Data type	Valid values	Defau	Requir	Description
`p_directory_name`	varchar	A valid database directory name.	—	Yes	The name of the directory to contain the backup files.
`p_label`	varchar	`a-z`, `A-Z`, `0-9`, `'_'`, `'-'`, `'.'`	—	No	A unique string that is included in the backup file names. **Note** The limit is 30 characters.
`p_owner`	varchar	A valid owner of the directory specified in	—	Yes	The owner of the directory to contain the backup files.

Parameter name	Data type	Valid values	Defau	Requir	Description
		p_directory_name .			

Parameter name	Data type	Valid values	Defau	Requir	Description
p_tag	varchar	a-z, A-Z, 0-9, '_', '-', '.'	NULL	No	A string that can be used to distinguish between backups to indicate the purpose or usage of backups, such as daily, weekly, or incremental-level backups. The limit is 30 characters. The tag is not case-sensitive. Tags are always stored in uppercase, regardless of the case used when entering them. Tags don't need to be unique, so multiple backups can have the same tag. If you don't specify a tag, then RMAN assigns a default tag automatically using the format TAG*YYYYMMDDT HHMMSS* , where *YYYY* is the year, *MM* is the month, *DD* is the day, *HH* is the hour (in 24-hour format), *MM* is the minutes, and *SS* is the seconds. The date and time refer to when RMAN started the backup. For example, a backup might receive a tag TAG20190927T214517 for a backup that started on 2019-09-27 at 21:45:17. The p_tag parameter is supported for the following Amazon RDS for Oracle DB engine versions: • Oracle Database 21c (21.0.0)

Parameter name	Data type	Valid values	Defau	Requir	Description
					• Oracle Database 19c (19.0.0), using 19.0.0.0.ru-2021-10.rur-2021-10.r1 or higher
p_compress	boolean	TRUE, FALSE	FALSE	No	Specify TRUE to enable BASIC backup compression. Specify FALSE to disable BASIC backup compression.
p_include_archive_logs	boolean	TRUE, FALSE	FALSE	No	Specify TRUE to include archived redo logs in the backup. Specify FALSE to exclude archived redo logs from the backup. If you include archived redo logs in the backup, set retention to one hour or greater using the rdsadmin.rdsadmin_util.set_configuration procedure. Also, call the rdsadmin.rdsadmin_rman_util.crosscheck_archivelog procedure immediately before running the backup. Otherwise, the backup might fail due to missing archived redo log files that have been deleted by Amazon RDS management procedures.
p_include_controlfile	boolean	TRUE, FALSE	FALSE	No	Specify TRUE to include the control file in the backup. Specify FALSE to exclude the control file from the backup.

Parameter name	Data type	Valid values	Defau	Requir	Description
p_optimize	boolear	TRUE, FALSE	TRUE	No	Specify TRUE to enable backup optimization, if archived redo logs are included, to reduce backup size. Specify FALSE to disable backup optimization.
p_parallel	numbe	A valid integer between 1 and 254 for Oracle Database Enterpris e Edition (EE) 1 for other Oracle Database editions	1	No	Number of channels.
p_rman_to _dbms_out put	boolear	TRUE, FALSE	FALSE	No	When TRUE, the RMAN output is sent to the DBMS_OUTPUT package in addition to a file in the BDUMP directory. In SQL*Plus, use SET SERVEROUTPUT ON to see the output. When FALSE, the RMAN output is only sent to a file in the BDUMP directory.

Parameter name	Data type	Valid values	Defau	Requir	Description
p_section_size_mb	numbe	A valid integer	NULL	No	The section size in megabytes (MB). Validates in parallel by dividing each file into the specified section size. When NULL, the parameter is ignored.
p_validation_type	varchar	'PHYSICAL' , 'PHYSICAL+LOGICAL'	'PHYS'	No	The level of corruption detection. Specify 'PHYSICAL' to check for physical corruption. An example of physical corruption is a block with a mismatch in the header and footer. Specify 'PHYSICAL+LOGICAL' to check for logical inconsistencies in addition to physical corruption. An example of logical corruption is a corrupt block.

Validating database files in RDS for Oracle

You can use the Amazon RDS package `rdsadmin.rdsadmin_rman_util` to validate Amazon RDS for Oracle database files, such as data files, tablespaces, control files, and server parameter files (SPFILEs).

For more information about RMAN validation, see Validating database files and backups and VALIDATE in the Oracle documentation.

Topics

- Validating a database
- Validating a tenant database
- Validating a tablespace
- Validating a control file

- Validating an SPFILE
- Validating an Oracle data file

Validating a database

To validate all of the relevant files used by an Oracle database in RDS for Oracle, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.validate_database`.

This procedure uses the following common parameters for RMAN tasks:

- `p_validation_type`
- `p_parallel`
- `p_section_size_mb`
- `p_rman_to_dbms_output`

For more information, see Common parameters for RMAN procedures.

The following example validates the database using the default values for the parameters.

```
EXEC rdsadmin.rdsadmin_rman_util.validate_database;
```

The following example validates the database using the specified values for the parameters.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.validate_database(
        p_validation_type     => 'PHYSICAL+LOGICAL',
        p_parallel            => 4,
        p_section_size_mb     => 10,
        p_rman_to_dbms_output => FALSE);
END;
/
```

When the `p_rman_to_dbms_output` parameter is set to `FALSE`, the RMAN output is written to a file in the `BDUMP` directory.

To view the files in the `BDUMP` directory, run the following `SELECT` statement.

```
SELECT * FROM table(rdsadmin.rds_file_util.listdir('BDUMP')) order by mtime;
```

To view the contents of a file in the BDUMP directory, run the following SELECT statement.

```
SELECT text FROM table(rdsadmin.rds_file_util.read_text_file('BDUMP','rds-rman-
validate-nnn.txt'));
```

Replace the file name with the name of the file you want to view.

Validating a tenant database

To validate the data files of the tenant database in a container database (CDB), use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.validate_tenant`.

This procedure applies only to the current tenant database and uses the following common parameters for RMAN tasks:

- `p_validation_type`
- `p_parallel`
- `p_section_size_mb`
- `p_rman_to_dbms_output`

For more information, see Common parameters for RMAN procedures. This procedure is supported for the following DB engine versions:

- Oracle Database 21c (21.0.0) CDB
- Oracle Database 19c (19.0.0) CDB

The following example validates the current tenant database using the default values for the parameters.

```
EXEC rdsadmin.rdsadmin_rman_util.validate_tenant;
```

The following example validates the current tenant database using the specified values for the parameters.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.validate_tenant(
        p_validation_type     => 'PHYSICAL+LOGICAL',
```

```
        p_parallel            => 4,
        p_section_size_mb     => 10,
        p_rman_to_dbms_output => FALSE);
END;
/
```

When the p_rman_to_dbms_output parameter is set to FALSE, the RMAN output is written to a file in the BDUMP directory.

To view the files in the BDUMP directory, run the following SELECT statement.

```
SELECT * FROM table(rdsadmin.rds_file_util.listdir('BDUMP')) order by mtime;
```

To view the contents of a file in the BDUMP directory, run the following SELECT statement.

```
SELECT text FROM table(rdsadmin.rds_file_util.read_text_file('BDUMP','rds-rman-
validate-nnn.txt'));
```

Replace the file name with the name of the file you want to view.

Validating a tablespace

To validate the files associated with a tablespace, use the Amazon RDS procedure rdsadmin.rdsadmin_rman_util.validate_tablespace.

This procedure uses the following common parameters for RMAN tasks:

- p_validation_type
- p_parallel
- p_section_size_mb
- p_rman_to_dbms_output

For more information, see Common parameters for RMAN procedures.

This procedure also uses the following additional parameter.

Parameter name	Data type	Valid values	Default	Required	Description
p_tablespace_name	varchar2	A valid tablespace name	—	Yes	The name of the tablespace.

Validating a control file

To validate only the control file used by an Amazon RDS Oracle DB instance, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.validate_current_controlfile`.

This procedure uses the following common parameter for RMAN tasks:

- `p_validation_type`
- `p_rman_to_dbms_output`

For more information, see Common parameters for RMAN procedures.

Validating an SPFILE

To validate only the server parameter file (SPFILE) used by an Amazon RDS Oracle DB instance, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.validate_spfile`.

This procedure uses the following common parameter for RMAN tasks:

- `p_validation_type`
- `p_rman_to_dbms_output`

For more information, see Common parameters for RMAN procedures.

Validating an Oracle data file

To validate a data file, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.validate_datafile`.

This procedure uses the following common parameters for RMAN tasks:

- `p_validation_type`

- p_parallel
- p_section_size_mb
- p_rman_to_dbms_output

For more information, see Common parameters for RMAN procedures.

This procedure also uses the following additional parameters.

Parameter name	Data type	Valid values	Default	Required	Description
p_datafile	varchar2	A valid datafile ID number or a valid datafile name including complete path	—	Yes	The datafile ID number (from v $datafile.file#) or the full datafile name including the path (from v $datafile.name).
p_from_block	number	A valid integer	NULL	No	The number of the block where the validation starts within the data file. When this is NULL, 1 is used.
p_to_block	number	A valid integer	NULL	No	The number of the block where the validation ends within the data file. When this is NULL, the maximum block in the data file is used.

Enabling and disabling block change tracking

Block changing tracking records changed blocks in a tracking file. This technique can improve the performance of RMAN incremental backups. For more information, see Using Block Change Tracking to Improve Incremental Backup Performance in the Oracle Database documentation.

RMAN features aren't supported in a read replica. However, as part of your high availability strategy, you might choose to enable block tracking in a read-only replica using the procedure `rdsadmin.rdsadmin_rman_util.enable_block_change_tracking`. If you promote this read-only replica to a source DB instance, block change tracking is enabled for the new source instance. Thus, your instance can benefit from fast incremental backups.

Block change tracking procedures are supported in Enterprise Edition only for the following DB engine versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

> **Note**
>
> In a single-tenant CDB, the following operations work, but no customer-visible mechanism can detect the current status of the operations. See also Limitations of RDS for Oracle CDBs.

To enable block change tracking for a DB instance, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.enable_block_change_tracking`. To disable block change tracking, use `disable_block_change_tracking`. These procedures take no parameters.

To determine whether block change tracking is enabled for your DB instance, run the following query.

```
SELECT STATUS, FILENAME FROM V$BLOCK_CHANGE_TRACKING;
```

The following example enables block change tracking for a DB instance.

```
EXEC rdsadmin.rdsadmin_rman_util.enable_block_change_tracking;
```

The following example disables block change tracking for a DB instance.

```
EXEC rdsadmin.rdsadmin_rman_util.disable_block_change_tracking;
```

Crosschecking archived redo logs

You can crosscheck archived redo logs using the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.crosscheck_archivelog`.

You can use this procedure to crosscheck the archived redo logs registered in the control file and optionally delete the expired logs records. When RMAN makes a backup, it creates a record in the control file. Over time, these records increase the size of the control file. We recommend that you remove expired records periodically.

> **Note**
>
> Standard Amazon RDS backups don't use RMAN and therefore don't create records in the control file.

This procedure uses the common parameter `p_rman_to_dbms_output` for RMAN tasks.

For more information, see Common parameters for RMAN procedures.

This procedure also uses the following additional parameter.

Parameter name	Data type	Valid values	Default	Required	Description
`p_delete_expired`	boolean	`TRUE`, `FALSE`	`TRUE`	No	When `TRUE`, delete expired archived redo log records from the control file. When `FALSE`, retain the expired archived redo log records in the control file.

This procedure is supported for the following Amazon RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

The following example marks archived redo log records in the control file as expired, but does not delete the records.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.crosscheck_archivelog(
        p_delete_expired      => FALSE,
        p_rman_to_dbms_output => FALSE);
END;
/
```

The following example deletes expired archived redo log records from the control file.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.crosscheck_archivelog(
        p_delete_expired      => TRUE,
        p_rman_to_dbms_output => FALSE);
END;
/
```

Backing up archived redo log files

You can use the Amazon RDS package `rdsadmin.rdsadmin_rman_util` to back up archived redo logs for an Amazon RDS Oracle DB instance.

The procedures for backing up archived redo logs are supported for the following Amazon RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

Topics

- Backing up all archived redo logs
- Backing up an archived redo log from a date range
- Backing up an archived redo log from an SCN range
- Backing up an archived redo log from a sequence number range

Backing up all archived redo logs

To back up all of the archived redo logs for an Amazon RDS Oracle DB instance, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_archivelog_all`.

This procedure uses the following common parameters for RMAN tasks:

- `p_owner`
- `p_directory_name`
- `p_label`
- `p_parallel`
- `p_compress`
- `p_rman_to_dbms_output`
- `p_tag`

For more information, see Common parameters for RMAN procedures.

The following example backs up all archived redo logs for the DB instance.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_archivelog_all(
        p_owner               => 'SYS',
        p_directory_name      => 'MYDIRECTORY',
        p_parallel            => 4,
        p_tag                 => 'MY_LOG_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Backing up an archived redo log from a date range

To back up specific archived redo logs for an Amazon RDS Oracle DB instance by specifying a date range, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_archivelog_date`. The date range specifies which archived redo logs to back up.

This procedure uses the following common parameters for RMAN tasks:

- p_owner
- p_directory_name
- p_label
- p_parallel
- p_compress
- p_rman_to_dbms_output
- p_tag

For more information, see Common parameters for RMAN procedures.

This procedure also uses the following additional parameters.

Parameter name	Data type	Valid values	Default	Required	Description
p_from_date	date	A date that is between the start_date and next_date of an archived redo log that exists on disk. The value must be less than or equal to the value specified	—	Yes	The starting date for the archived log backups.

Parameter name	Data type	Valid values	Default	Required	Description
		for p_to_date .			
p_to_date	date	A date that is between the start_date and next_date of an archived redo log that exists on disk. The value must be greater than or equal to the value specified for p_from_date .	—	Yes	The ending date for the archived log backups.

The following example backs up archived redo logs in the date range for the DB instance.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_archivelog_date(
        p_owner               => 'SYS',
```

```
        p_directory_name      => 'MYDIRECTORY',
        p_from_date           => '03/01/2019 00:00:00',
        p_to_date             => '03/02/2019 00:00:00',
        p_parallel            => 4,
        p_tag                 => 'MY_LOG_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Backing up an archived redo log from an SCN range

To back up specific archived redo logs for an Amazon RDS Oracle DB instance by specifying a system change number (SCN) range, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_archivelog_scn`. The SCN range specifies which archived redo logs to back up.

This procedure uses the following common parameters for RMAN tasks:

- `p_owner`
- `p_directory_name`
- `p_label`
- `p_parallel`
- `p_compress`
- `p_rman_to_dbms_output`
- `p_tag`

For more information, see Common parameters for RMAN procedures.

This procedure also uses the following additional parameters.

Parameter name	Data type	Valid values	Default	Required	Description
`p_from_scn`	number	An SCN of an archived redo	—	Yes	The starting SCN for the archived log backups.

Parameter name	Data type	Valid values	Default	Required	Description
		log that exists on disk. The value must be less than or equal to the value specified for `p_to_scn`.			
`p_to_scn`	number	An SCN of an archived redo log that exists on disk. The value must be greater than or equal to the value specified for `p_from_scn`.	—	Yes	The ending SCN for the archived log backups.

The following example backs up archived redo logs in the SCN range for the DB instance.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_archivelog_scn(
        p_owner                => 'SYS',
        p_directory_name       => 'MYDIRECTORY',
        p_from_scn             => 1533835,
        p_to_scn               => 1892447,
        p_parallel             => 4,
        p_tag                  => 'MY_LOG_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Backing up an archived redo log from a sequence number range

To back up specific archived redo logs for an Amazon RDS Oracle DB instance by specifying a sequence number range, use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_archivelog_sequence`. The sequence number range specifies which archived redo logs to back up.

This procedure uses the following common parameters for RMAN tasks:

- `p_owner`
- `p_directory_name`
- `p_label`
- `p_parallel`
- `p_compress`
- `p_rman_to_dbms_output`
- `p_tag`

For more information, see Common parameters for RMAN procedures.

This procedure also uses the following additional parameters.

Parameter name	Data type	Valid values	Default	Required	Description
`p_from_sequence`	number	A sequence	—	Yes	The starting sequence number

Parameter name	Data type	Valid values	Default	Required	Description
		number an archived redo log that exists on disk. The value must be less than or equal to the value specified for `p_to_sequence`.			for the archived log backups.

Parameter name	Data type	Valid values	Default	Required	Description
p_to_sequence	number	A sequence number of an archived redo log that exists on disk. The value must be greater than or equal to the value specified for p_from_sequence .	—	Yes	The ending sequence number for the archived log backups.

The following example backs up archived redo logs in the sequence number range for the DB instance.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_archivelog_sequence(
        p_owner               => 'SYS',
        p_directory_name      => 'MYDIRECTORY',
        p_from_sequence       => 11160,
        p_to_sequence         => 11160,
        p_parallel            => 4,
        p_tag                 => 'MY_LOG_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Performing a full database backup

You can perform a backup of all blocks of data files included in the backup using Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_database_full`.

This procedure uses the following common parameters for RMAN tasks:

- `p_owner`
- `p_directory_name`
- `p_label`
- `p_parallel`
- `p_section_size_mb`
- `p_include_archive_logs`
- `p_optimize`
- `p_compress`
- `p_rman_to_dbms_output`
- `p_tag`

For more information, see Common parameters for RMAN procedures.

This procedure is supported for the following Amazon RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

The following example performs a full backup of the DB instance using the specified values for the parameters.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_database_full(
        p_owner               => 'SYS',
        p_directory_name      => 'MYDIRECTORY',
        p_parallel            => 4,
        p_section_size_mb     => 10,
        p_tag                 => 'FULL_DB_BACKUP',
        p_rman_to_dbms_output => FALSE);
```

```
END;
/
```

Performing a full backup of a tenant database

You can perform a backup of all data blocks included a tenant database in a container database (CDB). Use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_tenant_full`. This procedure applies only to the current database backup and uses the following common parameters for RMAN tasks:

- `p_owner`
- `p_directory_name`
- `p_label`
- `p_parallel`
- `p_section_size_mb`
- `p_include_archive_logs`
- `p_optimize`
- `p_compress`
- `p_rman_to_dbms_output`
- `p_tag`

For more information, see Common parameters for RMAN procedures.

The `rdsadmin_rman_util.backup_tenant_full` procedure is supported for the following RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0) CDB
- Oracle Database 19c (19.0.0) CDB

The following example performs a full backup of the current tenant database using the specified values for the parameters.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_tenant_full(
        p_owner               => 'SYS',
```

```
        p_directory_name      => 'MYDIRECTORY',
        p_parallel            => 4,
        p_section_size_mb     => 10,
        p_tag                 => 'FULL_TENANT_DB_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Performing an incremental database backup

You can perform an incremental backup of your DB instance using the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_database_incremental`.

For more information about incremental backups, see Incremental backups in the Oracle documentation.

This procedure uses the following common parameters for RMAN tasks:

- `p_owner`
- `p_directory_name`
- `p_label`
- `p_parallel`
- `p_section_size_mb`
- `p_include_archive_logs`
- `p_include_controlfile`
- `p_optimize`
- `p_compress`
- `p_rman_to_dbms_output`
- `p_tag`

For more information, see Common parameters for RMAN procedures.

This procedure is supported for the following Amazon RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

This procedure also uses the following additional parameter.

Parameter name	Data type	Valid values	Default	Required	Description
p_level	number	0, 1	0	No	Specify 0 to enable a full incremental backup. Specify 1 to enable a non-cumulative incremental backup.

The following example performs an incremental backup of the DB instance using the specified values for the parameters.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_database_incremental(
        p_owner                => 'SYS',
        p_directory_name       => 'MYDIRECTORY',
        p_level                => 1,
        p_parallel             => 4,
        p_section_size_mb      => 10,
        p_tag                  => 'MY_INCREMENTAL_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Performing an incremental backup of a tenant database

You can perform an incremental backup of the current tenant database in your CDB. Use the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_tenant_incremental`.

For more information about incremental backups, see Incremental backups in the Oracle Database documentation.

This procedure applies only to the current tenant database and uses the following common parameters for RMAN tasks:

- p_owner

- p_directory_name
- p_label
- p_parallel
- p_section_size_mb
- p_include_archive_logs
- p_include_controlfile
- p_optimize
- p_compress
- p_rman_to_dbms_output
- p_tag

For more information, see Common parameters for RMAN procedures.

This procedure is supported for the following Amazon RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0) CDB
- Oracle Database 19c (19.0.0) CDB

This procedure also uses the following additional parameter.

Parameter name	Data type	Valid values	Default	Required	Description
p_level	number	0, 1	0	No	Specify 0 to enable a full incremental backup. Specify 1 to enable a non-cumulative incremental backup.

The following example performs an incremental backup of the current tenant database using the specified values for the parameters.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_tenant_incremental(
        p_owner                => 'SYS',
        p_directory_name       => 'MYDIRECTORY',
        p_level                => 1,
        p_parallel             => 4,
        p_section_size_mb      => 10,
        p_tag                  => 'MY_INCREMENTAL_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Backing up a tablespace

You can back up a tablespace using the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_tablespace`.

This procedure uses the following common parameters for RMAN tasks:

- `p_owner`
- `p_directory_name`
- `p_label`
- `p_parallel`
- `p_section_size_mb`
- `p_include_archive_logs`
- `p_include_controlfile`
- `p_optimize`
- `p_compress`
- `p_rman_to_dbms_output`
- `p_tag`

For more information, see Common parameters for RMAN procedures.

This procedure also uses the following additional parameter.

Parameter name	Data type	Valid values	Default	Required	Description
p_tablespace_name	varchar2	A valid tablespace name.	—	Yes	The name of the tablespace to back up.

This procedure is supported for the following Amazon RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

The following example performs a tablespace backup using the specified values for the parameters.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_tablespace(
        p_owner               => 'SYS',
        p_directory_name      => 'MYDIRECTORY',
        p_tablespace_name     => 'MYTABLESPACE',
        p_parallel            => 4,
        p_section_size_mb     => 10,
        p_tag                 => 'MYTABLESPACE_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Backing up a control file

You can back up a control file using the Amazon RDS procedure `rdsadmin.rdsadmin_rman_util.backup_current_controlfile`.

This procedure uses the following common parameters for RMAN tasks:

- `p_owner`
- `p_directory_name`
- `p_label`

- p_compress
- p_rman_to_dbms_output
- p_tag

For more information, see Common parameters for RMAN procedures.

This procedure is supported for the following Amazon RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

The following example backs up a control file using the specified values for the parameters.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.backup_current_controlfile(
        p_owner               => 'SYS',
        p_directory_name      => 'MYDIRECTORY',
        p_tag                 => 'CONTROL_FILE_BACKUP',
        p_rman_to_dbms_output => FALSE);
END;
/
```

Performing block media recovery

You can recovery individual data blocks, known as *block media recovery*, using the Amazon RDS procedures rdsadmin.rdsadmin_rman_util.recover_datafile_block. You can use this overloaded procedure to recover either an individual data block or a range of data blocks.

This procedure uses the following common parameter for RMAN tasks:

- p_rman_to_dbms_output

For more information, see Common parameters for RMAN procedures.

This procedure uses the following additional parameters.

Parameter name	Data type	Valid values	Default	Required	Description
p_datafile	NUMBER	A valid data file ID number.	—	Yes	The data file containing the corrupt blocks. Specify the data file in either of the following ways: • The data file ID number, which is located in V$DATAFILE.FILE# • The full data file name, including the path, located in V$DATAFILE.NAME
p_block	NUMBER	A valid integer.	—	Yes	The number of an individual block to be recovered. The following parameters are mutually exclusive: • p_block • p_from_block and p_to_block
p_from_block	NUMBER	A valid integer.	—	Yes	The first block number in a range of blocks to be recovered. The following parameters are mutually exclusive: • p_block

Parameter name	Data type	Valid values	Default	Required	Description
					• p_from_block and p_to_block
p_to_block	NUMBER	A valid integer.	—	Yes	The last block number in a range of blocks to be recovered. The following parameters are mutually exclusive: • p_block • p_from_block and p_to_block

This procedure is supported for the following Amazon RDS for Oracle DB engine versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

The following example recovers block 100 in data file 5.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.recover_datafile_block(
        p_datafile            => 5,
        p_block               => 100,
        p_rman_to_dbms_output => TRUE);
END;
/
```

The following example recovers blocks 100 to 150 in data file 5.

```
BEGIN
    rdsadmin.rdsadmin_rman_util.recover_datafile_block(
        p_datafile            => 5,
        p_from_block          => 100,
        p_to_block            => 150,
```

```
        p_rman_to_dbms_output => TRUE);
END;
/
```

Performing common scheduling tasks for Oracle DB instances

Some scheduler jobs owned by SYS can interfere with normal database operations. In such cases, Oracle Support recommends that you modify the schedule. If you need to enable or disable SYS jobs, test the operation on scheduled jobs in a test environment before implementing it in a production environment. To perform tasks for Oracle Scheduler jobs owned by SYS, use the Amazon RDS package `rdsadmin.rdsadmin_dbms_scheduler`.

The `rdsadmin.rdsadmin_dbms_scheduler` procedures are supported for the Amazon RDS for Oracle DB engine versions shown in the following table. When using this package, you can specify the SYS jobs listed in the table.

Database release	Jobs enabled by default	Jobs disabled by default
Oracle Database 19c	BSLN_MAINTAIN_STATS_JOB CLEANUP_NON_EXIST_OBJ CLEANUP_ONLINE_IND_BUILD CLEANUP_ONLINE_PMO CLEANUP_TAB_IOT_PMO CLEANUP_TRANSIENT_PKG CLEANUP_TRANSIENT_TYPE DRA_REEVALUATE_OPEN_FAILU RES FILE_SIZE_UPD ORA$AUTOTASK_CLEAN PMO_DEFERRED_GIDX_MAINT_JO B PURGE_LOG RSE$CLEAN_RECOVERABLE_SC RIPT SM$CLEAN_AUTO_SPLIT_MERGE	FGR$AUTOPURGE_JOB FILE_WATCHER HM_CREATE_OFFLINE_DICTIONARY LOAD_OPATCH_INVENTORY ORA$PREPLUGIN_BACKUP_JOB XMLDB_NFS_CLEANUP_JOB
Oracle Database 21c	BSLN_MAINTAIN_STATS_JOB CLEANUP_NON_EXIST_OBJ CLEANUP_ONLINE_IND_BUILD	FGR$AUTOPURGE_JOB FILE_WATCHER HM_CREATE_OFFLINE_DICTIONARY

Database release	Jobs enabled by default	Jobs disabled by default
	CLEANUP_ONLINE_PMO CLEANUP_TAB_IOT_PMO CLEANUP_TRANSIENT_PKG CLEANUP_TRANSIENT_TYPE DRA_REEVALUATE_OPEN_FAILURES FILE_SIZE_UPD ORA$AUTOTASK_CLEAN PMO_DEFERRED_GIDX_MAINT_JOB PURGE_LOG	LOAD_OPATCH_INVENTORY ORA$PREPLUGIN_BACKUP_JOB ORA$_ATSK_AUTOSTS XMLDB_NFS_CLEANUP_JOB

Common parameters for Oracle Scheduler procedures

To perform tasks with Oracle Scheduler, use procedures in the Amazon RDS package `rdsadmin.rdsadmin_dbms_scheduler`. Several parameters are common to the procedures in the package. The package has the following common parameters.

Parameter name	Data type	Valid values	Default	Required	Description
name	varchar2	The procedures listed in the table in Performing common scheduling tasks for Oracle DB instances	—	Yes	The name of the job to modify.

Parameter name	Data type	Valid values	Default	Required	Description
attribute	varchar2	'REPEAT_INTERVAL' _NAME'	–	Yes	Attribute to modify. To modify the repeat interval for the job, specify 'REPEAT_INTERVAL' . To modify the schedule name for the job, specify 'SCHEDULE_NAME' .
value	varchar2	A valid schedule interval or schedule name, depending on attribute used.	–	Yes	The new value of the attribute.

Modifying DBMS_SCHEDULER jobs

To modify certain components of Oracle Scheduler, use the Oracle procedure dbms_scheduler.set_attribute. For more information, see DBMS_SCHEDULER and SET_ATTRIBUTE procedure in the Oracle documentation.

When working with Amazon RDS DB instances, prepend the schema name SYS to the object name. The following example sets the resource plan attribute for the Monday window object.

```
BEGIN
    DBMS_SCHEDULER.SET_ATTRIBUTE(
        name      => 'SYS.MONDAY_WINDOW',
```

```
        attribute => 'RESOURCE_PLAN',
        value     => 'resource_plan_1');
END;
/
```

Modifying AutoTask maintenance windows

Amazon RDS for Oracle instances are created with default settings for maintenance windows. Automated maintenance tasks such as optimizer statistics collection run during these windows. By default, the maintenance windows turn on Oracle Database Resource Manager.

To modify the window, use the `DBMS_SCHEDULER` package. You might need to modify the maintenance window settings for the following reasons:

- You want maintenance jobs to run at a different time, with different settings, or not at all. For example, might want to modify the window duration, or change the repeat time and interval.
- You want to avoid the performance impact of enabling Resource Manager during maintenance. For example, if the default maintenance plan is specified, and if the maintenance window opens while the database is under load, you might see wait events such as `resmgr:cpu quantum`. This wait event is related to Database Resource Manager. You have the following options:
  - Ensure that maintenance windows are active during off-peak times for your DB instance.
  - Disable the default maintenance plan by setting the `resource_plan` attribute to an empty string.
  - Set the `resource_manager_plan` parameter to `FORCE:` in your parameter group. If your instance uses Enterprise Edition, this setting prevents Database Resource Manager plans from activating.

To modify your maintenance window settings

1. Connect to your database using an Oracle SQL client.
2. Query the current configuration for a scheduler window.

   The following example queries the configuration for `MONDAY_WINDOW`.

   ```
   SELECT ENABLED, RESOURCE_PLAN, DURATION, REPEAT_INTERVAL
   FROM   DBA_SCHEDULER_WINDOWS
   WHERE  WINDOW_NAME='MONDAY_WINDOW';
   ```

The following output shows that the window is using the default values.

```
ENABLED         RESOURCE_PLAN                  DURATION         REPEAT_INTERVAL
--------------- ------------------------------ ----------------
 ------------------------------
TRUE            DEFAULT_MAINTENANCE_PLAN       +000 04:00:00
 freq=daily;byday=MON;byhour=22
                                                                 ;byminute=0;
 bysecond=0
```

3. Modify the window using the DBMS_SCHEDULER package.

   The following example sets the resource plan to null so that the Resource Manager won't run during the maintenance window.

```
BEGIN
  -- disable the window to make changes
  DBMS_SCHEDULER.DISABLE(name=>'"SYS"."MONDAY_WINDOW"',force=>TRUE);

  -- specify the empty string to use no plan
  DBMS_SCHEDULER.SET_ATTRIBUTE(name=>'"SYS"."MONDAY_WINDOW"',
 attribute=>'RESOURCE_PLAN', value=>'');

  -- re-enable the window
  DBMS_SCHEDULER.ENABLE(name=>'"SYS"."MONDAY_WINDOW"');
END;
/
```

   The following example sets the maximum duration of the window to 2 hours.

```
BEGIN
  DBMS_SCHEDULER.DISABLE(name=>'"SYS"."MONDAY_WINDOW"',force=>TRUE);
  DBMS_SCHEDULER.SET_ATTRIBUTE(name=>'"SYS"."MONDAY_WINDOW"',
 attribute=>'DURATION', value=>'0 2:00:00');
  DBMS_SCHEDULER.ENABLE(name=>'"SYS"."MONDAY_WINDOW"');
END;
/
```

   The following example sets the repeat interval to every Monday at 10 AM.

```
BEGIN
```

```
  DBMS_SCHEDULER.DISABLE(name=>'"SYS"."MONDAY_WINDOW"',force=>TRUE);
  DBMS_SCHEDULER.SET_ATTRIBUTE(name=>'"SYS"."MONDAY_WINDOW"',
 attribute=>'REPEAT_INTERVAL',
 value=>'freq=daily;byday=MON;byhour=10;byminute=0;bysecond=0');
  DBMS_SCHEDULER.ENABLE(name=>'"SYS"."MONDAY_WINDOW"');
END;
/
```

Setting the time zone for Oracle Scheduler jobs

To modify the time zone for Oracle Scheduler, you can use the Oracle procedure `dbms_scheduler.set_scheduler_attribute`. For more information about the `dbms_scheduler` package, see DBMS_SCHEDULER and SET_SCHEDULER_ATTRIBUTE in the Oracle documentation.

To modify the current time zone setting

1. Connect to the database using a client such as SQL Developer. For more information, see Connecting to your DB instance using Oracle SQL developer.
2. Set the default time zone as following, substituting your time zone for *`time_zone_name`*.

```
BEGIN
  DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE(
    attribute => 'default_timezone',
    value => 'time_zone_name'
  );
END;
/
```

In the following example, you change the time zone to Asia/Shanghai.

Start by querying the current time zone, as shown following.

```
SELECT VALUE FROM DBA_SCHEDULER_GLOBAL_ATTRIBUTE WHERE
 ATTRIBUTE_NAME='DEFAULT_TIMEZONE';
```

The output shows that the current time zone is ETC/UTC.

```
VALUE
```

```
-------
Etc/UTC
```

Then you set the time zone to Asia/Shanghai.

```
BEGIN
  DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE(
    attribute => 'default_timezone',
    value => 'Asia/Shanghai'
  );
END;
/
```

For more information about changing the system time zone, see Oracle time zone.

Turning off Oracle Scheduler jobs owned by SYS

To disable an Oracle Scheduler job owned by the SYS user, use the `rdsadmin.rdsadmin_dbms_scheduler.disable` procedure.

This procedure uses the `name` common parameter for Oracle Scheduler tasks. For more information, see Common parameters for Oracle Scheduler procedures.

The following example disables the `SYS.CLEANUP_ONLINE_IND_BUILD` Oracle Scheduler job.

```
BEGIN
   rdsadmin.rdsadmin_dbms_scheduler.disable('SYS.CLEANUP_ONLINE_IND_BUILD');
END;
/
```

Turning on Oracle Scheduler jobs owned by SYS

To turn on an Oracle Scheduler job owned by SYS, use the `rdsadmin.rdsadmin_dbms_scheduler.enable` procedure.

This procedure uses the `name` common parameter for Oracle Scheduler tasks. For more information, see Common parameters for Oracle Scheduler procedures.

The following example enables the `SYS.CLEANUP_ONLINE_IND_BUILD` Oracle Scheduler job.

```
BEGIN
   rdsadmin.rdsadmin_dbms_scheduler.enable('SYS.CLEANUP_ONLINE_IND_BUILD');
```

```
END;
/
```

Modifying the Oracle Scheduler repeat interval for jobs of CALENDAR type

To modify the repeat interval to modify a SYS-owned Oracle Scheduler job of `CALENDAR` type, use the `rdsadmin.rdsadmin_dbms_scheduler.disable` procedure.

This procedure uses the following common parameters for Oracle Scheduler tasks:

- `name`
- `attribute`
- `value`

For more information, see Common parameters for Oracle Scheduler procedures.

The following example modifies the repeat interval of the `SYS.CLEANUP_ONLINE_IND_BUILD` Oracle Scheduler job.

```
BEGIN
     rdsadmin.rdsadmin_dbms_scheduler.set_attribute(
          name      => 'SYS.CLEANUP_ONLINE_IND_BUILD',
          attribute => 'repeat_interval',
          value     => 'freq=daily;byday=FRI,SAT;byhour=20;byminute=0;bysecond=0');
END;
/
```

Modifying the Oracle Scheduler repeat interval for jobs of NAMED type

Some Oracle Scheduler jobs use a schedule name instead of an interval. For this type of jobs, you must create a new named schedule in the master user schema. Use the standard Oracle `sys.dbms_scheduler.create_schedule` procedure to do this. Also, use the `rdsadmin.rdsadmin_dbms_scheduler.set_attribute procedure` to assign the new named schedule to the job.

This procedure uses the following common parameter for Oracle Scheduler tasks:

- `name`
- `attribute`

- `value`

For more information, see Common parameters for Oracle Scheduler procedures.

The following example modifies the repeat interval of the `SYS.BSLN_MAINTAIN_STATS_JOB` Oracle Scheduler job.

```
BEGIN
     DBMS_SCHEDULER.CREATE_SCHEDULE (
          schedule_name   => 'rds_master_user.new_schedule',
          start_date      => SYSTIMESTAMP,
          repeat_interval =>
 'freq=daily;byday=MON,TUE,WED,THU,FRI;byhour=0;byminute=0;bysecond=0',
          end_date        => NULL,
          comments        => 'Repeats daily forever');
END;
/

BEGIN
     rdsadmin.rdsadmin_dbms_scheduler.set_attribute (
          name      => 'SYS.BSLN_MAINTAIN_STATS_JOB',
          attribute => 'schedule_name',
          value     => 'rds_master_user.new_schedule');
END;
/
```

Turning off autocommit for Oracle Scheduler job creation

When `DBMS_SCHEDULER.CREATE_JOB` creates Oracle Scheduler jobs, it creates the jobs immediately and commits the changes. You might need to incorporate the creation of Oracle Scheduler jobs in the user transaction to do the following:

- Roll back the Oracle Schedule job when the user transaction is rolled back.
- Create the Oracle Scheduler job when the main user transaction is committed.

You can use the procedure `rdsadmin.rdsadmin_dbms_scheduler.set_no_commit_flag` to turn on this behavior. This procedure takes no parameters. You can use this procedure in the following RDS for Oracle releases:

- 21.0.0.0.ru-2022-07.rur-2022-07.r1 and higher

- 19.0.0.0.ru-2022-07.rur-2022-07.r1 and higher

The following example turns off autocommit for Oracle Scheduler, creates an Oracle Scheduler job, and then rolls back the transaction. Because autocommit is turned off, the database also rolls back the creation of the Oracle Scheduler job.

```
BEGIN
  rdsadmin.rdsadmin_dbms_scheduler.set_no_commit_flag;
  DBMS_SCHEDULER.CREATE_JOB(job_name   => 'EMPTY_JOB',
                            job_type   => 'PLSQL_BLOCK',
                            job_action => 'begin null; end;',
                            auto_drop  => false);
  ROLLBACK;
END;
/

PL/SQL procedure successfully completed.

SELECT * FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME='EMPTY_JOB';

no rows selected
```

Diagnosing problems with RDS for Oracle DB instances

Oracle Database includes a fault diagnosability infrastructure that you can use to investigate database problems. In Oracle terminology, a *problem* is a critical error such as a code bug or data corruption. An *incident* is the occurrence of a problem. If the same error occurs three times, then the infrastructure shows three incidents of this problem. For more information, see Diagnosing and resolving problems in the Oracle Database documentation.

The Automatic Diagnostic Repository Command Interpreter (ADRCI) utility is an Oracle command-line tool that you use to manage diagnostic data. For example, you can use this tool to investigate problems and package diagnostic data. An *incident package* includes diagnostic data for an incident or all incidents that reference a specific problem. You can upload an incident package, which is implemented as a .zip file, to Oracle Support.

To deliver a managed service experience, Amazon RDS doesn't provide shell access to ADRCI. To perform diagnostic tasks for your RDS for Oracle DB instance, use the Amazon RDS package `rdsadmin.rdsadmin_adrci_util`.

By using the functions in `rdsadmin_adrci_util`, you can list and package problems and incidents, and also show trace files. All functions return a task ID. This ID forms part of the name of log file that contains the ADRCI output, as in `dbtask-`*`task_id`*`.log`. The log file resides in the BDUMP directory. You can download the log file by following the procedure described in Downloading a database log file.

Common parameters for diagnostic procedures

To perform diagnostic tasks, use functions in the Amazon RDS package `rdsadmin.rdsadmin_adrci_util`. The package has the following common parameters.

Parameter name	Data type	Valid values	Default	Required	Description
`incident_id`	number	A valid incident ID or null	Null	No	If the value is null, the function shows all incidents. If the value isn't null and represents a valid incident ID, the function shows the specified incident.
`problem_id`	number	A valid problem ID or null	Null	No	If the value is null, the function shows all problems. If the value isn't null and represents a valid problem ID, the function shows the specified problem.
`last`	number	A valid integer greater than 0 or null	Null	No	If the value is null, then the function displays at most 50 items. If the value isn't null, the function

Parameter name	Data type	Valid values	Default	Required	Description
					displays the specified number.

Listing incidents

To list diagnostic incidents for Oracle, use the Amazon RDS function `rdsadmin.rdsadmin_adrci_util.list_adrci_incidents`. You can list incidents in either basic or detailed mode. By default, the function lists the 50 most recent incidents.

This function uses the following common parameters:

- `incident_id`
- `problem_id`
- `last`

If you specify `incident_id` and `problem_id`, then `incident_id` overrides `problem_id`. For more information, see Common parameters for diagnostic procedures.

This function uses the following additional parameter.

Parameter name	Data type	Valid values	Default	Required	Description
`detail`	boolean	TRUE or FALSE	FALSE	No	If TRUE, the function lists incidents in detail mode. If FALSE, the function lists incidents in basic mode.

To list all incidents, query the `rdsadmin.rdsadmin_adrci_util.list_adrci_incidents` function without any arguments. The query returns the task ID.

```
SQL> SELECT rdsadmin.rdsadmin_adrci_util.list_adrci_incidents AS task_id FROM DUAL;
```

```
TASK_ID
------------------
1590786706158-3126
```

Or call the `rdsadmin.rdsadmin_adrci_util.list_adrci_incidents` function without any arguments and store the output in a SQL client variable. You can use the variable in other statements.

```
SQL> VAR task_id VARCHAR2(80);
SQL> EXEC :task_id := rdsadmin.rdsadmin_adrci_util.list_adrci_incidents;

PL/SQL procedure successfully completed.
```

To read the log file, call the Amazon RDS procedure `rdsadmin.rds_file_util.read_text_file`. Supply the task ID as part of the file name. The following output shows three incidents: 53523, 53522, and 53521.

```
SQL> SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||:task_id||'.log'));

TEXT
--------------------------------------------------------------------------------
2020-05-29 21:11:46.193 UTC [INFO ] Listing ADRCI incidents.
2020-05-29 21:11:46.256 UTC [INFO ]
ADR Home = /rdsdbdata/log/diag/rdbms/orcl_a/ORCL:
*************************************************************************
INCIDENT_ID PROBLEM_KEY                                                 CREATE_TIME
----------- ------------------------------------------------------------
 ----------------------------------------
53523       ORA 700 [EVENT_CREATED_INCIDENT] [942] [SIMULATED_ERROR_003 2020-05-29
 20:15:20.928000 +00:00
53522       ORA 700 [EVENT_CREATED_INCIDENT] [942] [SIMULATED_ERROR_002 2020-05-29
 20:15:15.247000 +00:00
53521       ORA 700 [EVENT_CREATED_INCIDENT] [942] [SIMULATED_ERROR_001 2020-05-29
 20:15:06.047000 +00:00
3 rows fetched


2020-05-29 21:11:46.256 UTC [INFO ] The ADRCI incidents were successfully listed.
2020-05-29 21:11:46.256 UTC [INFO ] The task finished successfully.
```

```
14 rows selected.
```

To list a particular incident, specify its ID using the `incident_id` parameter. In the following example, you query the log file for incident 53523 only.

```
SQL> EXEC :task_id :=
 rdsadmin.rdsadmin_adrci_util.list_adrci_incidents(incident_id=>53523);

PL/SQL procedure successfully completed.

SQL> SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||:task_id||'.log'));

TEXT
--------------------------------------------------------------------------------
2020-05-29 21:15:25.358 UTC [INFO ] Listing ADRCI incidents.
2020-05-29 21:15:25.426 UTC [INFO ]
ADR Home = /rdsdbdata/log/diag/rdbms/orcl_a/ORCL:
*********************************************************************
INCIDENT_ID          PROBLEM_KEY
 CREATE_TIME
-------------------- ----------------------------------------------------------
 ---------------------------------
53523                ORA 700 [EVENT_CREATED_INCIDENT] [942] [SIMULATED_ERROR_003
 2020-05-29 20:15:20.928000 +00:00
1 rows fetched


2020-05-29 21:15:25.427 UTC [INFO ] The ADRCI incidents were successfully listed.
2020-05-29 21:15:25.427 UTC [INFO ] The task finished successfully.

12 rows selected.
```

Listing problems

To list diagnostic problems for Oracle, use the Amazon RDS function `rdsadmin.rdsadmin_adrci_util.list_adrci_problems`.

By default, the function lists the 50 most recent problems.

This function uses the common parameters `problem_id` and `last`. For more information, see Common parameters for diagnostic procedures.

To get the task ID for all problems, call the `rdsadmin.rdsadmin_adrci_util.list_adrci_problems` function without any arguments, and store the output in a SQL client variable.

```
SQL> EXEC :task_id := rdsadmin.rdsadmin_adrci_util.list_adrci_problems;

PL/SQL procedure successfully completed.
```

To read the log file, call the `rdsadmin.rds_file_util.read_text_file` function, supplying the task ID as part of the file name. In the following output, the log file shows three problems: 1, 2, and 3.

```
SQL> SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||:task_id||'.log'));

TEXT
--------------------------------------------------------------------------------
2020-05-29 21:18:50.764 UTC [INFO ] Listing ADRCI problems.
2020-05-29 21:18:50.829 UTC [INFO ]
ADR Home = /rdsdbdata/log/diag/rdbms/orcl_a/ORCL:
*************************************************************************
PROBLEM_ID   PROBLEM_KEY                                                 LAST_INCIDENT
       LASTINC_TIME
---------- ------------------------------------------------------------ -------------
 -------------------------------
2          ORA 700 [EVENT_CREATED_INCIDENT] [942] [SIMULATED_ERROR_003 53523
 2020-05-29 20:15:20.928000 +00:00
3          ORA 700 [EVENT_CREATED_INCIDENT] [942] [SIMULATED_ERROR_002 53522
 2020-05-29 20:15:15.247000 +00:00
1          ORA 700 [EVENT_CREATED_INCIDENT] [942] [SIMULATED_ERROR_001 53521
 2020-05-29 20:15:06.047000 +00:00
3 rows fetched


2020-05-29 21:18:50.829 UTC [INFO ] The ADRCI problems were successfully listed.
2020-05-29 21:18:50.829 UTC [INFO ] The task finished successfully.

14 rows selected.
```

In the following example, you list problem 3 only.

```
SQL> EXEC :task_id := rdsadmin.rdsadmin_adrci_util.list_adrci_problems(problem_id=>3);
```

```
PL/SQL procedure successfully completed.
```

To read the log file for problem 3, call `rdsadmin.rds_file_util.read_text_file`. Supply the task ID as part of the file name.

```
SQL> SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||:task_id||'.log'));

TEXT
--------------------------------------------------------------------------------
2020-05-29 21:19:42.533 UTC [INFO ] Listing ADRCI problems.
2020-05-29 21:19:42.599 UTC [INFO ]
ADR Home = /rdsdbdata/log/diag/rdbms/orcl_a/ORCL:
*********************************************************************
PROBLEM_ID PROBLEM_KEY                                                 LAST_INCIDENT
 LASTINC_TIME
---------- ------------------------------------------------------------ -------------
 -------------------------------
3          ORA 700 [EVENT_CREATED_INCIDENT] [942] [SIMULATED_ERROR_002 53522
 2020-05-29 20:15:15.247000 +00:00
1 rows fetched


2020-05-29 21:19:42.599 UTC [INFO ] The ADRCI problems were successfully listed.
2020-05-29 21:19:42.599 UTC [INFO ] The task finished successfully.

12 rows selected.
```

Creating incident packages

You can create incident packages using the Amazon RDS function `rdsadmin.rdsadmin_adrci_util.create_adrci_package`. The output is a .zip file that you can supply to Oracle Support.

This function uses the following common parameters:

- `problem_id`
- `incident_id`

Make sure to specify one of the preceding parameters. If you specify both parameters, `incident_id` overrides `problem_id`. For more information, see Common parameters for diagnostic procedures.

To create a package for a specific incident, call the Amazon RDS function `rdsadmin.rdsadmin_adrci_util.create_adrci_package` with the `incident_id` parameter. The following example creates a package for incident 53523.

```
SQL> EXEC :task_id :=
 rdsadmin.rdsadmin_adrci_util.create_adrci_package(incident_id=>53523);

PL/SQL procedure successfully completed.
```

To read the log file, call the `rdsadmin.rds_file_util.read_text_file`. You can supply the task ID as part of the file name. The output shows that you generated incident package `ORA700EVE_20200529212043_COM_1.zip`.

```
SQL> SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||:task_id||'.log'));

TEXT
--------------------------------------------------------------------------------
2020-05-29 21:20:43.031 UTC [INFO ] The ADRCI package is being created.
2020-05-29 21:20:47.641 UTC [INFO ] Generated package 1 in file /rdsdbdata/log/trace/
ORA700EVE_20200529212043_COM_1.zip, mode complete
2020-05-29 21:20:47.642 UTC [INFO ] The ADRCI package was successfully created.
2020-05-29 21:20:47.642 UTC [INFO ] The task finished successfully.
```

To package diagnostic data for a particular problem, specify its ID using the `problem_id` parameter. In the following example, you package data for problem 3 only.

```
SQL> EXEC :task_id := rdsadmin.rdsadmin_adrci_util.create_adrci_package(problem_id=>3);

PL/SQL procedure successfully completed.
```

To read the task output, call `rdsadmin.rds_file_util.read_text_file`, supplying the task ID as part of the file name. The output shows that you generated incident package `ORA700EVE_20200529212111_COM_1.zip`.

```
SQL> SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||:task_id||'.log'));

TEXT
--------------------------------------------------------------------------------
2020-05-29 21:21:11.050 UTC [INFO ] The ADRCI package is being created.
2020-05-29 21:21:15.646 UTC [INFO ] Generated package 2 in file /rdsdbdata/log/trace/
ORA700EVE_20200529212111_COM_1.zip, mode complete
2020-05-29 21:21:15.646 UTC [INFO ] The ADRCI package was successfully created.
2020-05-29 21:21:15.646 UTC [INFO ] The task finished successfully.
```

You can also download the log file. For more information, see Downloading a database log file.

Showing trace files

You can use the Amazon RDS function `rdsadmin.rdsadmin_adrci_util.show_adrci_tracefile` to list trace files under the trace directory and all incident directories under the current ADR home. You can also show the contents of trace files and incident trace files.

This function uses the following parameter.

Parameter name	Data type	Valid values	Default	Required	Description
`filename`	varchar2	A valid trace file name	Null	No	If the value is null, the function shows all trace files. If it isn't null, the function shows the specified file.

To show the trace file, call the Amazon RDS function `rdsadmin.rdsadmin_adrci_util.show_adrci_tracefile`.

```
SQL> EXEC :task_id := rdsadmin.rdsadmin_adrci_util.show_adrci_tracefile;

PL/SQL procedure successfully completed.
```

To list the trace file names, call the Amazon RDS procedure `rdsadmin.rds_file_util.read_text_file`, supplying the task ID as part of the file name.

```
SQL> SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||:task_id||'.log')) WHERE TEXT LIKE '%/alert_%';

TEXT
--------------------------------------------------------------
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log.2020-05-28
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log.2020-05-27
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log.2020-05-26
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log.2020-05-25
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log.2020-05-24
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log.2020-05-23
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log.2020-05-22
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log.2020-05-21
     diag/rdbms/orcl_a/ORCL/trace/alert_ORCL.log

9 rows selected.
```

In the following example, you generate output for `alert_ORCL.log`.

```
SQL> EXEC :task_id := rdsadmin.rdsadmin_adrci_util.show_adrci_tracefile('diag/rdbms/
orcl_a/ORCL/trace/alert_ORCL.log');

PL/SQL procedure successfully completed.
```

To read the log file, call `rdsadmin.rds_file_util.read_text_file`. Supply the task ID as part of the file name. The output shows the first 10 lines of alert_ORCL.log.

```
SQL> SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||:task_id||'.log')) WHERE ROWNUM <= 10;

TEXT
--------------------------------------------------------------------------------
2020-05-29 21:24:02.083 UTC [INFO ] The trace files are being displayed.
2020-05-29 21:24:02.128 UTC [INFO ] Thu May 28 23:59:10 2020
Thread 1 advanced to log sequence 2048 (LGWR switch)
  Current log# 3 seq# 2048 mem# 0: /rdsdbdata/db/ORCL_A/onlinelog/o1_mf_3_hbl2p8xs_.log
Thu May 28 23:59:10 2020
Archived Log entry 2037 added for thread 1 sequence 2047 ID 0x5d62ce43 dest 1:
Fri May 29 00:04:10 2020
```

```
Thread 1 advanced to log sequence 2049 (LGWR switch)
  Current log# 4 seq# 2049 mem# 0: /rdsdbdata/db/ORCL_A/onlinelog/o1_mf_4_hbl2qgmh_.log
Fri May 29 00:04:10 2020

10 rows selected.
```

You can also download the log file. For more information, see Downloading a database log file.

Performing miscellaneous tasks for Oracle DB instances

Following, you can find how to perform miscellaneous DBA tasks on your Amazon RDS DB instances running Oracle. To deliver a managed service experience, Amazon RDS doesn't provide shell access to DB instances, and restricts access to certain system procedures and tables that require advanced privileges.

Topics

- Creating and dropping directories in the main data storage space
- Listing files in a DB instance directory
- Reading files in a DB instance directory
- Accessing Opatch files
- Managing advisor tasks
- Transporting tablespaces

Creating and dropping directories in the main data storage space

To create directories, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.create_directory`. You can create up to 10,000 directories, all located in your main data storage space. To drop directories, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.drop_directory`.

The `create_directory` and `drop_directory` procedures have the following required parameter.

Parameter name	Data type	Default	Required	Description
`p_directory_name`	VARCHAR2	—	Yes	The name of the directory.

The following example creates a new directory named PRODUCT_DESCRIPTIONS.

```
EXEC rdsadmin.rdsadmin_util.create_directory(p_directory_name =>
 'product_descriptions');
```

The data dictionary stores the directory name in uppercase. You can list the directories by querying DBA_DIRECTORIES. The system chooses the actual host pathname automatically. The following example gets the directory path for the directory named PRODUCT_DESCRIPTIONS:

```
SELECT DIRECTORY_PATH
  FROM DBA_DIRECTORIES
 WHERE DIRECTORY_NAME='PRODUCT_DESCRIPTIONS';

DIRECTORY_PATH
--------------------------------------
/rdsdbdata/userdirs/01
```

The master user name for the DB instance has read and write privileges in the new directory, and can grant access to other users. EXECUTE privileges are not available for directories on a DB instance. Directories are created in your main data storage space and will consume space and I/O bandwidth.

The following example drops the directory named PRODUCT_DESCRIPTIONS.

```
EXEC rdsadmin.rdsadmin_util.drop_directory(p_directory_name => 'product_descriptions');
```

Note

You can also drop a directory by using the Oracle SQL command DROP DIRECTORY.

Dropping a directory doesn't remove its contents. Because the `rdsadmin.rdsadmin_util.create_directory` procedure can reuse pathnames, files in dropped directories can appear in a newly created directory. Before you drop a directory, we recommend that you use UTL_FILE.FREMOVE to remove files from the directory. For more information, see FREMOVE procedure in the Oracle documentation.

Listing files in a DB instance directory

To list the files in a directory, use the Amazon RDS procedure `rdsadmin.rds_file_util.listdir`. This procedure isn't supported on an Oracle replica. The `listdir` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_directory`	varchar2	—	Yes	The name of the directory to list.

The following example grants read/write privileges on the directory `PRODUCT_DESCRIPTIONS` to user `rdsadmin`, and then lists the files in this directory.

```
GRANT READ,WRITE ON DIRECTORY PRODUCT_DESCRIPTIONS TO rdsadmin;
SELECT * FROM TABLE(rdsadmin.rds_file_util.listdir(p_directory =>
 'PRODUCT_DESCRIPTIONS'));
```

Reading files in a DB instance directory

To read a text file, use the Amazon RDS procedure `rdsadmin.rds_file_util.read_text_file`. The `read_text_file` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
`p_directory`	varchar2	—	Yes	The name of the directory that contains the file.
`p_filename`	varchar2	—	Yes	The name of the file to read.

The following example creates the file `rice.txt` in the directory `PRODUCT_DESCRIPTIONS`.

```
declare
  fh sys.utl_file.file_type;
```

```
begin
  fh := utl_file.fopen(location=>'PRODUCT_DESCRIPTIONS', filename=>'rice.txt',
 open_mode=>'w');
  utl_file.put(file=>fh, buffer=>'AnyCompany brown rice, 15 lbs');
  utl_file.fclose(file=>fh);
end;
/
```

The following example reads the file `rice.txt` from the directory `PRODUCT_DESCRIPTIONS`.

```
SELECT * FROM TABLE
    (rdsadmin.rds_file_util.read_text_file(
        p_directory => 'PRODUCT_DESCRIPTIONS',
        p_filename  => 'rice.txt'));
```

Accessing Opatch files

Opatch is an Oracle utility that enables the application and rollback of patches to Oracle software. The Oracle mechanism for determining which patches have been applied to a database is the `opatch lsinventory` command. To open service requests for Bring Your Own Licence (BYOL) customers, Oracle Support requests the `lsinventory` file and sometimes the `lsinventory_detail` file generated by Opatch.

To deliver a managed service experience, Amazon RDS doesn't provide shell access to Opatch. Instead, the `lsinventory-`*`dbv`*`.txt` in the BDUMP directory contains the patch information related to your current engine version. When you perform a minor or major upgrade, Amazon RDS updates `lsinventory-`*`dbv`*`.txt` within an hour of applying the patch. To verify the applied patches, read `lsinventory-`*`dbv`*`.txt`. This action is similar to running the `opatch lsinventory` command.

> **Note**
>
> The examples in this section assume that the BDUMP directory is named `BDUMP`. On a read replica, the BDUMP directory name is different. To learn how to get the BDUMP name by querying `V$DATABASE.DB_UNIQUE_NAME` on a read replica, see Listing files.

The inventory files use the Amazon RDS naming convention `lsinventory-`*`dbv`*`.txt` and `lsinventory_detail-`*`dbv`*`.txt`, where *`dbv`* is the full name of your DB version.

The `lsinventory-dbv.txt` file is available on all DB versions. The corresponding `lsinventory_detail-dbv.txt` is available on 19.0.0.0, ru-2020-01.rur-2020-01.r1 or later.

For example, if your DB version is 19.0.0.0.ru-2021-07.rur-2021-07.r1, then your inventory files have the following names.

```
lsinventory-19.0.0.0.ru-2021-07.rur-2021-07.r1.txt
lsinventory_detail-19.0.0.0.ru-2021-07.rur-2021-07.r1.txt
```

Ensure that you download the files that match the current version of your DB engine.

Console

To download an inventory file using the console

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.
5. Scroll down to the **Logs** section.
6. In the **Logs** section, search for `lsinventory`.
7. Select the file that you want to access, and then choose **Download**.

SQL

To read the `lsinventory-dbv.txt` in a SQL client, you can use a `SELECT` statement. For this technique, use either of the following `rdsadmin` functions: `rdsadmin.rds_file_util.read_text_file` or `rdsadmin.tracefile_listing`.

In the following sample query, replace *dbv* with your Oracle DB version. For example, your DB version might be 19.0.0.0.ru-2020-04.rur-2020-04.r1.

```
SELECT text
FROM   TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP', 'lsinventory-dbv.txt'));
```

PL/SQL

To read the `lsinventory-dbv.txt` in a SQL client, you can write a PL/SQL program. This program uses `utl_file` to read the file, and `dbms_output` to print it. These are Oracle-supplied packages.

In the following sample program, replace *dbv* with your Oracle DB version. For example, your DB version might be 19.0.0.0.ru-2020-04.rur-2020-04.r1.

```
SET SERVEROUTPUT ON
DECLARE
  v_file              SYS.UTL_FILE.FILE_TYPE;
  v_line              VARCHAR2(1000);
  v_oracle_home_type  VARCHAR2(1000);
  c_directory         VARCHAR2(30) := 'BDUMP';
  c_output_file       VARCHAR2(30) := 'lsinventory-dbv.txt';
BEGIN
  v_file := SYS.UTL_FILE.FOPEN(c_directory, c_output_file, 'r');
  LOOP
    BEGIN
      SYS.UTL_FILE.GET_LINE(v_file, v_line,1000);
      DBMS_OUTPUT.PUT_LINE(v_line);
    EXCEPTION
      WHEN no_data_found THEN
        EXIT;
    END;
  END LOOP;
END;
/
```

Or query `rdsadmin.tracefile_listing`, and spool the output to a file. The following example spools the output to `/tmp/tracefile.txt`.

```
SPOOL /tmp/tracefile.txt
SELECT *
FROM   rdsadmin.tracefile_listing
WHERE  FILENAME LIKE 'lsinventory%';
SPOOL OFF;
```

Managing advisor tasks

Oracle Database includes a number of advisors. Each advisor supports automated and manual tasks. You can use procedures in the `rdsadmin.rdsadmin_util` package to manage some advisor tasks.

The advisor task procedures are available in the following engine versions:

- Oracle Database 21c (21.0.0)
- Version 19.0.0.0.ru-2021-01.rur-2021-01.r1 and higher Oracle Database 19c versions

  For more information, see Version 19.0.0.0.ru-2021-01.rur-2021-01.r1 in the *Amazon RDS for Oracle Release Notes*.

Topics

- Setting parameters for advisor tasks
- Disabling AUTO_STATS_ADVISOR_TASK
- Re-enabling AUTO_STATS_ADVISOR_TASK

Setting parameters for advisor tasks

To set parameters for some advisor tasks, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.advisor_task_set_parameter`. The `advisor_task_set_parameter` procedure has the following parameters.

Parameter name	Data type	Default	Require	Description
`p_task_name`	varchar2	—	Yes	The name of the advisor task whose parameters you want to change. The following values are valid: • `AUTO_STATS_ADVISOR_TASK` • `INDIVIDUAL_STATS_ADVISOR_TASK` • `SYS_AUTO_SPM_EVOLVE_TASK` • `SYS_AUTO_SQL_TUNING_TASK`

Parameter name	Data type	Default	Require	Description

p_parameter	varchar2	—	Yes	The name of the task parameter. To find valid parameters for an advisor task, run the following query. Substitute *p_task_name* with a valid value for p_task_name :
				`COL PARAMETER_NAME FORMAT a30`
				`COL PARAMETER_VALUE FORMAT a30`
				`SELECT PARAMETER_NAME, PARAMETER_VALUE`
				`FROM DBA_ADVISOR_PARAMETERS`
				`WHERE TASK_NAME='  p_task_name '`
				`AND PARAMETER_VALUE != 'UNUSED'`
				`ORDER BY PARAMETER_NAME;`

p_value	varchar2	—	Yes	The value for a task parameter. To find valid values for task parameters, run the following query. Substitute *p_task_name* with a valid value for p_task_name :
				`COL PARAMETER_NAME FORMAT a30`
				`COL PARAMETER_VALUE FORMAT a30`
				`SELECT PARAMETER_NAME, PARAMETER_VALUE`
				`FROM DBA_ADVISOR_PARAMETERS`
				`WHERE TASK_NAME='  p_task_name '`
				`AND PARAMETER_VALUE != 'UNUSED'`
				`ORDER BY PARAMETER_NAME;`

The following PL/SQL program sets ACCEPT_PLANS to FALSE for SYS_AUTO_SPM_EVOLVE_TASK. The SQL Plan Management automated task verifies the plans and generates a report of its findings, but does not evolve the plans automatically. You can use a report to identify new SQL plan baselines and accept them manually.

```
BEGIN
  rdsadmin.rdsadmin_util.advisor_task_set_parameter(
```

```
    p_task_name => 'SYS_AUTO_SPM_EVOLVE_TASK',
    p_parameter => 'ACCEPT_PLANS',
    p_value     => 'FALSE');
END;
```

The following PL/SQL program sets EXECUTION_DAYS_TO_EXPIRE to 10 for AUTO_STATS_ADVISOR_TASK. The predefined task AUTO_STATS_ADVISOR_TASK runs automatically in the maintenance window once per day. The example sets the retention period for the task execution to 10 days.

```
BEGIN
  rdsadmin.rdsadmin_util.advisor_task_set_parameter(
    p_task_name => 'AUTO_STATS_ADVISOR_TASK',
    p_parameter => 'EXECUTION_DAYS_TO_EXPIRE',
    p_value     => '10');
END;
```

Disabling AUTO_STATS_ADVISOR_TASK

To disable AUTO_STATS_ADVISOR_TASK, use the Amazon RDS procedure rdsadmin.rdsadmin_util.advisor_task_drop. The advisor_task_drop procedure accepts the following parameter.

Parameter name	Data type	Default	Require	Description
p_task_name	varchar2	—	Yes	The name of the advisor task to be disabled. The only valid value is AUTO_STATS_ADVISOR_TASK .

The following command drops AUTO_STATS_ADVISOR_TASK.

```
EXEC rdsadmin.rdsadmin_util.advisor_task_drop('AUTO_STATS_ADVISOR_TASK')
```

You can re-enable AUTO_STATS_ADVISOR_TASK using rdsadmin.rdsadmin_util.dbms_stats_init.

Re-enabling AUTO_STATS_ADVISOR_TASK

To re-enable AUTO_STATS_ADVISOR_TASK, use the Amazon RDS procedure `rdsadmin.rdsadmin_util.dbms_stats_init`. The `dbms_stats_init` procedure takes no parameters.

The following command re-enables AUTO_STATS_ADVISOR_TASK.

```
EXEC rdsadmin.rdsadmin_util.dbms_stats_init()
```

Transporting tablespaces

Use the Amazon RDS package `rdsadmin.rdsadmin_transport_util` to copy a set of tablespaces from an on-premises Oracle database to an RDS for Oracle DB instance. At the physical level, the transportable tablespace feature incrementally copies source data files and metadata files to your target instance. You can transfer the files using either Amazon EFS or Amazon S3. For more information, see Migrating using Oracle transportable tablespaces.

Topics

- Importing transported tablespaces to your DB instance
- Importing transportable tablespace metadata into your DB instance
- Listing orphaned files after a tablespace import
- Deleting orphaned data files after a tablespace import

Importing transported tablespaces to your DB instance

Use the procedure `rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces` to restore tablespaces that you have previously exported from a source DB instance. In the transport phase, you back up your read-only tablespaces, export Data Pump metadata, transfer these files to your target DB instance, and then import the tablespaces. For more information, see Phase 4: Transport the tablespaces.

Syntax

```
FUNCTION import_xtts_tablespaces(
    p_tablespace_list IN CLOB,
    p_directory_name  IN VARCHAR2,
    p_platform_id     IN NUMBER DEFAULT 13,
```

```
    p_parallel        IN INTEGER DEFAULT 0) RETURN VARCHAR2;
```

Parameters

Parameter name	Data type	Default	Required	Description
p_tablespace_list	CLOB	—	Yes	The list of tablespaces to import.
p_directory_name	VARCHAR2	—	Yes	The directory that contains the tablespace backups.
p_platform_id	NUMBER	13	No	Provide a platform ID that matches the one specified during the backup phase. To find a list of platforms, query V$TRANSPORTABLE_PLATFORM . The default platform is Linux x86 64-bit, which is little endian.
p_parallel	INTEGER	0	No	The degree of parallelism. By default, parallelism is disabled.

Examples

The following example imports the tablespaces *TBS1*, *TBS2*, and *TBS3* from the directory *DATA_PUMP_DIR*. The source platform is AIX-Based Systems (64-bit), which has the platform ID of 6. You can find the platform IDs by querying V$TRANSPORTABLE_PLATFORM.

```
VAR task_id CLOB

BEGIN
  :task_id:=rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces(
        'TBS1,TBS2,TBS3',
        'DATA_PUMP_DIR',
```

```
        p_platform_id => 6);
END;
/

PRINT task_id
```

Importing transportable tablespace metadata into your DB instance

Use the procedure `rdsadmin.rdsadmin_transport_util.import_xtts_metadata` to import transportable tablespace metadata into your RDS for Oracle DB instance. During the operation, the status of the metadata import is shown in the table `rdsadmin.rds_xtts_operation_info`. For more information, see Step 5: Import tablespace metadata on your target DB instance.

Syntax

```
PROCEDURE import_xtts_metadata(
    p_datapump_metadata_file IN SYS.DBA_DATA_FILES.FILE_NAME%TYPE,
    p_directory_name         IN VARCHAR2,
    p_exclude_stats          IN BOOLEAN DEFAULT FALSE,
    p_remap_tablespace_list  IN CLOB DEFAULT NULL,
    p_remap_user_list        IN CLOB DEFAULT NULL);
```

Parameters

Parameter name	Data type	Default	Required	Description
p_datapum p_metadat a_file	SYS.DBA_DATA_FILES .FILE_NAME%TYPE	—	Yes	The name of the Oracle Data Pump file that contains the metadata for your transportable tablespaces.
p_directo ry_name	VARCHAR2	—	Yes	The directory that contains the Data Pump file.
p_exclude_stats	BOOLEAN	FALSE	No	Flag that indicates whether to exclude statistics.

Parameter name	Data type	Default	Required	Description
p_remap_tablespace_list	CLOB	NULL	No	A list of tablespaces to be remapped during the metadata import. Use the format *from_tbs*:*to_tbs*. For example, specify users:user_data .
p_remap_user_list	CLOB	NULL	No	A list of user schemas to be remapped during the metadata import. Use the format *from_schema_name* :*to_schema_name* . For example, specify hr:human_resources .

Examples

The example imports the tablespace metadata from the file *xttdump.dmp*, which is located in directory *DATA_PUMP_DIR*.

```
BEGIN
  rdsadmin.rdsadmin_transport_util.import_xtts_metadata('xttdump.dmp','DATA_PUMP_DIR');
END;
/
```

Listing orphaned files after a tablespace import

Use the rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files procedure to list data files that were orphaned after a tablespace

import. After you identify the data files, you can delete them by calling `rdsadmin.rdsadmin_transport_util.cleanup_incomplete_xtts_import`.

Syntax

```
FUNCTION list_xtts_orphan_files RETURN xtts_orphan_files_list_t PIPELINED;
```

Examples

The following example runs the procedure `rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files`. The output shows two data files that are orphaned.

```
SQL> SELECT * FROM TABLE(rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files);

FILENAME       FILESIZE
-------------- ---------
datafile_7.dbf 104865792
datafile_8.dbf 104865792
```

Deleting orphaned data files after a tablespace import

Use the `rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files` procedure to delete data files that were orphaned after a tablespace import. Running this command generates a log file that uses the name format `rds-xtts-delete_xtts_orphaned_files-`*`YYYY-MM-DD.HH24-MI-SS.FF`*`.log` in the BDUMP directory. Use the procedure `rdsadmin.rdsadmin_transport_util.cleanup_incomplete_xtts_import` to find the orphaned files. You can read the log file by calling the procedure `rdsadmin.rds_file_util.read_text_file`. For more information, see Phase 6: Clean up leftover files.

Syntax

```
PROCEDURE cleanup_incomplete_xtts_import(
    p_directory_name IN VARCHAR2);
```

Parameters

Parameter name	Data type	Default	Required	Description
p_directory_name	VARCHAR2	—	Yes	The directory that contains the orphaned data files.

Examples

The following example deletes the orphaned data files in *DATA_PUMP_DIR*.

```
BEGIN
  rdsadmin.rdsadmin_transport_util.cleanup_incomplete_xtts_import('DATA_PUMP_DIR');
END;
/
```

The following example reads the log file generated by the previous command.

```
SELECT *
FROM TABLE(rdsadmin.rds_file_util.read_text_file(
       p_directory => 'BDUMP',
       p_filename  => 'rds-xtts-
delete_xtts_orphaned_files-2023-06-01.09-33-11.868894000.log'));

TEXT
--------------------------------------------------------------------------------
orphan transported datafile datafile_7.dbf deleted.
orphan transported datafile datafile_8.dbf deleted.
```

Working with storage in RDS for Oracle

Every RDS for Oracle instance has a primary storage volume. To increase storage capacity, you can attach up to three additional storage volumes to your DB instance. Depending on your workload requirements, choose between gp3 and io2 storage for each volume. For example, you might put frequently accessed data on an io2 volume and historical data on a gp3 volume.

Use additional storage volumes to enable the following benefits:

- **Enhanced capacity** – Scale your total storage up to 256 TiB per DB instance by attaching up to three additional storage volumes.
- **Flexible storage configuration and performance optimization** – Mix different storage types (gp3 and io2) to optimize for both cost and performance based on your data access patterns. Separate frequently accessed data on high-performance io2 storage from archival data on cost-effective gp3 storage.
- **Expand and reduce storage capacity as needed** – Attach a volume when you need additional storage, as during data migration, and then later delete the volume. In this way, you can expand and reduce the total DB instance storage.
- **Online data movement** – Use the built-in capabilities of Oracle database to move data between volumes without downtime.

> **Note**
>
> You can remove additional storage volumes, but you can't remove the primary volume.

Topics

- Considerations for using additional storage volumes with RDS for Oracle
- Limitations of using additional storage volumes with RDS for Oracle
- Database management operations with additional storage volumes in RDS for Oracle
- Add, remove, or modify storage volumes with RDS for Oracle
- Backing up and restoring data with additional storage volumes in RDS for Oracle
- Use cases for additional storage volumes in RDS for Oracle

Considerations for using additional storage volumes with RDS for Oracle

Consider the following when using additional storage volumes with RDS for Oracle:

- You can add up to 3 additional storage volumes per instance.
- Additional storage volumes must use the following volume names:
  - rdsdbdata2
  - rdsdbdata3
  - rdsdbdata4
- You can only add General Purpose SSD (gp3) and Provisioned IOPS SSD (io2) storage types.
- You can use Oracle's online relocation capabilities to move data between volumes while your applications continue running.
- When you create an additional storage volume by modifying the DB instance, RDS immediately creates the storage volume regardless of the schedule modifications setting. Adding a storage volume is an online operation and does not impact your database performance. See Using the schedule modifications setting.

For optimal performance, check the following when you are using additional storage volumes:

- Data movement planning
  - Schedule large movements during off-peak hours
  - Break large operations into smaller chunks
  - Monitor system resources during moves
- Resource management
  - Keep sufficient free space on both volumes
  - Monitor I/O patterns using AWR or Statspack
  - Watch for storage-full scenarios
- Best practices
  - Use online datafile relocation operations where possible
  - Maintain appropriate indexes
  - Regularly monitor space usage

When using additional storage volumes with replicas:

- When you are creating an RDS for Oracle replica for a DB instance that has additional storage volumes, RDS automatically configures additional storage volumes on the replica. However, any subsequent modifications made in storage volumes of your primary DB instance are not automatically applied to the replica.
- When managing datafile locations across volumes, we recommend using parameter group settings instead of session-level changes to ensure consistent behavior between primary and replica instances.

Limitations of using additional storage volumes with RDS for Oracle

The following limitations apply to using additional storage volumes with RDS for Oracle:

- You can’t add a storage volume to the instance types with less than 64GiB memory because they don’t have sufficient memory to support large storage volumes.
- The minimum storage size is 200GiB for additional storage volumes. The primary storage volume of your DB instance should be equal to or larger than 200GiB to attach additional storage volumes. The maximum storage size for your DB instance is 256 TiB total across all volumes.
- The following capabilities aren’t supported for DB instances with additional storage volumes:
  - Cross-region automated backups
  - Storage autoscaling (for additional storage volumes)
  - Cross-account snapshot copy
  - Public snapshots
- You can't delete the primary storage volume (`rdsdbdata`), but you can delete other additional storage volumes as long as they're empty.
- You can't store the online redo logs, archived redo logs, and control files in additional storage volumes. These files can only be stored in the primary storage volume (`rdsdbdata`).

Database management operations with additional storage volumes in RDS for Oracle

You can perform database management operations such as creating tablespaces or moving data between storage volumes while using additional storage volumes in RDS for Oracle. For more

information about database management operations with additional storage volumes, see the following sections:

- Specifying database file locations in RDS for Oracle
- Creating and sizing tablespaces in RDS for Oracle
- Moving data files between volumes in RDS for Oracle

Add, remove, or modify storage volumes with RDS for Oracle

You can add, modify, and remove additional storage volumes using the AWS Management Console or AWS CLI. All operations use the `modify-db-instance` command with the `additional-storage-volumes` parameter.

> **Important**
>
> Adding or removing additional storage volumes creates a backup pending action and a blackout window. The blackout window closes when the backup workflow completes.

Adding storage volumes

You can add up to three storage volumes beyond the primary storage volume. To add a new storage volume to your RDS for Oracle DB instance, use the `modify-db-instance` command with the `additional-storage-volumes` parameter.

The following code snippet adds a mew 5,000 GiB general purpose SSD (gp3) volume with 4000 provision IOPS name `rdsdbdata3`.

```
aws rds modify-db-instance \
  --db-instance-identifier my-oracle-instance \
  --region us-east-1 \
  --additional-storage-volumes '[
        {
            "VolumeName":"rdsdbdata3",
            "StorageType":"gp3",
            "AllocatedStorage":5000
            "IOPS":4000}
```

```
    ]' \
  --apply-immediately
```

Modifying storage volumes

You can modify storage type, allocated storage size, IOPS, and storage throughput settings of your additional storage volume. The following code snippet modifies the IOPS setting for the `rdsdbdata2` volume.

```
aws rds modify-db-instance \
  --db-instance-identifier my-oracle-instance \
  --region us-east-1 \
  --additional-storage-volumes '[
        {
            "VolumeName":"rdsdbdata2",
            "IOPS":8000}
    ]' \
  --apply-immediately
```

Note

You can't reduce the storage allocation for an additional storage volume once you've added it to the instance.

Removing storage volumes

You can remove additional storage volumes from RDS for Oracle DB instances when they are no longer needed. Before removing a volume, make sure that you have moved all database files from the volume and no database objects are referencing it. Verify that the volume status is `Not-in-use`. You can remove additional storage volumes, but you can't remove the primary storage volume.

Warning

Before you remove an additional storage volume, make sure that no database files are stored on the volume. Removing a volume with active database files causes database corruption.

The following example removes the `rdsdbdata4` volume.

```
aws rds modify-db-instance \
  --db-instance-identifier my-oracle-instance \
  --region us-east-1 \
  --additional-storage-volumes '[
        {
            "VolumeName":"rdsdbdata2",
            "SetForDelete":true}
    ]' \
  --apply-immediately
```

Backing up and restoring data with additional storage volumes in RDS for Oracle

You can use automated backups and create a DB snapshot with your DB instance with additional storage volumes. All backup operations include both the primary volume and additional storage volumes. You can also use point-in-time recovery for your DB instance with additional storage volumes. When you restore your database, you can add storage volumes. You can also modify the storage settings of existing volumes. You cannot delete additional storage volumes when you restore your database from a snapshot.

Topics

- Creating manual snapshots
- Restoring manual snapshots
- Point-in-time recovery

Creating manual snapshots

The following example creates a manual snapshot of your database with additional storage volumes:

```
aws rds create-db-snapshot \
--db-instance-identifier my-oracle-asv-instance \
--db-snapshot-identifier my-snapshot
```

Restoring manual snapshots

When restoring from a snapshot, you can add new additional storage volumes or modify the IOPS or throughput settings of existing volumes. The following example restores a DB instance from a snapshot and modifies the IOPS setting for the `rdsdbdata2` volume:

```
aws rds restore-db-instance-from-db-snapshot \
  --db-instance-identifier my-restored-instance \
  --db-snapshot-identifier my-snapshot \
  --region us-east-1 \
  --additional-storage-volumes '[
        {
            "VolumeName":"rdsdbdata2",
            "IOPS":5000
        }
    ]'
```

Point-in-time recovery

During point-in-time recovery (PITR), you can add new additional storage volumes with custom configurations. The following example performs PITR and adds a new 5,000 GiB General Purpose SSD (gp3) with 5000 IOPS and 200 MB/s storage throughput for the `rdsdbdata2` volume:

```
aws rds restore-db-instance-to-point-in-time \
  --source-db-instance-identifier my-source-instancemy-source-instance \
  --target-db-instance my-pitr-instance\
  --use-latest-restorable-time \
  --region us-east-1 \
  --additional-storage-volumes '[
        {
            "VolumeName":"rdsdbdata2",
            "StorageType":"gp3",
            "AllocatedStorage":5000,
            "IOPS":5000,
            "StorageThroughput":200
        }
    ]'
```

Use cases for additional storage volumes in RDS for Oracle

Additional storage volumes support various database management scenarios. The following sections describe common use cases and implementation approaches.

Topics

- Extending storage capacity beyond 64 TiB
- Storage tiering of frequently and infrequently accessed data on separate volumes
- Temporary storage for data loading and unloading
- Using Oracle transportable tablespaces with an additional storage volume

Extending storage capacity beyond 64 TiB

You can use additional storage volumes when your primary storage volume approaches the 64 TiB limit but need more storage space in your database. You can attach additional storage volumes to your DB instance, each up to 64TiB, using the `modify-db-instance` command. After attaching additional storage volumes, you can create tablespaces on additional storage volumes and move objects such as tables, indexes, and partitions to these tablespaces using standard Oracle SQL. For more information, see Database management operations with additional storage volumes in RDS for Oracle.

Storage tiering of frequently and infrequently accessed data on separate volumes

You can use additional storage volumes to optimize cost and performance by configuring different storage types between volumes. For example, you can use high-performance Provisioned IOPS SSD storage (io2) volumes for frequently accessed data while storing historical data on cost-effective General Purpose (gp3) storage volumes. You can move specific database objects (tables, indexes, and partitions) to these tablespaces using standard Oracle commands. For more information, see Database management operations with additional storage volumes in RDS for Oracle.

Temporary storage for data loading and unloading

You can use additional storage volumes as temporary storage for large data loads or exports with the following steps:

- Create a directory on an additional storage volume with the following command:

```
BEGIN
```

```
rdsadmin.rdsadmin_util.create_directory(
            p_directory_name => 'DATA_PUMP_DIR2',
            p_database_volume_name => 'rdsdbdata2');
END;
/
```

- After the creation of the directory, follow the steps described in Importing using Oracle Data Pump to export and import your data to the new directory.
- After the completion of the operation, remove files and optionally delete the volume to save the storage costs. You can remove the additional storage volume only when the volume is empty.

Using Oracle transportable tablespaces with an additional storage volume

You can use additional storage volumes to move datafiles to an additional storage volume using Oracle transportable tablespaces with the following steps:

- Set db_create_file_dest parameter at session level before importing transportable tablespaces into the target database with an additional storage volume.

```
ALTER SESSION SET db_create_file_dest = '/rdsdbdata2/db';

VAR x CLOB;

BEGIN
:x := rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces(
p_tablespace_list => 'TBTEST1',
p_directory_name => 'XTTS_DIR_DATA2',
p_platform_id => 13);
END;
/

PRINT :x;
```

- Check the transportable tablespace import status:

```
ALTER SESSION SET nls_date_format = 'DD.MM.YYYY HH24:MI:SS';

COL xtts_operation_start_utc FORMAT A30
COL xtts_operation_end_utc FORMAT A30
COL xtts_operation_state FORMAT A30
COL xtts_operation_type FORMAT A30
```

```
SELECT xtts_operation_start_utc, xtts_operation_type, xtts_operation_state
FROM rdsadmin.rds_xtts_operation_info;
```

- When transportable tablespace import completes, import transportable tablespace metadata.

```
BEGIN
rdsadmin.rdsadmin_transport_util.import_xtts_metadata(
p_datapump_metadata_file => 'xttdump.dmp',
p_directory_name => 'XTTS_DIR_DATA2');
END;
/
```

Configuring advanced RDS for Oracle features

RDS for Oracle supports various advanced features, including HugePages, an instance store, and extended data types.

Topics

- Storing temporary data in an RDS for Oracle instance store
- Turning on HugePages for an RDS for Oracle instance
- Turning on extended data types in RDS for Oracle

Storing temporary data in an RDS for Oracle instance store

Use an instance store for the temporary tablespaces and the Database Smart Flash Cache (the flash cache) on supported RDS for Oracle DB instance classes.

Topics

- Overview of the RDS for Oracle instance store
- Turning on an RDS for Oracle instance store
- Configuring an RDS for Oracle instance store
- Working with an instance store on an Oracle read replica
- Configuring a temporary tablespace group on an instance store and Amazon EBS
- Removing an RDS for Oracle instance store

Overview of the RDS for Oracle instance store

An *instance store* provides temporary block-level storage for an RDS for Oracle DB instance. You can use an instance store for temporary storage of information that changes frequently.

An instance store is based on Non-Volatile Memory Express (NVMe) devices that are physically attached to the host computer. The storage is optimized for low latency, random I/O performance, and sequential read throughput.

The size of the instance store varies by DB instance type. For more information about the instance store, see Amazon EC2 instance store in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

Topics

- Types of data in the RDS for Oracle instance store
- Benefits of the RDS for Oracle instance store
- Supported instance classes for the RDS for Oracle instance store
- Supported engine versions for the RDS for Oracle instance store
- Supported AWS Regions for the RDS for Oracle instance store
- Cost of the RDS for Oracle instance store

Types of data in the RDS for Oracle instance store

You can place the following types of RDS for Oracle temporary data in an instance store:

A temporary tablespace

Oracle Database uses temporary tablespaces to store intermediate query results that don't fit in memory. Larger queries can generate large amounts of intermediate data that needs to be cached temporarily, but doesn't need to persist. In particular, a temporary tablespace is useful for sorts, hash aggregations, and joins. If your RDS for Oracle DB instance uses the Enterprise Edition or Standard Edition 2, you can place a temporary tablespace in an instance store.

The flash cache

The flash cache improves the performance of single-block random reads in the conventional path. A best practice is to size the cache to accommodate most of your active data set. If your RDS for Oracle DB instance uses the Enterprise Edition, you can place the flash cache in an instance store.

By default, an instance store is configured for a temporary tablespace but not for the flash cache. You can't place Oracle data files and database log files in an instance store.

Benefits of the RDS for Oracle instance store

You might consider using an instance store to store temporary files and caches that you can afford to lose. If you want to improve DB performance, or if an increasing workload is causing performance problems for your Amazon EBS storage, consider scaling to an instance class that supports an instance store.

By placing your temporary tablespace and flash cache on an instance store, you get the following benefits:

- Lower read latencies
- Higher throughput
- Reduced load on your Amazon EBS volumes
- Lower storage and snapshot costs because of reduced Amazon EBS load
- Less need to provision high IOPS, possibly lowering your overall cost

By placing your temporary tablespace on the instance store, you deliver an immediate performance boost to queries that use temporary space. When you place the flash cache on the instance store, cached block reads typically have much lower latency than Amazon EBS reads. The flash cache needs to be "warmed up" before it delivers performance benefits. The cache warms up by itself because the database writes blocks to the flash cache as they age out of the database buffer cache.

> **Note**
>
> In some cases, the flash cache causes performance overhead because of cache management. Before you turn on the flash cache in a production environment, we recommend that you analyze your workload and test the cache in a test environment.

Supported instance classes for the RDS for Oracle instance store

Amazon RDS supports the instance store for the following DB instance classes:

- db.m5d
- db.m6id
- db.r5d
- db.r6id
- db.x2idn
- db.x2iedn

RDS for Oracle supports the preceding DB instance classes for the BYOL licensing model only. For more information, see Supported RDS for Oracle DB instance classes and Bring Your Own License (BYOL) for EE and SE2.

To see the total instance storage for the supported DB instance types, run the following command in the AWS CLI.

Example

```
aws ec2 describe-instance-types \
  --filters "Name=instance-type,Values=*5d.*large*,*6id.*large*" \
  --query "InstanceTypes[?contains(InstanceType,'m5d')||contains(InstanceType,'r5d')||
contains(InstanceType,'m6id')||contains(InstanceType,'r6id')][InstanceType,
 InstanceStorageInfo.TotalSizeInGB]" \
  --output table
```

The preceding command returns the raw device size for the instance store. RDS for Oracle uses a small portion of this space for configuration. The space in the instance store that is available for temporary tablespaces or the flash cache is slightly smaller.

Supported engine versions for the RDS for Oracle instance store

The instance store is supported for the following RDS for Oracle engine versions:

- 21.0.0.0.ru-2022-01.rur-2022-01.r1 or higher Oracle Database 21c versions
- 19.0.0.0.ru-2021-10.rur-2021-10.r1 or higher Oracle Database 19c versions

Supported AWS Regions for the RDS for Oracle instance store

The instance store is available in all AWS Regions where one or more of these instance types are supported. For more information on the db.m5d and db.r5d instance classes, see DB instance classes. For more information on the instance classes supported by Amazon RDS for Oracle, see RDS for Oracle DB instance classes.

Cost of the RDS for Oracle instance store

The cost of the instance store is built into the cost of the instance-store turned on instances. You don't incur additional costs by enabling an instance store on an RDS for Oracle DB instance. For more information about instance-store turned on instances, see Supported instance classes for the RDS for Oracle instance store.

Turning on an RDS for Oracle instance store

To turn on the instance store for RDS for Oracle temporary data, do one of the following:

- Create an RDS for Oracle DB instance using a supported instance class. For more information, see Creating an Amazon RDS DB instance.

- Modify an existing RDS for Oracle DB instance to use a supported instance class. For more information, see Modifying an Amazon RDS DB instance.

Configuring an RDS for Oracle instance store

By default, 100% of instance store space is allocated to the temporary tablespace. To configure the instance store to allocate space to the flash cache and temporary tablespace, set the following parameters in the parameter group for your instance:

db_flash_cache_size={DBInstanceStore*{0,2,4,6,8,10}/10}

This parameter specifies the amount of storage space allocated for the flash cache. This parameter is valid only for Oracle Database Enterprise Edition. The default value is `{DBInstanceStore*0/10}`. If you set a nonzero value for `db_flash_cache_size`, your RDS for Oracle instance enables the flash cache after you restart the instance.

rds.instance_store_temp_size={DBInstanceStore*{0,2,4,6,8,10}/10}

This parameter specifies the amount of storage space allocated for the temporary tablespace. The default value is `{DBInstanceStore*10/10}`. This parameter is modifiable for Oracle Database Enterprise Edition and read-only for Standard Edition 2. If you set a nonzero value for `rds.instance_store_temp_size`, Amazon RDS allocates space in the instance store for the temporary tablespace.

You can set the `db_flash_cache_size` and `rds.instance_store_temp_size` parameters for DB instances that don't use an instance store. In this case, both settings evaluate to `0`, which turns off the feature. In this case, you can use the same parameter group for different instance sizes and for instances that don't use an instance store. If you modify these parameters, make sure to reboot the associated instances so that the changes can take effect.

> **Important**
>
> If you allocate space for a temporary tablespace, Amazon RDS doesn't create the temporary tablespace automatically. To learn how to create the temporary tablespace on the instance store, see Creating a temporary tablespace on the instance store.

The combined value of the preceding parameters must not exceed 10/10, or 100%. The following table illustrates valid and invalid parameter settings.

db_flash_cache_size setting	rds.instance_store_temp_size setting	Explanation
db_flash_cache_size={DBInstanceStore*0/10}	rds.instance_store_temp_size={DBInstanceStore*10/10}	This is a valid configuration for all editions of Oracle Database. Amazon RDS allocates 100% of instance store space to the temporary tablespace. This is the default.
db_flash_cache_size={DBInstanceStore*10/10}	rds.instance_store_temp_size={DBInstanceStore*0/10}	This is a valid configuration for Oracle Database Enterprise Edition only. Amazon RDS allocates 100% of instance store space to the flash cache.
db_flash_cache_size={DBInstanceStore*2/10}	rds.instance_store_temp_size={DBInstanceStore*8/10}	This is a valid configuration for Oracle Database Enterprise Edition only. Amazon RDS allocates 20% of instance store space to

db_flash_cache_size setting	rds.instance_store_temp_size setting	Explanation
		the flash cache, and 80% of instance store space to the temporary tablespace.
db_flash_cache_size={DBInstanceStore*6/10}	rds.instance_store_temp_size={DBInstanceStore*4/10}	This is a valid configuration for Oracle Database Enterprise Edition only. Amazon RDS allocates 60% of instance store space to the flash cache, and 40% of instance store space to the temporary tablespace.

db_flash_cache_size setting	rds.instance_store_temp_size setting	Explanation
db_flash_cache_size={DBInstanceStore*2/10}	rds.instance_store_temp_size={DBInstanceStore*4/10}	This is a valid configuration for Oracle Database Enterprise Edition only. Amazon RDS allocates 20% of instance store space to the flash cache, and 40% of instance store space to the temporary tablespace.
db_flash_cache_size={DBInstanceStore*8/10}	rds.instance_store_temp_size={DBInstanceStore*8/10}	This is an invalid configuration because the combined percentage of instance store space exceeds 100%. In such cases, Amazon RDS fails the attempt.

Considerations when changing the DB instance type

If you change your DB instance type, it can affect the configuration of the flash cache or the temporary tablespace on the instance store. Consider the following modifications and their effects:

You scale up or scale down the DB instance that supports the instance store.

The following values increase or decrease proportionally to the new size of the instance store:

- The new size of the flash cache.
- The space allocated to the temporary tablespaces that reside in the instance store.

For example, the setting `db_flash_cache_size={DBInstanceStore*6/10}` on a db.m5d.4xlarge instance provides around 340 GB of flash cache space. If you scale up the instance type to db.m5d.8xlarge, the flash cache space increases to around 680 GB.

You modify a DB instance that doesn't use an instance store to an instance that does use an instance store.

If `db_flash_cache_size` is set to a value larger than `0`, the flash cache is configured. If `rds.instance_store_temp_size` is set to a value larger than `0`, the instance store space is allocated for use by a temporary tablespace. RDS for Oracle doesn't move tempfiles to the instance store automatically. For information about using the allocated space, see Creating a temporary tablespace on the instance store or Adding a tempfile to the instance store on a read replica.

You modify a DB instance that uses an instance store to an instance that doesn't use an instance store.

In this case, RDS for Oracle removes the flash cache. RDS re-creates the tempfile that is currently located on the instance store on an Amazon EBS volume. The maximum size of the new tempfile is the former size of the `rds.instance_store_temp_size` parameter.

Working with an instance store on an Oracle read replica

Read replicas support the flash cache and temporary tablespaces on an instance store. While the flash cache works the same way as on the primary DB instance, note the following differences for temporary tablespaces:

- You can't create a temporary tablespace on a read replica. If you create a new temporary tablespace on the primary instance, RDS for Oracle replicates the tablespace information without tempfiles. To add a new tempfile, use either of the following techniques:
  - Use the Amazon RDS procedure `rdsadmin.rdsadmin_util.add_inst_store_tempfile`. RDS for Oracle creates a tempfile in the instance store on your read replica, and adds it to the specified temporary tablespace.

- Run the `ALTER TABLESPACE ... ADD TEMPFILE` command. RDS for Oracle places the tempfile on Amazon EBS storage.

> **Note**
>
> The tempfile sizes and storage types can be different on the primary DB instance and the read replica.

- You can manage the default temporary tablespace setting only on the primary DB instance. RDS for Oracle replicates the setting to all read replicas.
- You can configure the temporary tablespace groups only on the primary DB instance. RDS for Oracle replicates the setting to all read replicas.

Configuring a temporary tablespace group on an instance store and Amazon EBS

You can configure a temporary tablespace group to include temporary tablespaces on both an instance store and Amazon EBS. This technique is useful when you want more temporary storage than is allowed by the maximum setting of `rds.instance_store_temp_size`.

When you configure a temporary tablespace group on both an instance store and Amazon EBS, the two tablespaces have significantly different performance characteristics. Oracle Database chooses the tablespace to serve queries based on an internal algorithm. Therefore, similar queries can vary in performance.

Typically, you create a temporary tablespace in the instance store as follows:

1. Create a temporary tablespace in the instance store.
2. Set the new tablespace as the database default temporary tablespace.

If the tablespace size in the instance store is insufficient, you can create additional temporary storage as follows:

1. Assign the temporary tablespace in the instance store to a temporary tablespace group.
2. Create a new temporary tablespace in Amazon EBS if one doesn't exist.
3. Assign the temporary tablespace in Amazon EBS to the same tablespace group that includes the instance store tablespace.
4. Set the tablespace group as the default temporary tablespace.

The following example assumes that the size of the temporary tablespace in the instance store doesn't meet your application requirements. The example creates the temporary tablespace `temp_in_inst_store` in the instance store, assigns it to tablespace group `temp_group`, adds the existing Amazon EBS tablespace named `temp_in_ebs` to this group, and sets this group as the default temporary tablespace.

```
SQL> EXEC rdsadmin.rdsadmin_util.create_inst_store_tmp_tblspace('temp_in_inst_store');

PL/SQL procedure successfully completed.

SQL> ALTER TABLESPACE temp_in_inst_store TABLESPACE GROUP temp_group;

Tablespace altered.

SQL> ALTER TABLESPACE temp_in_ebs TABLESPACE GROUP temp_group;

Tablespace altered.

SQL> EXEC rdsadmin.rdsadmin_util.alter_default_temp_tablespace('temp_group');

PL/SQL procedure successfully completed.

SQL> SELECT * FROM DBA_TABLESPACE_GROUPS;

GROUP_NAME                     TABLESPACE_NAME
------------------------------ ------------------------------
TEMP_GROUP                     TEMP_IN_EBS
TEMP_GROUP                     TEMP_IN_INST_STORE

SQL> SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
 PROPERTY_NAME='DEFAULT_TEMP_TABLESPACE';

PROPERTY_VALUE
--------------
TEMP_GROUP
```

Removing an RDS for Oracle instance store

To remove the instance store, modify your RDS for Oracle DB instance to use an instance type that doesn't support instance store, such as db.m5 or db.r5.

Turning on HugePages for an RDS for Oracle instance

Amazon RDS for Oracle supports Linux kernel HugePages for increased database scalability. HugePages results in smaller page tables and less CPU time spent on memory management, increasing the performance of large database instances. For more information, see Overview of HugePages in the Oracle documentation.

You can use HugePages with all supported versions and editions of RDS for Oracle.

The `use_large_pages` parameter controls whether HugePages are turned on for a DB instance. The possible settings for this parameter are `ONLY`, `FALSE`, and `{DBInstanceClassHugePagesDefault}`. The `use_large_pages` parameter is set to `{DBInstanceClassHugePagesDefault}` in the default DB parameter group for Oracle.

To control whether HugePages are turned on for a DB instance automatically, you can use the `DBInstanceClassHugePagesDefault` formula variable in parameter groups. The value is determined as follows:

- For the DB instance classes mentioned in the table following, `DBInstanceClassHugePagesDefault` always evaluates to `FALSE` by default, and `use_large_pages` evaluates to `FALSE`. You can turn on HugePages manually for these DB instance classes if the DB instance class has at least 14 GiB of memory.
- For DB instance classes not mentioned in the table following, if the DB instance class has less than 14 GiB of memory, `DBInstanceClassHugePagesDefault` always evaluates to `FALSE`. Also, `use_large_pages` evaluates to `FALSE`.
- For DB instance classes not mentioned in the table following, if the instance class has at least 14 GiB of memory and less than 100 GiB of memory, `DBInstanceClassHugePagesDefault` evaluates to `TRUE` by default. Also, `use_large_pages` evaluates to `ONLY`. You can turn off HugePages manually by setting `use_large_pages` to `FALSE`.
- For DB instance classes not mentioned in the table following, if the instance class has at least 100 GiB of memory, `DBInstanceClassHugePagesDefault` always evaluates to `TRUE`. Also, `use_large_pages` evaluates to `ONLY` and HugePages can't be disabled.

HugePages are not turned on by default for the following DB instance classes.

DB instance class family	DB instance classes with HugePages not turned on by default
db.m5	db.m5.large
db.m4	db.m4.large, db.m4.xlarge, db.m4.2xlarge, db.m4.4xlarge, db.m4.10xlarge
db.t3	db.t3.micro, db.t3.small, db.t3.medium, db.t3.large

For more information about DB instance classes, see Hardware specifications for DB instance classes.

To turn on HugePages for new or existing DB instances manually, set the `use_large_pages` parameter to `ONLY`. You can't use HugePages with Oracle Automatic Memory Management (AMM). If you set the parameter `use_large_pages` to `ONLY`, then you must also set both `memory_target` and `memory_max_target` to `0`. For more information about setting DB parameters for your DB instance, see Parameter groups for Amazon RDS.

You can also set the `sga_target`, `sga_max_size`, and `pga_aggregate_target` parameters. When you set system global area (SGA) and program global area (PGA) memory parameters, add the values together. Subtract this total from your available instance memory (`DBInstanceClassMemory`) to determine the free memory beyond the HugePages allocation. You must leave free memory of at least 2 GiB, or 10 percent of the total available instance memory, whichever is smaller.

After you configure your parameters, you must reboot your DB instance for the changes to take effect. For more information, see Rebooting a DB instance.

Note

The Oracle DB instance defers changes to SGA-related initialization parameters until you reboot the instance without failover. In the Amazon RDS console, choose **Reboot** but *do not* choose **Reboot with failover**. In the AWS CLI, call the `reboot-db-instance` command with the `--no-force-failover` parameter. The DB instance does not process the SGA-related parameters during failover or during other maintenance operations that cause the instance to restart.

The following is a sample parameter configuration for HugePages that enables HugePages manually. You should set the values to meet your needs.

```
memory_target            = 0
memory_max_target        = 0
pga_aggregate_target     = {DBInstanceClassMemory*1/8}
sga_target               = {DBInstanceClassMemory*3/4}
sga_max_size             = {DBInstanceClassMemory*3/4}
use_large_pages          = ONLY
```

Assume the following parameters values are set in a parameter group.

```
memory_target            = IF({DBInstanceClassHugePagesDefault}, 0,
 {DBInstanceClassMemory*3/4})
memory_max_target        = IF({DBInstanceClassHugePagesDefault}, 0,
 {DBInstanceClassMemory*3/4})
pga_aggregate_target     = IF({DBInstanceClassHugePagesDefault},
 {DBInstanceClassMemory*1/8}, 0)
sga_target               = IF({DBInstanceClassHugePagesDefault},
 {DBInstanceClassMemory*3/4}, 0)
sga_max_size             = IF({DBInstanceClassHugePagesDefault},
 {DBInstanceClassMemory*3/4}, 0)
use_large_pages          = {DBInstanceClassHugePagesDefault}
```

The parameter group is used by a db.r4 DB instance class with less than 100 GiB of memory. With these parameter settings and `use_large_pages` set to `{DBInstanceClassHugePagesDefault}`, HugePages are turned on for the db.r4 instance.

Consider another example with following parameters values set in a parameter group.

```
memory_target           = IF({DBInstanceClassHugePagesDefault}, 0,
 {DBInstanceClassMemory*3/4})
memory_max_target       = IF({DBInstanceClassHugePagesDefault}, 0,
 {DBInstanceClassMemory*3/4})
pga_aggregate_target    = IF({DBInstanceClassHugePagesDefault},
 {DBInstanceClassMemory*1/8}, 0)
sga_target              = IF({DBInstanceClassHugePagesDefault},
 {DBInstanceClassMemory*3/4}, 0)
sga_max_size            = IF({DBInstanceClassHugePagesDefault},
 {DBInstanceClassMemory*3/4}, 0)
use_large_pages         = FALSE
```

The parameter group is used by a db.r4 DB instance class and a db.r5 DB instance class, both with less than 100 GiB of memory. With these parameter settings, HugePages are turned off on the db.r4 and db.r5 instance.

> **Note**
>
> If this parameter group is used by a db.r4 DB instance class or db.r5 DB instance class with at least 100 GiB of memory, the `FALSE` setting for `use_large_pages` is overridden and set to `ONLY`. In this case, a customer notification regarding the override is sent.

After HugePages are active on your DB instance, you can view HugePages information by enabling enhanced monitoring. For more information, see Monitoring OS metrics with Enhanced Monitoring.

Turning on extended data types in RDS for Oracle

Amazon RDS for Oracle supports extended data types. With extended data types, the maximum size is 32,767 bytes for the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types. To use extended data types, set the `MAX_STRING_SIZE` parameter to `EXTENDED`. For more information, see Extended data types in the Oracle documentation.

If you don't want to use extended data types, keep the `MAX_STRING_SIZE` parameter set to `STANDARD` (the default). In this case, the size limits are 4,000 bytes for the `VARCHAR2` and `NVARCHAR2` data types, and 2,000 bytes for the RAW data type.

You can turn on extended data types on a new or existing DB instance. For new DB instances, DB instance creation time is typically longer when you turn on extended data types. For existing DB instances, the DB instance is unavailable during the conversion process.

Considerations for extended data types

Consider the following when you enable extended data types for your DB instance:

- When you turn on extended data types for a new or existing DB instance, you must reboot the instance for the change to take effect.
- After you turn on extended data types, you can't change the DB instance back to use the standard size for data types. If you set the `MAX_STRING_SIZE` parameter back to `STANDARD` it results in the `incompatible-parameters` status.

- When you restore a DB instance that uses extended data types, you must specify a parameter group with the `MAX_STRING_SIZE` parameter set to `EXTENDED`. During restore, if you specify the default parameter group or any other parameter group with `MAX_STRING_SIZE` set to `STANDARD` it results in the `incompatible-parameters` status.
- When the DB instance status is `incompatible-parameters` because of the `MAX_STRING_SIZE` setting, the DB instance remains unavailable until you set the `MAX_STRING_SIZE` parameter to `EXTENDED` and reboot the DB instance.

Turning on extended data types for a new DB instance

When you create a DB instance with `MAX_STRING_SIZE` set to `EXTENDED`, the instance shows `MAX_STRING_SIZE` set to the default `STANDARD`. Reboot the instance to enable the change.

To turn on extended data types for a new DB instance

1. Set the `MAX_STRING_SIZE` parameter to `EXTENDED` in a parameter group.

   To set the parameter, you can either create a new parameter group or modify an existing parameter group.

   For more information, see Parameter groups for Amazon RDS.
2. Create a new RDS for Oracle DB instance.

   For more information, see Creating an Amazon RDS DB instance.
3. Associate the parameter group with `MAX_STRING_SIZE` set to `EXTENDED` with the DB instance.

   For more information, see Creating an Amazon RDS DB instance.
4. Reboot the DB instance for the parameter change to take effect.

   For more information, see Rebooting a DB instance.

Turning on extended data types for an existing DB instance

When you modify a DB instance to turn on extended data types, RDS converts the data in the database to use the extended sizes. The conversion and downtime occur when you next reboot the database after the parameter change. The DB instance is unavailable during the conversion.

The amount of time it takes to convert the data depends on the DB instance class, the database size, and the time of the last DB snapshot. To reduce downtime, consider taking a snapshot immediately before rebooting. This shortens the time of the backup that occurs during the conversion workflow.

> **Note**
>
> After you turn on extended data types, you can't perform a point-in-time restore to a time during the conversion. You can restore to the time immediately before the conversion or after the conversion.

To turn on extended data types for an existing DB instance

1. Take a snapshot of the database.

   If there are invalid objects in the database, Amazon RDS tries to recompile them. The conversion to extended data types can fail if Amazon RDS can't recompile an invalid object. The snapshot enables you to restore the database if there is a problem with the conversion. Always check for invalid objects before conversion and fix or drop those invalid objects. For production databases, we recommend testing the conversion process on a copy of your DB instance first.

   For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.
2. Set the `MAX_STRING_SIZE` parameter to `EXTENDED` in a parameter group.

   To set the parameter, you can either create a new parameter group or modify an existing parameter group.

   For more information, see Parameter groups for Amazon RDS.
3. Modify the DB instance to associate it with the parameter group with `MAX_STRING_SIZE` set to `EXTENDED`.

   For more information, see Modifying an Amazon RDS DB instance.
4. Reboot the DB instance for the parameter change to take effect.

   For more information, see Rebooting a DB instance.

Importing data into Oracle on Amazon RDS

How you import data into an Amazon RDS for Oracle DB instance depends on the following:

- The amount of data you have
- The number of database objects in your database
- The variety of database objects in your database

For example, you can use the following tools, depending on your requirements:

- Oracle SQL Developer – Import a simple, 20 MB database.
- Oracle Data Pump – Import complex databases, or databases that are several hundred megabytes or several terabytes in size. For example, you can transport tablespaces from an on-premises database to your RDS for Oracle DB instance. You can use Amazon S3 or Amazon EFS to transfer the data files and metadata. For more information, see Migrating using Oracle transportable tablespaces, Amazon EFS integration, and Amazon S3 integration.
- AWS Database Migration Service (AWS DMS) – Migrate databases without downtime. For more information about AWS DMS, see What is AWS Database Migration Service and the blog post Migrating Oracle databases with near-zero downtime using AWS DMS.

> **Important**
>
> Before you use the preceding migration techniques, we recommend that you back up your database. After you import the data, you can back up your RDS for Oracle DB instances by creating snapshots. Later, you can restore the snapshots. For more information, see Backing up, restoring, and exporting data.

For many database engines, ongoing replication can continue until you are ready to switch over to the target database. You can use AWS DMS to migrate to RDS for Oracle from either the same database engine or a different engine. If you migrate from a different database engine, you can use the AWS Schema Conversion Tool to migrate schema objects that AWS DMS doesn't migrate.

Topics

- Importing using Oracle SQL Developer
- Migrating using Oracle transportable tablespaces

- Importing using Oracle Data Pump
- Importing using Oracle Export/Import
- Importing using Oracle SQL*Loader
- Migrating with Oracle materialized views

Importing using Oracle SQL Developer

Oracle SQL Developer is a graphical Java tool distributed without cost by Oracle. SQL Developer provides options for migrating data between two Oracle databases, or for migrating data from other databases, such as MySQL, to an Oracle database. This tool is best for migrating small databases.

You can install this tool on your desktop computer (Windows, Linux, or Mac) or on one of your servers. After you install SQL Developer, you can use it to connect to your source and target databases. Use the **Database Copy** command on the Tools menu to copy your data to your RDS for Oracle DB instance.

To download SQL Developer, go to http://www.oracle.com/technetwork/developer-tools/sql-developer.

We recommend that you read the Oracle SQL Developer product documentation before you begin migrating your data. Oracle also has documentation on how to migrate from other databases, including MySQL and SQL Server. For more information, see http://www.oracle.com/technetwork/database/migration in the Oracle documentation.

Migrating using Oracle transportable tablespaces

You can use the Oracle transportable tablespaces feature to copy a set of tablespaces from an on-premises Oracle database to an RDS for Oracle DB instance. At the physical level, you transfer source data files and metadata files to your target DB instance using either Amazon EFS or Amazon S3. The transportable tablespaces feature uses the `rdsadmin.rdsadmin_transport_util` package. For syntax and semantics of this package, see Transporting tablespaces.

For blog posts that explain how to transport tablespaces, see Migrate Oracle Databases to AWS using transportable tablespace and Amazon RDS for Oracle Transportable Tablespaces using RMAN.

Topics

- Overview of Oracle transportable tablespaces
- Phase 1: Set up your source host
- Phase 2: Prepare the full tablespace backup
- Phase 3: Make and transfer incremental backups
- Phase 4: Transport the tablespaces
- Phase 5: Validate the transported tablespaces
- Phase 6: Clean up leftover files

Overview of Oracle transportable tablespaces

A transportable tablespace set consists of data files for the set of tablespaces being transported and an export dump file containing tablespace metadata. In a physical migration solution such as transportable tablespaces, you transfer physical files: data files, configuration files, and Data Pump dump files.

Topics

- Advantages and disadvantages of transportable tablespaces
- Limitations for transportable tablespaces
- Prerequisites for transportable tablespaces

Advantages and disadvantages of transportable tablespaces

We recommend that you use transportable tablespaces when you need to migrate one or more large tablespaces to RDS with minimum downtime. Transportable tablespaces offer the following advantages over logical migration:

- Downtime is lower than most other Oracle migration solutions.
- Because the transportable tablespace feature copies only physical files, it avoids the data integrity errors and logical corruption that can occur in logical migration.
- No additional license is required.
- You can migrate a set of tablespaces across different platforms and endianness types, for example, from an Oracle Solaris platform to Linux. However, transporting tablespaces to and from Windows servers isn't supported.

> **Note**
>
> Linux is fully tested and supported. Not all UNIX variations have been tested.

If you use transportable tablespaces, you can transport data using either Amazon S3 or Amazon EFS:

- When you use EFS, your backups remain in the EFS file system for the duration of the import. You can remove the files afterward. In this technique, you don't need to provision EBS storage for your DB instance. For this reason, we recommend using Amazon EFS instead of S3. For more information, see Amazon EFS integration.
- When you use S3, you download RMAN backups to EBS storage attached to your DB instance. The files remain in your EBS storage during the import. After the import, you can free up this space, which remains allocated to your DB instance.

The primary disadvantage of transportable tablespaces is that you need relatively advanced knowledge of Oracle Database. For more information, see Transporting Tablespaces Between Databases in the *Oracle Database Administrator's Guide*.

Limitations for transportable tablespaces

Oracle Database limitations for transportable tablespaces apply when you use this feature in RDS for Oracle. For more information, see Limitations on Transportable Tablespaces and General Limitations on Transporting Data in the *Oracle Database Administrator's Guide*. Note the following additional limitations for transportable tablespaces in RDS for Oracle:

- Neither the source or target database can use Standard Edition 2 (SE2). Only Enterprise Edition is supported.
- You can't use an Oracle Database 11g database as a source. The RMAN cross-platform transportable tablespaces feature relies on the RMAN transport mechanism, which Oracle Database 11g doesn't support.
- You can't migrate data from an RDS for Oracle DB instance using transportable tablespaces. You can only use transportable tablespaces to migrate data to an RDS for Oracle DB instance.
- The Windows operating system isn't supported.

- You can't transport tablespaces into a database at a lower release level. The target database must be at the same or later release level as the source database. For example, you can't transport tablespaces from Oracle Database 21c into Oracle Database 19c.
- You can't transport administrative tablespaces such as SYSTEM and SYSAUX.
- You can't transport non-data objects such as PL/SQL packages, Java classes, views, triggers, sequences, users, roles, and temporary tables. To transport non-data objects, create them manually or use Data Pump metadata export and import. For more information, see My Oracle Support Note 1454872.1.
- You can't transport tablespaces that are encrypted or use encrypted columns.
- If you transfer files using Amazon S3, the maximum supported file size is 5 TiB.
- If the source database uses Oracle options such as Spatial, you can't transport tablespaces unless the same options are configured on the target database.
- You can't transport tablespaces into an RDS for Oracle DB instance in an Oracle replica configuration. As a workaround, you can delete all replicas, transport the tablespaces, and then recreate the replicas.

Prerequisites for transportable tablespaces

Before you begin, complete the following tasks:

- Review the requirements for transportable tablespaces described in the following documents in My Oracle Support:
  - Reduce Transportable Tablespace Downtime using Cross Platform Incremental Backup (Doc ID 2471245.1)
  - Transportable Tablespace (TTS) Restrictions and Limitations: Details, Reference, and Version Where Applicable (Doc ID 1454872.1)
  - Primary Note for Transportable Tablespaces (TTS) -- Common Questions and Issues (Doc ID 1166564.1)
- Plan for endianness conversion. If you specify the source platform ID, RDS for Oracle converts the endianness automatically. To learn how to find platform IDs, see Data Guard Support for Heterogeneous Primary and Physical Standbys in Same Data Guard Configuration (Doc ID 413484.1).
- Make sure that the transportable tablespace feature is enabled on your target DB instance. The feature is enabled only if you don't get an ORA-20304 error when you run the following query:

```
SELECT * FROM TABLE(rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files);
```

If the transportable tablespace feature isn't enabled, reboot your DB instance. For more information, see Rebooting a DB instance.

- Make sure that the time zone file is the same in the source and target databases.
- Make sure that the database character sets on the source and target databases meet either of the following requirements:
  - The character sets are the same.
  - The character sets are compatible. For a list of compatibility requirements, see General Limitations on Transporting Data in the Oracle Database documentation.
- If you plan to transfer files using Amazon S3, do the following:
  - Make sure that an Amazon S3 bucket is available for file transfers, and that the Amazon S3 bucket is in the same AWS Region as your DB instance. For instructions, see Create a bucket in the *Amazon Simple Storage Service Getting Started Guide*.
  - Prepare the Amazon S3 bucket for Amazon RDS integration by following the instructions in Configuring IAM permissions for RDS for Oracle integration with Amazon S3.
- If you plan to transfer files using Amazon EFS, make sure that you have configured EFS according to the instructions in Amazon EFS integration.
- We strongly recommend that you turn on automatic backups in your target DB instance. Because the metadata import step can potentially fail, it's important to be able to restore your DB instance to its state before the import, thereby avoiding the necessity to back up, transfer, and import your tablespaces again.

Phase 1: Set up your source host

In this step, you copy the transport tablespaces scripts provided by My Oracle Support and set up necessary configuration files. In the following steps, the *source host* is running the database that contains the tablespaces to be transported to your *target instance*.

To set up your source host

1. Log in to your source host as the owner of your Oracle home.
2. Make sure that your `ORACLE_HOME` and `ORACLE_SID` environment variables point to your source database.

3. Log in to your database as an administrator, and verify that the time zone version, DB character set, and national character set are the same as in your target database.

```
SELECT * FROM V$TIMEZONE_FILE;
SELECT * FROM NLS_DATABASE_PARAMETERS
  WHERE PARAMETER IN ('NLS_CHARACTERSET','NLS_NCHAR_CHARACTERSET');
```

4. Set up the transportable tablespace utility as described in Oracle Support note 2471245.1.

   The setup includes editing the `xtt.properties` file on your source host. The following sample `xtt.properties` file specifies backups of three tablespaces in the `/dsk1/backups` directory. These are the tablespaces that you intend to transport to your target DB instance. It also specifies the source platform ID to convert the endianness automatically.

   > **Note**
   >
   > For valid platform IDs, see Data Guard Support for Heterogeneous Primary and Physical Standbys in Same Data Guard Configuration (Doc ID 413484.1).

```
#linux system
platformid=13
#list of tablespaces to transport
tablespaces=TBS1,TBS2,TBS3
#location where backup will be generated
src_scratch_location=/dsk1/backups
#RMAN command for performing backup
usermantransport=1
```

Phase 2: Prepare the full tablespace backup

In this phase, you back up your tablespaces for the first time, transfer the backups to your target host, and then restore them using the procedure `rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces`. When this phase is complete, the initial tablespace backups reside on your target DB instance and can be updated with incremental backups.

Topics

- Step 1: Back up the tablespaces on your source host
- Step 2: Transfer the backup files to your target DB instance
- Step 3: Import the tablespaces on your target DB instance

Step 1: Back up the tablespaces on your source host

In this step, you use the `xttdriver.pl` script to make a full backup of your tablespaces. The output of `xttdriver.pl` is stored in the `TMPDIR` environment variable.

To back up your tablespaces

1. If your tablespaces are in read-only mode, log in to your source database as a user with the `ALTER TABLESPACE` privilege, and place your tablespaces in read/write mode. Otherwise, skip to the next step.

   The following example places `tbs1`, `tbs2`, and `tbs3` in read/write mode.

   ```
   ALTER TABLESPACE tbs1 READ WRITE;
   ALTER TABLESPACE tbs2 READ WRITE;
   ALTER TABLESPACE tbs3 READ WRITE;
   ```

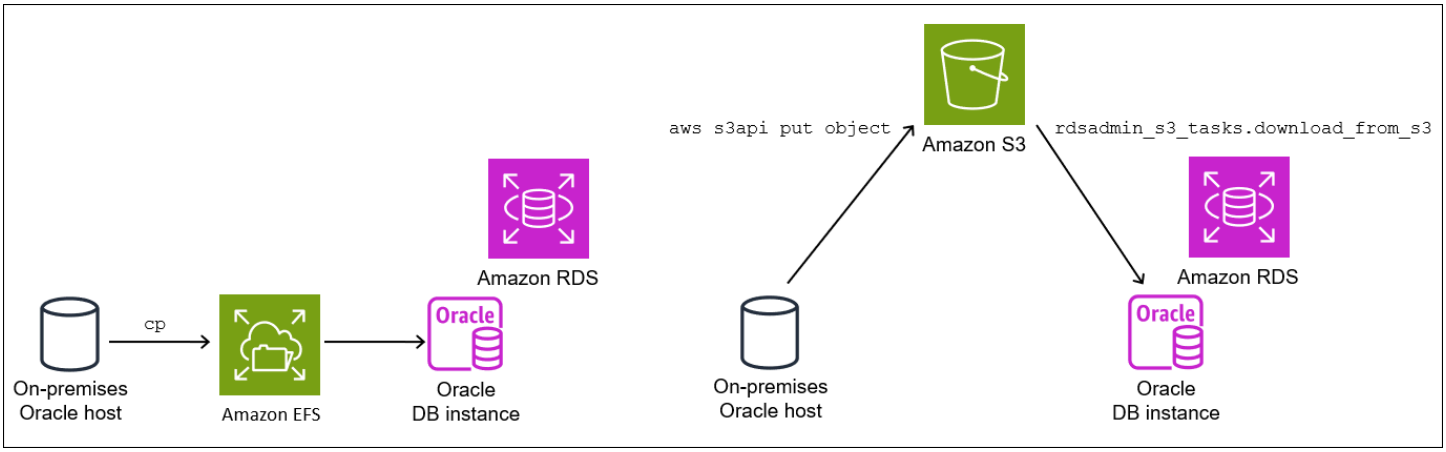
2. Back up your tablespaces using the `xttdriver.pl` script. Optionally, you can specify `--debug` to run the script in debug mode.

   ```
   export TMPDIR=location_of_log_files
   cd location_of_xttdriver.pl
   $ORACLE_HOME/perl/bin/perl xttdriver.pl --backup
   ```

Step 2: Transfer the backup files to your target DB instance

In this step, copy the backup and configuration files from your scratch location to your target DB instance. Choose one of the following options:

- If the source and target hosts share an Amazon EFS file system, use an operating system utility such as `cp` to copy your backup files and the `res.txt` file from your scratch location to a shared directory. Then skip to Step 3: Import the tablespaces on your target DB instance.
- If you need to stage your backups to an Amazon S3 bucket, complete the following steps.



Step 2.2: Upload the backups to your Amazon S3 bucket

Upload your backups and the `res.txt` file from your scratch directory to your Amazon S3 bucket. For more information, see Uploading objects in the *Amazon Simple Storage Service User Guide*.

Step 2.3: Download the backups from your Amazon S3 bucket to your target DB instance

In this step, you use the procedure `rdsadmin.rdsadmin_s3_tasks.download_from_s3` to download your backups to your RDS for Oracle DB instance.

To download your backups from your Amazon S3 bucket

1. Start SQL*Plus or Oracle SQL Developer and log in to your RDS for Oracle DB instance.
2. Download the backups from the Amazon S3 bucket to your target DB instance by using the Amazon RDS procedure `rdsadmin.rdsadmin_s3_tasks.download_from_s3` to d. The following example downloads all of the files from an Amazon S3 bucket named *`amzn-s3-demo-bucket`* to the *`DATA_PUMP_DIR`* directory.

```
EXEC UTL_FILE.FREMOVE ('DATA_PUMP_DIR', 'res.txt');
SELECT rdsadmin.rdsadmin_s3_tasks.download_from_s3(
  p_bucket_name    =>  'amzn-s3-demo-bucket',
  p_directory_name =>  'DATA_PUMP_DIR')
AS TASK_ID FROM DUAL;
```

   The SELECT statement returns the ID of the task in a VARCHAR2 data type. For more information, see Downloading files from an Amazon S3 bucket to an Oracle DB instance.

Step 3: Import the tablespaces on your target DB instance

To restore your tablespaces to your target DB instance, use the procedure `rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces`. This procedure automatically converts the data files to the correct endian format.

If you import from a platform other than Linux, specify the source platform using the parameter `p_platform_id` when you call `import_xtts_tablespaces`. Make sure that the platform ID that you specify matches the one specified in the `xtt.properties` file in Step 2: Export tablespace metadata on your source host.

Import the tablespaces on your target DB instance

1. Start an Oracle SQL client and log in to your target RDS for Oracle DB instance as the master user.
2. Run the procedure `rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces`, specifying the tablespaces to import and the directory containing the backups.

   The following example imports the tablespaces *TBS1*, *TBS2*, and *TBS3* from the directory *DATA_PUMP_DIR*. The source platform is AIX-Based Systems (64-bit), which has the platform ID of 6. You can find the platform IDs by querying `V$TRANSPORTABLE_PLATFORM`.

   ```
   VAR task_id CLOB

   BEGIN
     :task_id:=rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces(
           'TBS1,TBS2,TBS3',
           'DATA_PUMP_DIR',
           p_platform_id => 6);
   END;
   /

   PRINT task_id
   ```

3. (Optional) Monitor progress by querying the table `rdsadmin.rds_xtts_operation_info`. The `xtts_operation_state` column shows the value `EXECUTING`, `COMPLETED`, or `FAILED`.

   ```
   SELECT * FROM rdsadmin.rds_xtts_operation_info;
   ```

> **Note**
>
> For long-running operations, you can also query V$SESSION_LONGOPS, V$RMAN_STATUS, and V$RMAN_OUTPUT.

4. View the log of the completed import by using the task ID from the previous step.

```
SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||'&task_id'||'.log'));
```

Make sure that the import succeeded before continuing to the next step.

Phase 3: Make and transfer incremental backups

In this phase, you make and transfer incremental backups periodically while the source database is active. This technique reduces the size of your final tablespace backup. If you take multiple incremental backups, you must copy the `res.txt` file after the last incremental backup before you can apply it on the target instance.

The steps are the same as in Phase 2: Prepare the full tablespace backup, except that the import step is optional.

Phase 4: Transport the tablespaces

In this phase, you back up your read-only tablespaces and export Data Pump metadata, transfer these files to your target host, and import both the tablespaces and the metadata.

Topics

- Step 1: Back up your read-only tablespaces
- Step 2: Export tablespace metadata on your source host
- Step 3: (Amazon S3 only) Transfer the backup and export files to your target DB instance
- Step 4: Import the tablespaces on your target DB instance
- Step 5: Import tablespace metadata on your target DB instance

Step 1: Back up your read-only tablespaces

This step is identical to Step 1: Back up the tablespaces on your source host, with one key difference: you place your tablespaces in read-only mode before backing up your tablespaces for the last time.

The following example places `tbs1`, `tbs2`, and `tbs3` in read-only mode.

```
ALTER TABLESPACE tbs1 READ ONLY;
ALTER TABLESPACE tbs2 READ ONLY;
ALTER TABLESPACE tbs3 READ ONLY;
```

Step 2: Export tablespace metadata on your source host

Export your tablespace metadata by running the `expdb` utility on your source host. The following example exports tablespaces *`TBS1`*, *`TBS2`*, and *`TBS3`* to dump file *`xttdump.dmp`* in directory *`DATA_PUMP_DIR`*.

```
expdp username/pwd \
dumpfile=xttdump.dmp \
directory=DATA_PUMP_DIR \
statistics=NONE \
transport_tablespaces=TBS1,TBS2,TBS3 \
transport_full_check=y \
logfile=tts_export.log
```

If *`DATA_PUMP_DIR`* is a shared directory in Amazon EFS, skip to Step 4: Import the tablespaces on your target DB instance.

Step 3: (Amazon S3 only) Transfer the backup and export files to your target DB instance

If you are using Amazon S3 to stage your tablespace backups and Data Pump export file, complete the following steps.

Step 3.1: Upload the backups and dump file from your source host to your Amazon S3 bucket

Upload your backup and dump files from your source host to your Amazon S3 bucket. For more information, see Uploading objects in the *Amazon Simple Storage Service User Guide*.

Step 3.2: Download the backups and dump file from your Amazon S3 bucket to your target DB instance

In this step, you use the procedure `rdsadmin.rdsadmin_s3_tasks.download_from_s3` to download your backups and dump file to your RDS for Oracle DB instance. Follow the steps in Step 2.3: Download the backups from your Amazon S3 bucket to your target DB instance.

Step 4: Import the tablespaces on your target DB instance

Use the procedure `rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces` to restore the tablespaces. For syntax and semantics of this procedure, see Importing transported tablespaces to your DB instance

> **Important**
>
> After you complete your final tablespace import, the next step is importing the Oracle Data Pump metadata. If the import fails, it's important to return your DB instance to its state before the failure. Thus, we recommend that you create a DB snapshot of your DB instance by following the instructions in Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS. The snapshot will contain all imported tablespaces, so if the import fails, you don't need to repeat the backup and import process.
> If your target DB instance has automatic backups turned on, and Amazon RDS doesn't detect that a valid snapshot was initiated before you import the metadata, RDS attempts to create a snapshot. Depending on your instance activity, this snapshot might or might not succeed. If a valid snapshot isn't detected or a snapshot can't be initiated, the metadata import exits with errors.

Import the tablespaces on your target DB instance

1. Start an Oracle SQL client and log in to your target RDS for Oracle DB instance as the master user.
2. Run the procedure `rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces`, specifying the tablespaces to import and the directory containing the backups.

   The following example imports the tablespaces *TBS1*, *TBS2*, and *TBS3* from the directory *DATA_PUMP_DIR*.

   ```
   BEGIN
   ```

```
 :task_id:=rdsadmin.rdsadmin_transport_util.import_xtts_tablespaces('TBS1,TBS2,TBS3','DATA_
END;
/
PRINT task_id
```

3. (Optional) Monitor progress by querying the table `rdsadmin.rds_xtts_operation_info`. The `xtts_operation_state` column shows the value `EXECUTING`, `COMPLETED`, or `FAILED`.

```
SELECT * FROM rdsadmin.rds_xtts_operation_info;
```

> **Note**
>
> For long-running operations, you can also query `V$SESSION_LONGOPS`, `V$RMAN_STATUS`, and `V$RMAN_OUTPUT`.

4. View the log of the completed import by using the task ID from the previous step.

```
SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file('BDUMP',
 'dbtask-'||'&task_id'||'.log'));
```

   Make sure that the import succeeded before continuing to the next step.

5. Take a manual DB snapshot by following the instructions in Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Step 5: Import tablespace metadata on your target DB instance

In this step, you import the transportable tablespace metadata into your RDS for Oracle DB instance using the procedure `rdsadmin.rdsadmin_transport_util.import_xtts_metadata`. For syntax and semantics of this procedure, see Importing transportable tablespace metadata into your DB instance. During the operation, the status of the import is shown in the table `rdsadmin.rds_xtts_operation_info`.

> **Important**
>
> Before you import metadata, we strongly recommend that you confirm that a DB snapshot was successfully created after you imported your tablespaces. If the import step fails, restore your DB instance, address the import errors, and then attempt the import again.

Import the Data Pump metadata into your RDS for Oracle DB instance

1. Start your Oracle SQL client and log in to your target DB instance as the master user.
2. Create the users that own schemas in your transported tablespaces, if these users don't already exist.

```
CREATE USER tbs_owner IDENTIFIED BY password;
```

3. Import the metadata, specifying the name of the dump file and its directory location.

```
BEGIN

 rdsadmin.rdsadmin_transport_util.import_xtts_metadata('xttdump.dmp','DATA_PUMP_DIR');
END;
/
```

4. (Optional) Query the transportable tablespace history table to see the status of the metadata import.

```
SELECT * FROM rdsadmin.rds_xtts_operation_info;
```

   When the operation completes, your tablespaces are in read-only mode.

5. (Optional) View the log file.

   The following example lists the contents of the BDUMP directory and then queries the import log.

```
SELECT * FROM TABLE(rdsadmin.rds_file_util.listdir(p_directory => 'BDUMP'));

SELECT * FROM TABLE(rdsadmin.rds_file_util.read_text_file(
  p_directory => 'BDUMP',
```

```
  p_filename => 'rds-xtts-
import_xtts_metadata-2023-05-22.01-52-35.560858000.log'));
```

Phase 5: Validate the transported tablespaces

In this optional step, you validate your transported tablespaces using the procedure `rdsadmin.rdsadmin_rman_util.validate_tablespace`, and then place your tablespaces in read/write mode.

To validate the transported data

1. Start SQL*Plus or SQL Developer and log in to your target DB instance as the master user.
2. Validate the tablespaces using the procedure `rdsadmin.rdsadmin_rman_util.validate_tablespace`.

```
SET SERVEROUTPUT ON
BEGIN
    rdsadmin.rdsadmin_rman_util.validate_tablespace(
        p_tablespace_name     => 'TBS1',
        p_validation_type     => 'PHYSICAL+LOGICAL',
        p_rman_to_dbms_output => TRUE);
    rdsadmin.rdsadmin_rman_util.validate_tablespace(
        p_tablespace_name     => 'TBS2',
        p_validation_type     => 'PHYSICAL+LOGICAL',
        p_rman_to_dbms_output => TRUE);
    rdsadmin.rdsadmin_rman_util.validate_tablespace(
        p_tablespace_name     => 'TBS3',
        p_validation_type     => 'PHYSICAL+LOGICAL',
        p_rman_to_dbms_output => TRUE);
END;
/
```

3. Place your tablespaces in read/write mode.

```
ALTER TABLESPACE TBS1 READ WRITE;
ALTER TABLESPACE TBS2 READ WRITE;
ALTER TABLESPACE TBS3 READ WRITE;
```

Phase 6: Clean up leftover files

In this optional step, you remove any unneeded files. Use the `rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files` procedure to list data files that were orphaned after a tablespace import, and then use `rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files` procedure to delete them. For syntax and semantics of these procedures, see Listing orphaned files after a tablespace import and Deleting orphaned data files after a tablespace import.

To clean up leftover files

1. Remove old backups in *DATA_PUMP_DIR* as follows:

   a. List the backup files by running `rdsadmin.rdsadmin_file_util.listdir`.

   ```
   SELECT * FROM TABLE(rdsadmin.rds_file_util.listdir(p_directory =>
    'DATA_PUMP_DIR'));
   ```

   b. Remove the backups one by one by calling `UTL_FILE.FREMOVE`.

   ```
   EXEC UTL_FILE.FREMOVE ('DATA_PUMP_DIR', 'backup_filename');
   ```

2. If you imported tablespaces but didn't import metadata for these tablespaces, you can delete the orphaned data files as follows:

   a. List the orphaned data files that you need to delete. The following example runs the procedure `rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files`.

   ```
   SQL> SELECT * FROM
    TABLE(rdsadmin.rdsadmin_transport_util.list_xtts_orphan_files);

   FILENAME       FILESIZE
   -------------- ---------
   datafile_7.dbf 104865792
   datafile_8.dbf 104865792
   ```

   b. Delete the orphaned files by running the procedure `rdsadmin.rdsadmin_transport_util.cleanup_incomplete_xtts_import`.

   ```
   BEGIN
   ```

```
 rdsadmin.rdsadmin_transport_util.cleanup_incomplete_xtts_import('DATA_PUMP_DIR');
END;
/
```

The cleanup operation generates a log file that uses the name format `rds-xtts-delete_xtts_orphaned_files-YYYY-MM-DD.HH24-MI-SS.FF.log` in the BDUMP directory.

c. Read the log file generated in the previous step. The following example reads log `rds-xtts-delete_xtts_orphaned_files-2023-06-01.09-33-11.868894000.log`.

```
SELECT *
FROM TABLE(rdsadmin.rds_file_util.read_text_file(
       p_directory => 'BDUMP',
       p_filename  => 'rds-xtts-
delete_xtts_orphaned_files-2023-06-01.09-33-11.868894000.log'));

TEXT
--------------------------------------------------------------------------------
orphan transported datafile datafile_7.dbf deleted.
orphan transported datafile datafile_8.dbf deleted.
```

3. If you imported tablespaces and imported metadata for these tablespaces, but you encountered compatibility errors or other Oracle Data Pump issues, clean up the partially transported data files as follows:

a. List the tablespaces that contain partially transported data files by querying DBA_TABLESPACES.

```
SQL> SELECT TABLESPACE_NAME FROM DBA_TABLESPACES WHERE PLUGGED_IN='YES';

TABLESPACE_NAME
--------------------------------------------------------------------------------
TBS_3
```

b. Drop the tablespaces and the partially transported data files.

```
DROP TABLESPACE TBS_3 INCLUDING CONTENTS AND DATAFILES;
```

Importing using Oracle Data Pump

Oracle Data Pump is a utility that allows you to export Oracle data to a dump file and import it into another Oracle database. It is a long-term replacement for the Oracle Export/Import utilities. Oracle Data Pump is the recommended way to move large amounts of data from an Oracle database to an Amazon RDS DB instance.

The examples in this section show one way to import data into an Oracle database, but Oracle Data Pump supports other techniques. For more information, see the Oracle Database documentation.

The examples in this section use the `DBMS_DATAPUMP` package. You can accomplish the same tasks using the Oracle Data Pump command line utilities `impdp` and `expdp`. You can install these utilities on a remote host as part of an Oracle Client installation, including Oracle Instant Client. For more information, see How do I use Oracle Instant Client to run Data Pump Import or Export for my Amazon RDS for Oracle DB instance?

Topics

- Overview of Oracle Data Pump
- Importing data with Oracle Data Pump and an Amazon S3 bucket
- Importing data with Oracle Data Pump and a database link

Overview of Oracle Data Pump

Oracle Data Pump is made up of the following components:

- Command-line clients `expdp` and `impdp`
- The `DBMS_DATAPUMP` PL/SQL package
- The `DBMS_METADATA` PL/SQL package

You can use Oracle Data Pump for the following scenarios:

- Import data from an Oracle database, either on-premises or on an Amazon EC2 instance, to an RDS for Oracle DB instance.
- Import data from an RDS for Oracle DB instance to an Oracle database, either on-premises or on an Amazon EC2 instance.
- Import data between RDS for Oracle DB instances, for example, to migrate data from EC2-Classic to VPC.

To download Oracle Data Pump utilities, see Oracle database software downloads on the Oracle Technology Network website. For compatibility considerations when migrating between versions of Oracle Database, see the Oracle Database documentation.

Oracle Data Pump workflow

Typically, you use Oracle Data Pump in the following stages:

1. Export your data into a dump file on the source database.
2. Upload your dump file to your destination RDS for Oracle DB instance. You can transfer using an Amazon S3 bucket or by using a database link between the two databases.
3. Import the data from your dump file into your RDS for Oracle DB instance.

Oracle Data Pump best practices

When you use Oracle Data Pump to import data into an RDS for Oracle instance, we recommend the following best practices:

- Perform imports in `schema` or `table` mode to import specific schemas and objects.
- Limit the schemas you import to those required by your application.
- Don't import in `full` mode or import schemas for system-maintained components.

  Because RDS for Oracle doesn't allow access to SYS or SYSDBA administrative users, these actions might damage the Oracle data dictionary and affect the stability of your database.
- When loading large amounts of data, do the following:
  1. Transfer the dump file to the target RDS for Oracle DB instance.
  2. Take a DB snapshot of your instance.
  3. Test the import to verify that it succeeds.

  If database components are invalidated, you can delete the DB instance and re-create it from the DB snapshot. The restored DB instance includes any dump files staged on the DB instance when you took the DB snapshot.
- Don't import dump files that were created using the Oracle Data Pump export parameters `TRANSPORT_TABLESPACES`, `TRANSPORTABLE`, or `TRANSPORT_FULL_CHECK`. RDS for Oracle DB instances don't support importing these dump files.
- Don't import dump files that contain Oracle Scheduler objects in SYS, `SYSTEM`, `RDSADMIN`, `RDSSEC`, and `RDS_DATAGUARD`, and belong to the following categories:

- Jobs
- Programs
- Schedules
- Chains
- Rules
- Evaluation contexts
- Rule sets

RDS for Oracle DB instances don't support importing these dump files.

- To exclude unsupported Oracle Scheduler objects, use additional directives during the Data Pump export. If you use DBMS_DATAPUMP, you can add an additional METADATA_FILTER before the DBMS_METADATA.START_JOB:

```
DBMS_DATAPUMP.METADATA_FILTER(
  v_hdnl,
  'EXCLUDE_NAME_EXPR',
  q'[IN (SELECT NAME FROM SYS.OBJ$
         WHERE TYPE# IN (66,67,74,79,59,62,46)
         AND OWNER# IN
           (SELECT USER# FROM SYS.USER$
            WHERE NAME IN ('RDSADMIN','SYS','SYSTEM','RDS_DATAGUARD','RDSSEC')
            )
        )
  ]',
  'PROCOBJ'
);
```

If you use expdp, create a parameter file that contains the exclude directive shown in the following example. Then use PARFILE=*parameter_file* with your expdp command.

```
exclude=procobj:"IN
  (SELECT NAME FROM sys.OBJ$
   WHERE TYPE# IN (66,67,74,79,59,62,46)
   AND OWNER# IN
     (SELECT USER# FROM SYS.USER$
      WHERE NAME IN ('RDSADMIN','SYS','SYSTEM','RDS_DATAGUARD','RDSSEC')
     )
  )"
```

Importing data with Oracle Data Pump and an Amazon S3 bucket

The following import process uses Oracle Data Pump and an Amazon S3 bucket. The steps are as follows:

1. Export data on the source database using the Oracle DBMS_DATAPUMP package.
2. Place the dump file in an Amazon S3 bucket.
3. Download the dump file from the Amazon S3 bucket to the `DATA_PUMP_DIR` directory on the target RDS for Oracle DB instance.
4. Import the data from the copied dump file into the RDS for Oracle DB instance using the package `DBMS_DATAPUMP`.

Topics

- Requirements for Importing data with Oracle Data Pump and an Amazon S3 bucket
- Step 1: Grant privileges to the database user on the RDS for Oracle target DB instance
- Step 2: Export data into a dump file using DBMS_DATAPUMP
- Step 3: Upload the dump file to your Amazon S3 bucket
- Step 4: Download the dump file from your Amazon S3 bucket to your target DB instance
- Step 5: Import your dump file into your target DB instance using DBMS_DATAPUMP
- Step 6: Clean up

Requirements for Importing data with Oracle Data Pump and an Amazon S3 bucket

The process has the following requirements:

- Make sure that an Amazon S3 bucket is available for file transfers, and that the Amazon S3 bucket is in the same AWS Region as the DB instance. For instructions, see Create a bucket in the *Amazon Simple Storage Service Getting Started Guide*.
- The object that you upload into the Amazon S3 bucket must be 5 TB or less. For more information about working with objects in Amazon S3, see Amazon Simple Storage Service User Guide.

> **Note**
>
> If you dump file exceeds 5 TB, you can run the Oracle Data Pump export with the parallel option. This operation spreads the data into multiple dump files so that you do not exceed the 5 TB limit for individual files.

- You must prepare the Amazon S3 bucket for Amazon RDS integration by following the instructions in Configuring IAM permissions for RDS for Oracle integration with Amazon S3.
- You must ensure that you have enough storage space to store the dump file on the source instance and the target DB instance.

> **Note**
>
> This process imports a dump file into the `DATA_PUMP_DIR` directory, a preconfigured directory on all Oracle DB instances. This directory is located on the same storage volume as your data files. When you import the dump file, the existing Oracle data files use more space. Thus, you should make sure that your DB instance can accommodate that additional use of space. The imported dump file is not automatically deleted or purged from the `DATA_PUMP_DIR` directory. To remove the imported dump file, use UTL_FILE.FREMOVE, found on the Oracle website.

Step 1: Grant privileges to the database user on the RDS for Oracle target DB instance

In this step, you create the schemas into which you plan to import data and grant the users necessary privileges.

To create users and grant necessary privileges on the RDS for Oracle target instance

1. Use SQL*Plus or Oracle SQL Developer to log in as the master user to the RDS for Oracle DB instance into which the data will be imported. For information about connecting to a DB instance, see Connecting to your Oracle DB instance.
2. Create the required tablespaces before you import the data. For more information, see Creating and sizing tablespaces in RDS for Oracle.

3. Create the user account and grant the necessary permissions and roles if the user account into which the data is imported doesn't exist. If you plan to import data into multiple user schemas, create each user account and grant the necessary privileges and roles to it.

   For example, the following SQL statements create a new user and grant the necessary permissions and roles to import the data into the schema owned by this user. Replace *schema_1* with the name of your schema in this step and in the following steps.

   ```
   CREATE USER schema_1 IDENTIFIED BY my_password;
   GRANT CREATE SESSION, RESOURCE TO schema_1;
   ALTER USER schema_1 QUOTA 100M ON users;
   ```

   > **Note**
   >
   > Specify a password other than the prompt shown here as a security best practice.

   The preceding statements grant the new user the `CREATE SESSION` privilege and the `RESOURCE` role. You might need additional privileges and roles depending on the database objects that you import.

Step 2: Export data into a dump file using DBMS_DATAPUMP

To create a dump file, use the `DBMS_DATAPUMP` package.

To export Oracle data into a dump file

1. Use SQL Plus or Oracle SQL Developer to connect to the source RDS for Oracle DB instance with an administrative user. If the source database is an RDS for Oracle DB instance, connect with the Amazon RDS master user.
2. Export the data by calling `DBMS_DATAPUMP` procedures.

   The following script exports the *SCHEMA_1* schema into a dump file named `sample.dmp` in the `DATA_PUMP_DIR` directory. Replace *SCHEMA_1* with the name of the schema that you want to export.

   ```
   DECLARE
     v_hdnl NUMBER;
   ```

```
BEGIN
  v_hdnl := DBMS_DATAPUMP.OPEN(
    operation => 'EXPORT',
    job_mode  => 'SCHEMA',
    job_name  => null
  );
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl         ,
    filename  => 'sample.dmp'   ,
    directory => 'DATA_PUMP_DIR',
    filetype  => dbms_datapump.ku$_file_type_dump_file
  );
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'sample_exp.log',
    directory => 'DATA_PUMP_DIR' ,
    filetype  => dbms_datapump.ku$_file_type_log_file
  );
  DBMS_DATAPUMP.METADATA_FILTER(v_hdnl,'SCHEMA_EXPR','IN (''SCHEMA_1'')');
  DBMS_DATAPUMP.METADATA_FILTER(
    v_hdnl,
    'EXCLUDE_NAME_EXPR',
    q'[IN (SELECT NAME FROM SYS.OBJ$
           WHERE TYPE# IN (66,67,74,79,59,62,46)
           AND OWNER# IN
             (SELECT USER# FROM SYS.USER$
              WHERE NAME IN ('RDSADMIN','SYS','SYSTEM','RDS_DATAGUARD','RDSSEC')
             )
          )
    ]',
    'PROCOBJ'
  );
  DBMS_DATAPUMP.START_JOB(v_hdnl);
END;
/
```

Note

Data Pump starts jobs asynchronously. For information about monitoring a Data Pump job, see Monitoring job status in the Oracle documentation.

3. (Optional) View the contents of the export log by calling the `rdsadmin.rds_file_util.read_text_file` procedure. For more information, see Reading files in a DB instance directory.

Step 3: Upload the dump file to your Amazon S3 bucket

Use the Amazon RDS procedure `rdsadmin.rdsadmin_s3_tasks.upload_to_s3` to copy the dump file to the Amazon S3 bucket. The following example uploads all of the files from the `DATA_PUMP_DIR` directory to an Amazon S3 bucket named *amzn-s3-demo-bucket*.

```
SELECT rdsadmin.rdsadmin_s3_tasks.upload_to_s3(
  p_bucket_name    =>  'amzn-s3-demo-bucket',
  p_directory_name =>  'DATA_PUMP_DIR')
AS TASK_ID FROM DUAL;
```

The SELECT statement returns the ID of the task in a VARCHAR2 data type. For more information, see Uploading files from your RDS for Oracle DB instance to an Amazon S3 bucket.

Step 4: Download the dump file from your Amazon S3 bucket to your target DB instance

Perform this step using the Amazon RDS procedure `rdsadmin.rdsadmin_s3_tasks.download_from_s3`. When you download a file to a directory, the procedure `download_from_s3` skips the download if an identically named file already exists in the directory. To remove a file from the download directory, use UTL_FILE.FREMOVE, found on the Oracle website.

To download your dump file

1. Start SQL*Plus or Oracle SQL Developer and log in as the master on your Amazon RDS target Oracle DB instance
2. Download the dump file using the Amazon RDS procedure `rdsadmin.rdsadmin_s3_tasks.download_from_s3`.

   The following example downloads all files from an Amazon S3 bucket named *amzn-s3-demo-bucket* to the directory `DATA_PUMP_DIR`.

   ```
   SELECT rdsadmin.rdsadmin_s3_tasks.download_from_s3(
     p_bucket_name    =>  'amzn-s3-demo-bucket',
     p_directory_name =>  'DATA_PUMP_DIR')
   ```

```
AS TASK_ID FROM DUAL;
```

The SELECT statement returns the ID of the task in a VARCHAR2 data type. For more information, see Downloading files from an Amazon S3 bucket to an Oracle DB instance.

Step 5: Import your dump file into your target DB instance using DBMS_DATAPUMP

Use DBMS_DATAPUMP to import the schema into your RDS for Oracle DB instance. Additional options such as METADATA_REMAP might be required.

To import data into your target DB instance

1. Start SQL*Plus or SQL Developer and log in as the master user to your RDS for Oracle DB instance.
2. Import the data by calling DBMS_DATAPUMP procedures.

   The following example imports the *SCHEMA_1* data from `sample_copied.dmp` into your target DB instance.

```
DECLARE
  v_hdnl NUMBER;
BEGIN
  v_hdnl := DBMS_DATAPUMP.OPEN(
    operation => 'IMPORT',
    job_mode  => 'SCHEMA',
    job_name  => null);
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'sample_copied.dmp',
    directory => 'DATA_PUMP_DIR',
    filetype  => dbms_datapump.ku$_file_type_dump_file);
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'sample_imp.log',
    directory => 'DATA_PUMP_DIR',
    filetype  => dbms_datapump.ku$_file_type_log_file);
  DBMS_DATAPUMP.METADATA_FILTER(v_hdnl,'SCHEMA_EXPR','IN (''SCHEMA_1'')');
  DBMS_DATAPUMP.START_JOB(v_hdnl);
END;
/
```

> **Note**
>
> Data Pump jobs are started asynchronously. For information about monitoring a Data Pump job, see Monitoring job status in the Oracle documentation. You can view the contents of the import log by using the `rdsadmin.rds_file_util.read_text_file` procedure. For more information, see Reading files in a DB instance directory.

3. Verify the data import by listing the schema tables on your target DB instance.

   For example, the following query returns the number of tables for *SCHEMA_1*.

```
SELECT COUNT(*) FROM DBA_TABLES WHERE OWNER='SCHEMA_1';
```

Step 6: Clean up

After the data has been imported, you can delete the files that you don't want to keep.

To remove unneeded files

1. Start SQL*Plus or SQL Developer and log in as the master user to your RDS for Oracle DB instance.
2. List the files in `DATA_PUMP_DIR` using the following command.

```
SELECT * FROM TABLE(rdsadmin.rds_file_util.listdir('DATA_PUMP_DIR')) ORDER BY
 MTIME;
```

3. Delete files in `DATA_PUMP_DIR` that you no longer require, use the following command.

```
EXEC UTL_FILE.FREMOVE('DATA_PUMP_DIR','filename');
```

   For example, the following command deletes the file named `sample_copied.dmp`.

```
EXEC UTL_FILE.FREMOVE('DATA_PUMP_DIR','sample_copied.dmp');
```

Importing data with Oracle Data Pump and a database link

The following import process uses Oracle Data Pump and the Oracle DBMS_FILE_TRANSFER package. The steps are as follows:

1. Connect to a source Oracle database, which can be an on-premises database, Amazon EC2 instance, or an RDS for Oracle DB instance.
2. Export data using the DBMS_DATAPUMP package.
3. Use `DBMS_FILE_TRANSFER.PUT_FILE` to copy the dump file from the Oracle database to the `DATA_PUMP_DIR` directory on the target RDS for Oracle DB instance that is connected using a database link.
4. Import the data from the copied dump file into the RDS for Oracle DB instance using the `DBMS_DATAPUMP` package.

The import process using Oracle Data Pump and the `DBMS_FILE_TRANSFER` package has the following steps.

Topics

- Requirements for importing data with Oracle Data Pump and a database link
- Step 1: Grant privileges to the user on the RDS for Oracle target DB instance
- Step 2: Grant privileges to the user on the source database
- Step 3: Create a dump file using DBMS_DATAPUMP
- Step 4: Create a database link to the target DB instance
- Step 5: Copy the exported dump file to the target DB instance using DBMS_FILE_TRANSFER
- Step 6: Import the data file to the target DB instance using DBMS_DATAPUMP
- Step 7: Clean up

Requirements for importing data with Oracle Data Pump and a database link

The process has the following requirements:

- You must have execute privileges on the `DBMS_FILE_TRANSFER` and `DBMS_DATAPUMP` packages.
- You must have write privileges to the `DATA_PUMP_DIR` directory on the source DB instance.
- You must ensure that you have enough storage space to store the dump file on the source instance and the target DB instance.

> **Note**
>
> This process imports a dump file into the DATA_PUMP_DIR directory, a preconfigured directory on all Oracle DB instances. This directory is located on the same storage volume as your data files. When you import the dump file, the existing Oracle data files use more space. Thus, you should make sure that your DB instance can accommodate that additional use of space. The imported dump file is not automatically deleted or purged from the DATA_PUMP_DIR directory. To remove the imported dump file, use UTL_FILE.FREMOVE, found on the Oracle website.

Step 1: Grant privileges to the user on the RDS for Oracle target DB instance

To grant privileges to the user on the RDS for Oracle target DB instance, take the following steps:

1. Use SQL Plus or Oracle SQL Developer to connect to the RDS for Oracle DB instance into which you intend to import the data. Connect as the Amazon RDS master user. For information about connecting to the DB instance, see Connecting to your Oracle DB instance.
2. Create the required tablespaces before you import the data. For more information, see Creating and sizing tablespaces in RDS for Oracle.
3. If the user account into which the data is imported doesn't exist, create the user account and grant the necessary permissions and roles. If you plan to import data into multiple user schemas, create each user account and grant the necessary privileges and roles to it.

   For example, the following commands create a new user named *schema_1* and grant the necessary permissions and roles to import the data into the schema for this user.

   ```
   CREATE USER schema_1 IDENTIFIED BY my-password;
   GRANT CREATE SESSION, RESOURCE TO schema_1;
   ALTER USER schema_1 QUOTA 100M ON users;
   ```

   > **Note**
   >
   > Specify a password other than the prompt shown here as a security best practice.

The preceding example grants the new user the `CREATE SESSION` privilege and the `RESOURCE` role. Additional privileges and roles might be required depending on the database objects that you import.

> **Note**
>
> Replace *schema_1* with the name of your schema in this step and in the following steps.

Step 2: Grant privileges to the user on the source database

Use SQL*Plus or Oracle SQL Developer to connect to the RDS for Oracle DB instance that contains the data to be imported. If necessary, create a user account and grant the necessary permissions.

> **Note**
>
> If the source database is an Amazon RDS instance, you can skip this step. You use your Amazon RDS master user account to perform the export.

The following commands create a new user and grant the necessary permissions.

```
CREATE USER export_user IDENTIFIED BY my-password;
GRANT CREATE SESSION, CREATE TABLE, CREATE DATABASE LINK TO export_user;
ALTER USER export_user QUOTA 100M ON users;
GRANT READ, WRITE ON DIRECTORY data_pump_dir TO export_user;
GRANT SELECT_CATALOG_ROLE TO export_user;
GRANT EXECUTE ON DBMS_DATAPUMP TO export_user;
GRANT EXECUTE ON DBMS_FILE_TRANSFER TO export_user;
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Step 3: Create a dump file using DBMS_DATAPUMP

To create a dump file, do the following:

1. Use SQL*Plus or Oracle SQL Developer to connect to the source Oracle instance with an administrative user or with the user you created in step 2. If the source database is an Amazon RDS for Oracle DB instance, connect with the Amazon RDS master user.
2. Create a dump file using the Oracle Data Pump utility.

   The following script creates a dump file named *sample.dmp* in the DATA_PUMP_DIR directory.

```
DECLARE
  v_hdnl NUMBER;
BEGIN
  v_hdnl := DBMS_DATAPUMP.OPEN(
    operation => 'EXPORT' ,
    job_mode  => 'SCHEMA' ,
    job_name  => null
  );
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'sample.dmp'    ,
    directory => 'DATA_PUMP_DIR' ,
    filetype  => dbms_datapump.ku$_file_type_dump_file
  );
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl           ,
    filename  => 'sample_exp.log' ,
    directory => 'DATA_PUMP_DIR'  ,
    filetype  => dbms_datapump.ku$_file_type_log_file
  );
  DBMS_DATAPUMP.METADATA_FILTER(
    v_hdnl              ,
    'SCHEMA_EXPR'       ,
    'IN (''SCHEMA_1'')'
  );
  DBMS_DATAPUMP.METADATA_FILTER(
    v_hdnl,
    'EXCLUDE_NAME_EXPR',
    q'[IN (SELECT NAME FROM sys.OBJ$
           WHERE TYPE# IN (66,67,74,79,59,62,46)
           AND OWNER# IN
             (SELECT USER# FROM SYS.USER$
              WHERE NAME IN ('RDSADMIN','SYS','SYSTEM','RDS_DATAGUARD','RDSSEC')
             )
         )
    ]',
```

```
    'PROCOBJ'
  );
  DBMS_DATAPUMP.START_JOB(v_hdnl);
END;
/
```

> **Note**
>
> Data Pump jobs are started asynchronously. For information about monitoring a Data Pump job, see Monitoring job status in the Oracle documentation. You can view the contents of the export log by using the `rdsadmin.rds_file_util.read_text_file` procedure. For more information, see Reading files in a DB instance directory.

Step 4: Create a database link to the target DB instance

Create a database link between your source DB instance and your target DB instance. Your local Oracle instance must have network connectivity to the DB instance in order to create a database link and to transfer your export dump file.

Perform this step connected with the same user account as the previous step.

If you are creating a database link between two DB instances inside the same VPC or peered VPCs, the two DB instances should have a valid route between them. The security group of each DB instance must allow ingress to and egress from the other DB instance. The security group inbound and outbound rules can refer to security groups from the same VPC or a peered VPC. For more information, see Adjusting database links for use with DB instances in a VPC.

The following command creates a database link named `to_rds` that connects to the Amazon RDS master user at the target DB instance.

```
CREATE DATABASE LINK to_rds
  CONNECT TO <master_user_account> IDENTIFIED BY <password>
  USING '(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=<dns or ip address of remote db>)
         (PORT=<listener port>))(CONNECT_DATA=(SID=<remote SID>)))';
```

Step 5: Copy the exported dump file to the target DB instance using DBMS_FILE_TRANSFER

Use DBMS_FILE_TRANSFER to copy the dump file from the source database instance to the target DB instance. The following script copies a dump file named sample.dmp from the source instance to a target database link named *to_rds* (created in the previous step).

```
BEGIN
  DBMS_FILE_TRANSFER.PUT_FILE(
    source_directory_object       => 'DATA_PUMP_DIR',
    source_file_name              => 'sample.dmp',
    destination_directory_object  => 'DATA_PUMP_DIR',
    destination_file_name         => 'sample_copied.dmp',
    destination_database          => 'to_rds' );
END;
/
```

Step 6: Import the data file to the target DB instance using DBMS_DATAPUMP

Use Oracle Data Pump to import the schema in the DB instance. Additional options such as METADATA_REMAP might be required.

Connect to the DB instance with the Amazon RDS master user account to perform the import.

```
DECLARE
  v_hdnl NUMBER;
BEGIN
  v_hdnl := DBMS_DATAPUMP.OPEN(
    operation => 'IMPORT',
    job_mode  => 'SCHEMA',
    job_name  => null);
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'sample_copied.dmp',
    directory => 'DATA_PUMP_DIR',
    filetype  => dbms_datapump.ku$_file_type_dump_file );
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'sample_imp.log',
    directory => 'DATA_PUMP_DIR',
    filetype  => dbms_datapump.ku$_file_type_log_file);
  DBMS_DATAPUMP.METADATA_FILTER(v_hdnl,'SCHEMA_EXPR','IN (''SCHEMA_1'')');
  DBMS_DATAPUMP.START_JOB(v_hdnl);
```

```
END;
/
```

> **Note**
>
> Data Pump jobs are started asynchronously. For information about monitoring a Data Pump job, see Monitoring job status in the Oracle documentation. You can view the contents of the import log by using the `rdsadmin.rds_file_util.read_text_file` procedure. For more information, see Reading files in a DB instance directory.

You can verify the data import by viewing the user's tables on the DB instance. For example, the following query returns the number of tables for *schema_1*.

```
SELECT COUNT(*) FROM DBA_TABLES WHERE OWNER='SCHEMA_1';
```

Step 7: Clean up

After the data has been imported, you can delete the files that you don't want to keep. You can list the files in `DATA_PUMP_DIR` using the following command.

```
SELECT * FROM TABLE(rdsadmin.rds_file_util.listdir('DATA_PUMP_DIR')) ORDER BY MTIME;
```

To delete files in `DATA_PUMP_DIR` that you no longer require, use the following command.

```
EXEC UTL_FILE.FREMOVE('DATA_PUMP_DIR','<file name>');
```

For example, the following command deletes the file named `"sample_copied.dmp"`.

```
EXEC UTL_FILE.FREMOVE('DATA_PUMP_DIR','sample_copied.dmp');
```

Importing using Oracle Export/Import

You might consider Oracle Export/Import utilities for migrations in the following conditions:

- Your data size is small.
- Data types such as binary float and double aren't required.

The import process creates the necessary schema objects. Thus, you don't need to run a script to create the objects beforehand.

The easiest way to install the Oracle the export and import utilities is to install the Oracle Instant Client. To download the software, go to https://www.oracle.com/database/technologies/instant-client.html. For documentation, see Instant Client for SQL*Loader, Export, and Import in the *Oracle Database Utilities* manual.

To export tables and then import them

1. Export the tables from the source database using the `exp` command.

   The following command exports the tables named `tab1`, `tab2`, and `tab3`. The dump file is `exp_file.dmp`.

   ```
   exp cust_dba@ORCL FILE=exp_file.dmp TABLES=(tab1,tab2,tab3) LOG=exp_file.log
   ```

   The export creates a binary dump file that contains both the schema and data for the specified tables.
2. Import the schema and data into a target database using the `imp` command.

   The following command imports the tables `tab1`, `tab2`, and `tab3` from dump file `exp_file.dmp`.

   ```
   imp cust_dba@targetdb FROMUSER=cust_schema TOUSER=cust_schema \
   TABLES=(tab1,tab2,tab3) FILE=exp_file.dmp LOG=imp_file.log
   ```

Export and Import have other variations that might be better suited to your requirements. See the Oracle Database documentation for full details.

Importing using Oracle SQL*Loader

You might consider Oracle SQL*Loader for large databases that contain a limited number of objects. Because the process of exporting from a source database and loading to a target database is specific to the schema, the following example creates the sample schema objects, exports from a source, and then loads the data into a target database.

The easiest way to install Oracle SQL*Loader is to install the Oracle Instant Client. To download the software, go to https://www.oracle.com/database/technologies/instant-client.html. For

documentation, see Instant Client for SQL*Loader, Export, and Import in the *Oracle Database Utilities* manual.

To import data using Oracle SQL*Loader

1. Create a sample source table using the following SQL statement.

```
CREATE TABLE customer_0 TABLESPACE users
   AS (SELECT ROWNUM id, o.*
       FROM   ALL_OBJECTS o, ALL_OBJECTS x
       WHERE  ROWNUM <= 1000000);
```

2. On the target RDS for Oracle DB instance, create a destination table for loading the data. The clause `WHERE 1=2` ensures that you copy the structure of `ALL_OBJECTS`, but don't copy any rows.

```
CREATE TABLE customer_1 TABLESPACE users
  AS (SELECT 0 AS ID, OWNER, OBJECT_NAME, CREATED
      FROM   ALL_OBJECTS
      WHERE  1=2);
```

3. Export the data from the source database to a text file. The following example uses SQL*Plus. For your data, you will likely need to generate a script that does the export for all the objects in the database.

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY/MM/DD HH24:MI:SS'

SET LINESIZE 800 HEADING OFF FEEDBACK OFF ARRAY 5000 PAGESIZE 0
SPOOL customer_0.out
SET MARKUP HTML PREFORMAT ON
SET COLSEP ','

SELECT id, owner, object_name, created
FROM   customer_0;

SPOOL OFF
```

4. Create a control file to describe the data. You might need to write a script to perform this step.

```
cat << EOF > sqlldr_1.ctl
load data
infile customer_0.out
```

```
into table customer_1
APPEND
fields terminated by "," optionally enclosed by '"'
(
  id           POSITION(01:10)    INTEGER EXTERNAL,
  owner        POSITION(12:41)    CHAR,
  object_name  POSITION(43:72)    CHAR,
  created      POSITION(74:92)    date "YYYY/MM/DD HH24:MI:SS"
)
```

If needed, copy the files generated by the preceding code to a staging area, such as an Amazon EC2 instance.

5. Import the data using SQL*Loader with the appropriate user name and password for the target database.

```
sqlldr cust_dba@targetdb CONTROL=sqlldr_1.ctl BINDSIZE=10485760 READSIZE=10485760
 ROWS=1000
```

Migrating with Oracle materialized views

To migrate large datasets efficiently, you can use Oracle materialized view replication. With replication, you can keep the target tables synchronized with the source tables. Thus, you can switch over to Amazon RDS later, if needed.

Before you can migrate using materialized views, make sure that you meet the following requirements:

- Configure access from the target database to the source database. In the following example, access rules were enabled on the source database to allow the RDS for Oracle target database to connect to the source over SQL*Net.
- Create a database link from the RDS for Oracle DB instance to the source database.

To migrate data using materialized views

1. Create a user account on both source and RDS for Oracle target instances that can authenticate with the same password. The following example creates a user named `dblink_user`.

```
CREATE USER dblink_user IDENTIFIED BY my-password
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp;

GRANT CREATE SESSION TO dblink_user;

GRANT SELECT ANY TABLE TO dblink_user;

GRANT SELECT ANY DICTIONARY TO dblink_user;
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

2. Create a database link from the RDS for Oracle target instance to the source instance using your newly created user.

```
CREATE DATABASE LINK remote_site
  CONNECT TO dblink_user IDENTIFIED BY my-password
  USING '(description=(address=(protocol=tcp) (host=my-host)
    (port=my-listener-port)) (connect_data=(sid=my-source-db-sid)))';
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

3. Test the link:

```
SELECT * FROM V$INSTANCE@remote_site;
```

4. Create a sample table with primary key and materialized view log on the source instance.

```
CREATE TABLE customer_0 TABLESPACE users
  AS (SELECT ROWNUM id, o.*
      FROM   ALL_OBJECTS o, ALL_OBJECTS x
      WHERE  ROWNUM <= 1000000);

ALTER TABLE customer_0 ADD CONSTRAINT pk_customer_0 PRIMARY KEY (id) USING INDEX;
```

```
CREATE MATERIALIZED VIEW LOG ON customer_0;
```

5. On the target RDS for Oracle DB instance, create a materialized view.

```
CREATE MATERIALIZED VIEW customer_0
  BUILD IMMEDIATE REFRESH FAST
  AS (SELECT *
      FROM   cust_dba.customer_0@remote_site);
```

6. On the target RDS for Oracle DB instance, refresh the materialized view.

```
EXEC DBMS_MVIEW.REFRESH('CUSTOMER_0', 'f');
```

7. Drop the materialized view and include the `PRESERVE TABLE` clause to retain the materialized view container table and its contents.

```
DROP MATERIALIZED VIEW customer_0 PRESERVE TABLE;
```

   The retained table has the same name as the dropped materialized view.

Working with read replicas for Amazon RDS for Oracle

To configure replication between Oracle DB instances, you can create replica databases. For an overview of Amazon RDS read replicas, see Overview of Amazon RDS read replicas. For a summary of the differences between Oracle replicas and other DB engines, see Differences between read replicas for DB engines.

Topics

- Overview of RDS for Oracle replicas
- Requirements and considerations for RDS for Oracle replicas
- Preparing to create an Oracle replica
- Creating an RDS for Oracle replica in mounted mode
- Modifying the RDS for Oracle replica mode
- Working with RDS for Oracle replica backups
- Performing an Oracle Data Guard switchover
- Troubleshooting RDS for Oracle replicas
- Redo transport compression with RDS for Oracle

Overview of RDS for Oracle replicas

An *Oracle replica* database is a physical copy of your primary database. An Oracle replica in read-only mode is called a *read replica*. An Oracle replica in mounted mode is called a *mounted replica*. Oracle Database doesn't permit writes in a replica, but you can promote a replica to make it writable. The promoted read replica has the replicated data to the point when the request was made to promote it.

The following video provides a helpful overview of RDS for Oracle disaster recovery.

For more information, see the blog post Managed disaster recovery with Amazon RDS for Oracle cross-Region automated backups - Part 1 and Managed disaster recovery with Amazon RDS for Oracle cross-Region automated backups - Part 2.

Topics

- Read-only and mounted replicas
- Read replicas of CDBs

- Archived redo log retention
- Outages during Oracle replication

Read-only and mounted replicas

When creating or modifying an Oracle replica, you can place it in either of the following modes:

Read-only

This is the default. Active Data Guard transmits and applies changes from the source database to all read replica databases.

You can create up to five read replicas from one source DB instance. For general information about read replicas that applies to all DB engines, see Working with DB instance read replicas. For information about Oracle Data Guard, see Oracle Data Guard concepts and administration in the Oracle documentation.

Mounted

In this case, replication uses Oracle Data Guard, but the replica database doesn't accept user connections. The primary use for mounted replicas is cross-Region disaster recovery.

A mounted replica can't serve a read-only workload. The mounted replica deletes archived redo log files after it applies them, regardless of the archived log retention policy.

You can create a combination of mounted and read-only DB replicas for the same source DB instance. You can change a read-only replica to mounted mode, or change a mounted replica to read-only mode. In either case, the Oracle database preserves the archived log retention setting.

Read replicas of CDBs

RDS for Oracle supports Data Guard read replicas for Oracle Database 19c and 21c CDBs in both single-tenant and multi-tenant configurations. You can create, manage, and promote read replicas in a CDB just as you can in a non-CDB. Mounted replicas are also supported. You get the following benefits:

- Managed disaster recovery, high availability, and read-only access to your replicas
- The ability to create read replicas in a different AWS Region.
- Integration with the existing RDS read replica APIs: CreateDBInstanceReadReplica, PromoteReadReplica, and SwitchoverReadReplica

To use this feature, you need an Active Data Guard license and an Oracle Database Enterprise Edition license for both the replica and primary DB instances. There are no additional costs related to using CDB architecture. You pay only for your DB instances.

For more information about the single-tenant and multi-tenant configurations of the CDB architecture, see Overview of RDS for Oracle CDBs.

Archived redo log retention

If a primary DB instance has no cross-Region read replicas, Amazon RDS for Oracle keeps a minimum of two hours of archived redo logs on the source DB instance. This is true regardless of the setting for `archivelog retention hours` in `rdsadmin.rdsadmin_util.set_configuration`.

RDS purges logs from the source DB instance after two hours or after the archive log retention hours setting has passed, whichever is longer. RDS purges logs from the read replica after the archive log retention hours setting has passed only if they have been successfully applied to the database.

In some cases, a primary DB instance might have one or more cross-Region read replicas. If so, Amazon RDS for Oracle keeps the transaction logs on the source DB instance until they have been transmitted and applied to all cross-Region read replicas. For information about `rdsadmin.rdsadmin_util.set_configuration`, see Retaining archived redo logs.

Outages during Oracle replication

When you create a read replica, Amazon RDS takes a DB snapshot of your source DB instance and begins replication. The source DB instance experiences a very brief I/O suspension when the DB snapshot operation begins. The I/O suspension typically lasts about one second. You can avoid the I/O suspension if the source DB instance is a Multi-AZ deployment, because in that case the snapshot is taken from the secondary DB instance.

The DB snapshot becomes the Oracle replica. Amazon RDS sets the necessary parameters and permissions for the source database and replica without service interruption. Similarly, if you delete a replica, no outage occurs.

Requirements and considerations for RDS for Oracle replicas

Before creating an Oracle replica, familiarize yourself with the following requirements and considerations.

Topics

- Version and licensing requirements for RDS for Oracle replicas
- Option group limitations for RDS for Oracle replicas
- Backup and restore considerations for RDS for Oracle replicas
- Oracle Data Guard requirements and limitations for RDS for Oracle replicas
- Multi-tenant configuration limitations for RDS for Oracle replicas
- Miscellaneous considerations for RDS for Oracle replicas

Version and licensing requirements for RDS for Oracle replicas

Before you create an RDS for Oracle replica, consider the following:

- If the replica is in read-only mode, make sure that you have an Active Data Guard license. If you place the replica in mounted mode, you don't need an Active Data Guard license. Only the Oracle DB engine supports mounted replicas.
- Oracle replicas are supported only for Oracle Enterprise Edition (EE).
- Oracle replicas of non-CDBs are supported only for DB instances created using non-CDB instances running Oracle Database 19c.
- Oracle replicas are available for DB instances running only on DB instance classes with two or more vCPUs. A source DB instance can't use the db.t3.small instance class.
- The Oracle DB engine version of the source DB instance and all its replicas must be the same. Amazon RDS upgrades the replicas immediately after upgrading the source DB instance, regardless of a replica's maintenance window. For major version upgrades of cross-Region replicas, Amazon RDS automatically does the following:
  - Generates an option group for the target version.
  - Copies all options and option settings from the original option group to the new option group.
  - Associates the upgraded cross-Region replica with the new option group.

  For more information about upgrading the DB engine version, see Upgrading the RDS for Oracle DB engine.

Option group limitations for RDS for Oracle replicas

When working with option groups for your RDS for Oracle replica, consider the following:

- You can't use a replica option group different from the source DB instance option group when the source and replica are in the same AWS Region.

  Modifications to the source option group or source option group membership propagate to Oracle replicas. These changes are applied to the replicas immediately after they are applied to the source DB instance, regardless of the replica's maintenance window. For more information about option groups, see Working with option groups.
- You can't remove an RDS for Oracle cross-Region replica from its dedicated option group, which is automatically created for the replica.
- You can't add the dedicated option group for an RDS for Oracle cross-Region replica to a different DB instance.
- You can't add or remove nonreplicated options from a dedicated option group for an RDS for Oracle cross-Region replica, with the exception of the following options:
  - `NATIVE_NETWORK_ENCRYPTION`
  - `OEM`
  - `OEM_AGENT`
  - `SSL`

  To add other options to an RDS for Oracle cross-Region replica, add them to the source DB instance's option group. The option is also installed on all of the source DB instance's replicas. For licensed options, make sure that there are sufficient licenses for the replicas.

  When you promote an RDS for Oracle cross-Region replica, the promoted replica behaves the same as other Oracle DB instances, including the management of its options. You can promote a replica explicitly or implicitly by deleting its source DB instance.

  For more information about option groups, see Working with option groups.
- You can't add the `EFS_INTEGRATION` option to RDS for Oracle cross-Region replicas.

Backup and restore considerations for RDS for Oracle replicas

Before you create an RDS for Oracle replica, consider the following:

- To create snapshots of RDS for Oracle replicas or turn on automatic backups, make sure to set the backup retention period manually. Automatic backups aren't turned on by default.

- When you restore a replica backup, you restore to the database time, not the time that the backup was taken. The *database time* refers to the latest applied transaction time of the data in the backup. The difference is significant because a replica can lag behind the primary for minutes or hours.

  To find the difference, use the `describe-db-snapshots` command. Compare the `snapshotDatabaseTime`, which is the database time of the replica backup, and the `OriginalSnapshotCreateTime` field, which is the latest applied transaction on the primary database.

Oracle Data Guard requirements and limitations for RDS for Oracle replicas

Before you create an RDS for Oracle replica, note the following requirements and limitations:

- If your primary DB instance uses the single-tenant or multi-tenant configuration of the multitenant architecture, consider the following:
  - You must use Oracle Database 19c or higher with the Enterprise Edition.
  - Your primary CDB instance must be in an `ACTIVE` lifecycle.
  - You can't convert a non-CDB primary instance to a CDB instance and convert its replicas in the same operation. Instead, delete the non-CDB replicas, convert the primary DB instance to a CDB, and then create new replicas
- Make sure that a logon trigger on a primary DB instance permits access to the `RDS_DATAGUARD` user and to any user whose `AUTHENTICATED_IDENTITY` value is `RDS_DATAGUARD` or `rdsdb`. Also, the trigger must not set the current schema for the `RDS_DATAGUARD` user.
- To avoid blocking connections from the Data Guard broker process, don't enable restricted sessions. For more information about restricted sessions, see Enabling and disabling restricted sessions.

Multi-tenant configuration limitations for RDS for Oracle replicas

When using the multi-tenant configuration on an RDS for Oracle replica, note the following limitations:

- You can only create, delete, or modify tenant databases on the primary DB instance. These changes automatically propagate to the replicas.

- Tenant databases on an RDS for Oracle primary, source, or replica cannot be created with a custom character set. If you require a custom character set, create the tenant databases before creating read replicas for the DB instance.

Miscellaneous considerations for RDS for Oracle replicas

Before you create an RDS for Oracle replica, consider the following:

- When you are creating an RDS for Oracle replica for a DB instance that has additional storage volumes, RDS automatically configure additional storage volumes on the replica. However, any subsequent modifications made in storage volumes of your primary DB instance are not automatically applied to the replica.
- If you add additional storage volumes in your primary DB instance, RDS does not automatically add additional storage volumes to the replica. You need to modify your replica to add additional storage volumes.
- If you modify storage volumes configuration such as storage size and IOPS in your primary DB instance, RDS does not automatically modify storage volumes in the replica. You need to modify your replica to update storage volume configurations.
- When managing datafile locations across volumes, note that changes made on your primary instance don't automatically sync to replicas.
  - For read-only replicas: You can either use parameter group settings to control default file locations, or manually move files after they're created.
  - For mounted replicas: Manual changes to datafile locations in the primary database require recreating the mounted replica to reflect those changes. To avoid this, we recommend using parameter group settings to manage default file locations.
- If your DB instance is a source for one or more cross-Region replicas, the source DB retains its archived redo log files until they are applied on all cross-Region replicas. The archived redo logs might result in increased storage consumption.
- To avoid disrupting RDS automation, system triggers must permit specific users to log on to the primary and replica database. System triggers include DDL, logon, and database role triggers. We recommend that you add code to your triggers to exclude the users listed in the following sample code:

```
-- Determine who the user is
SELECT SYS_CONTEXT('USERENV','AUTHENTICATED_IDENTITY') INTO CURRENT_USER FROM DUAL;
```

```
-- The following users should always be able to login to either the Primary or
 Replica
IF CURRENT_USER IN ('master_user', 'SYS', 'SYSTEM', 'RDS_DATAGUARD', 'rdsdb') THEN
RETURN;
END IF;
```

- Block change tracking is supported for read-only replicas, but not for mounted replicas. You can change a mounted replica to a read-only replica, and then enable block change tracking. For more information, see Enabling and disabling block change tracking.
- You can't create an Oracle read replica when the source database manages master user credentials with Secrets Manager.

Preparing to create an Oracle replica

Before you can begin using your replica, perform the following tasks.

Topics

- Enabling automatic backups
- Enabling force logging mode
- Changing your logging configuration
- Setting the MAX_STRING_SIZE parameter
- Planning compute and storage resources

Enabling automatic backups

Before a DB instance can serve as a source DB instance, make sure to enable automatic backups on the source DB instance. To learn how to perform this procedure, see Enabling automated backups.

Enabling force logging mode

We recommend that you enable force logging mode. In force logging mode, the Oracle database writes redo records even when `NOLOGGING` is used with data definition language (DDL) statements.

To enable force logging mode

1. Log in to your Oracle database using a client tool such as SQL Developer.
2. Enable force logging mode by running the following procedure.

```
exec rdsadmin.rdsadmin_util.force_logging(p_enable => true);
```

For more information about this procedure, see Setting force logging.

Changing your logging configuration

For *n* online redo logs of size *m*, RDS automatically creates *n*+1 standby logs of size *m* on the primary DB instance and all replicas. Whenever you change the logging configuration on the primary, the changes propagate automatically to the replicas.

If you change your logging configuration, consider the following guidelines:

- We recommend that you complete the changes before making a DB instance the source for replicas. RDS for Oracle also supports updating the instance after it becomes a source.
- Before you change the logging configuration on the primary DB instance, check that each replica has enough storage to accommodate the new configuration.

You can modify the logging configuration for a DB instance by using the Amazon RDS procedures `rdsadmin.rdsadmin_util.add_logfile` and `rdsadmin.rdsadmin_util.drop_logfile`. For more information, see Adding online redo logs and Dropping online redo logs.

Setting the MAX_STRING_SIZE parameter

Before you create an Oracle replica, ensure that the setting of the `MAX_STRING_SIZE` parameter is the same on the source DB instance and the replica. You can do this by associating them with the same parameter group. If you have different parameter groups for the source and the replica, you can set `MAX_STRING_SIZE` to the same value. For more information about setting this parameter, see Turning on extended data types for a new DB instance.

Planning compute and storage resources

Ensure that the source DB instance and its replicas are sized properly, in terms of compute and storage, to suit their operational load. If a replica reaches compute, network, or storage resource capacity, the replica stops receiving or applying changes from its source. Amazon RDS for Oracle doesn't intervene to mitigate high replica lag between a source DB instance and its replicas. You can modify the storage and CPU resources of a replica independently from its source and other replicas.

Creating an RDS for Oracle replica in mounted mode

By default, Oracle replicas are read-only. To create a replica in mounted mode, use the console, the AWS CLI, or the RDS API.

Console

To create a mounted replica from a source Oracle DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the Oracle DB instance that you want to use as the source for a mounted replica.
4. For **Actions**, choose **Create replica**.
5. For **Replica mode**, choose **Mounted**.
6. Choose the settings that you want to use. For **DB instance identifier**, enter a name for the read replica. Adjust other settings as needed.
7. For **Regions**, choose the Region where the mounted replica will be launched.
8. Choose your instance size and storage type. We recommend that you use the same DB instance class and storage type as the source DB instance for the read replica.
9. For **Multi-AZ deployment**, choose **Create a standby instance** to create a standby of your replica in another Availability Zone for failover support for the mounted replica. Creating your mounted replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ DB instance.
10. Choose the other settings that you want to use.
11. Choose **Create replica**.

In the **Databases** page, the mounted replica has the role Replica.

AWS CLI

To create an Oracle replica in mounted mode, set `--replica-mode` to `mounted` in the AWS CLI command create-db-instance-read-replica.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance-read-replica \
    --db-instance-identifier myreadreplica \
    --source-db-instance-identifier mydbinstance \
    --replica-mode mounted
```

For Windows:

```
aws rds create-db-instance-read-replica ^
    --db-instance-identifier myreadreplica ^
    --source-db-instance-identifier mydbinstance ^
    --replica-mode mounted
```

To change a read-only replica to a mounted state, set `--replica-mode` to `mounted` in the AWS CLI command modify-db-instance. To place a mounted replica in read-only mode, set `--replica-mode` to `open-read-only`.

RDS API

To create an Oracle replica in mounted mode, specify `ReplicaMode=mounted` in the RDS API operation CreateDBInstanceReadReplica.

Modifying the RDS for Oracle replica mode

To change the replica mode of an existing replica, use the console, AWS CLI, or RDS API. When you change to mounted mode, the replica disconnects all active connections. When you change to read-only mode, Amazon RDS initializes Active Data Guard.

The change operation can take a few minutes. During the operation, the DB instance status changes to **modifying**. For more information about status changes, see Viewing Amazon RDSDB instance status.

Console

To change the replica mode of an Oracle replica from mounted to read-only

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the mounted replica database.

4. Choose **Modify**.
5. For **Replica mode**, choose **Read-only**.
6. Choose the other settings that you want to change.
7. Choose **Continue**.
8. For **Scheduling of modifications**, choose **Apply immediately**.
9. Choose **Modify DB instance**.

AWS CLI

To change a read replica to mounted mode, set `--replica-mode` to `mounted` in the AWS CLI command modify-db-instance. To change a mounted replica to read-only mode, set `--replica-mode` to `open-read-only`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier myreadreplica \
    --replica-mode mode
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier myreadreplica ^
    --replica-mode mode
```

RDS API

To change a read-only replica to mounted mode, set `ReplicaMode=mounted` in ModifyDBInstance. To change a mounted replica to read-only mode, set `ReplicaMode=read-only`.

Working with RDS for Oracle replica backups

You can create and restore backups of an RDS for Oracle replica. Both automatic backups and manual snapshots are supported. For more information, see Backing up, restoring, and exporting

data. The following sections describe the key differences between managing backups of a primary and an RDS for Oracle replica.

Turning on RDS for Oracle replica backups

An Oracle replica doesn't have automated backups turned on by default. You turn on automated backups by setting the backup retention period to a positive nonzero value.

Console

To enable automated backups immediately

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance or Multi-AZ DB cluster that you want to modify.
3. Choose **Modify**.
4. For **Backup retention period**, choose a positive nonzero value, for example three days.
5. Choose **Continue**.
6. Choose **Apply immediately**.
7. Choose **Modify DB instance** or **Modify cluster** to save your changes and enable automated backups.

AWS CLI

To enable automated backups, use the AWS CLI modify-db-instance or modify-db-cluster command.

Include the following parameters:

- `--db-instance-identifier` (or `--db-cluster-identifier` for a Multi-AZ DB cluster)
- `--backup-retention-period`
- `--apply-immediately` or `--no-apply-immediately`

In the following example, we enable automated backups by setting the backup retention period to three days. The changes are applied immediately.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier my_db_instance  \
    --backup-retention-period 3 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier my_db_instance  ^
    --backup-retention-period 3 ^
    --apply-immediately
```

RDS API

To enable automated backups, use the RDS API ModifyDBInstance or ModifyDBCluster operation with the following required parameters:

- `DBInstanceIdentifier` or `DBClusterIdentifier`
- `BackupRetentionPeriod`

Restoring an RDS for Oracle replica backup

You can restore an Oracle replica backup just as you can restore a backup of the primary instance. For more information, see the following:

- Restoring to a DB instance
- Restoring a DB instance to a specified time for Amazon RDS

The main consideration when you restore a replica backup is determining the point in time to which you are restoring. The *database time* refers to the latest applied transaction time of the data in the backup. When you restore a replica backup, you restore to the database time, not the time when the backup completed. The difference is significant because an RDS for Oracle replica can lag behind the primary by minutes or hours. Thus, the database time of a replica backup, and thus the point in time to which you restore it, might be much earlier than the backup creation time.

To find the difference between database time and creation time, use the `describe-db-snapshots` command. Compare the `SnapshotDatabaseTime`, which is the database time of the replica backup, and the `OriginalSnapshotCreateTime` field, which is the latest applied transaction on the primary database. The following example shows the difference between the two times:

```
aws rds describe-db-snapshots \
    --db-instance-identifier my-oracle-replica
    --db-snapshot-identifier my-replica-snapshot

{
    "DBSnapshots": [
        {
            "DBSnapshotIdentifier": "my-replica-snapshot",
            "DBInstanceIdentifier": "my-oracle-replica",
            "SnapshotDatabaseTime": "2022-07-26T17:49:44Z",
            ...
            "OriginalSnapshotCreateTime": "2021-07-26T19:49:44Z"
        }
    ]
}
```

Performing an Oracle Data Guard switchover

A *switchover* is a role reversal between a primary database and a standby database. During a switchover, the original primary database transitions to a standby role, while the original standby database transitions to the primary role.

In an Oracle Data Guard environment, a primary database supports one or more standby databases. You can perform a managed, switchover-based role transition from a primary database to a standby database. A *switchover* is a role reversal between a primary database and a standby database. During a switchover, the original primary database transitions to a standby role, while the original standby database transitions to the primary role.

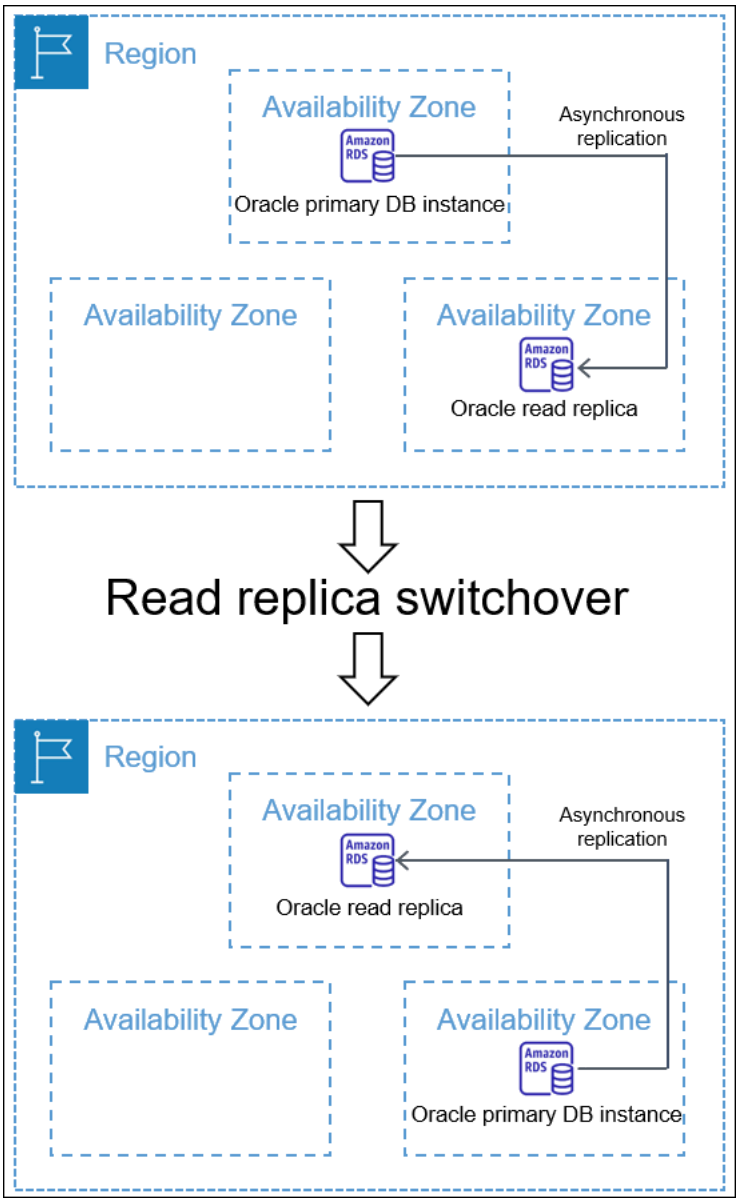
Topics

- Overview of Oracle Data Guard switchover
- Requirements for the Oracle Data Guard switchover
- Initiating the Oracle Data Guard switchover
- Monitoring the Oracle Data Guard switchover

Overview of Oracle Data Guard switchover

Amazon RDS supports a fully managed, switchover-based role transition for Oracle Database replicas. You can only initiate a switchover to a standby database that is mounted or open read-only.

The replicas can reside in separate AWS Regions or in different Availability Zones (AZs) of a single Region. All AWS Regions are supported.



A switchover differs from a read replica promotion. In a switchover, the source and replica DB instances change roles. In a promotion, a read replica becomes a source DB instance, but the source

DB instance doesn't become a replica. For more information, see Promoting a read replica to be a standalone DB instance.

Topics

- Benefits of Oracle Data Guard switchover
- Supported Oracle Database versions
- Cost of Oracle Data Guard switchover
- How Oracle Data Guard switchover works

Benefits of Oracle Data Guard switchover

Just as for RDS for Oracle read replicas, a managed switchover relies on Oracle Data Guard. The operation is designed to have zero data loss. Amazon RDS automates the following aspects of the switchover:

- Reverses the roles of your primary database and specified standby database, putting the new standby database in the same state (mounted or read-only) as the original standby
- Ensures data consistency
- Maintains your replication configuration after the transition
- Supports repeated reversals, allowing your new standby database to return to its original primary role

Supported Oracle Database versions

Oracle Data Guard switchover is supported for Oracle Database 19c and higher releases.

Cost of Oracle Data Guard switchover

The Oracle Data Guard switchover feature doesn't incur additional costs. Oracle Database Enterprise Edition includes support for standby databases in mounted mode. To open standby databases in read-only mode, you need the Oracle Active Data Guard option.

How Oracle Data Guard switchover works

Oracle Data Guard switchover is a fully managed operation. You initiate the switchover for a standby database by issuing the CLI command `switchover-read-replica`. Then Amazon RDS modifies the primary and standby roles in your replication configuration.

The *original standby* and *original primary* are the roles that exist before the switchover. The *new standby* and *new primary* are the roles that exist after the switchover. A *bystander replica* is a replica database that serves as a standby database in the Oracle Data Guard environment but is not switching roles.

Topics

- Stages of the Oracle Data Guard switchover
- After the Oracle Data Guard switchover

Stages of the Oracle Data Guard switchover

To perform the switchover, Amazon RDS must take the following steps:

1. Block new transactions on the original primary database. During the switchover, Amazon RDS interrupts replication for all databases in your Oracle Data Guard configuration. During the switchover, the original primary database can't process write requests.
2. Ship unapplied transactions to the original standby database, and apply them.
3. Restart the new standby database in read-only or mounted mode. The mode depends on the open state of the original standby database before the switchover.
4. Open the new primary database in read/write mode.

After the Oracle Data Guard switchover

Amazon RDS switches the roles of the primary and standby database. You are responsible for reconnecting your application and performing any other desired configuration.

Topics

- Success criteria
- Connection to the new primary database
- Configuration of the new primary database

Success criteria

The Oracle Data Guard switchover is successful when the original standby database does the following:

- Transitions to its role as new primary database

- Completes its reconfiguration

To limit downtime, your new primary database becomes active as soon as possible. Because Amazon RDS configures bystander replicas asynchronously, these replicas might become active after the original primary database.

Connection to the new primary database

Amazon RDS won't propagate your current database connections to the new primary database after the switchover. After the Oracle Data Guard switchover completes, reconnect your application to the new primary database.

Configuration of the new primary database

To perform a switchover to the new primary database, Amazon RDS changes the mode of the original standby database to open. The change in role is the only change to the database. Amazon RDS doesn't set up features such as Multi-AZ replication.

If you perform a switchover to a cross-Region replica with different options, the new primary database keeps its own options. Amazon RDS won't migrate the options on the original primary database. If the original primary database had options such as SSL, NNE, OEM, and OEM_AGENT, Amazon RDS doesn't propagate them to the new primary database.

Requirements for the Oracle Data Guard switchover

Before initiating the Oracle Data Guard switchover, make sure that your replication environment meets the following requirements:

- The original standby database is mounted or open read-only.
- Automatic backups are enabled on the original standby database.
- The original primary database and the original standby database are in the `available` state.
- The original primary database and the original standby database don't have pending maintenance actions in any of the following states: `required`, `next window`, or `in progress`. Actions in these states block switchover. To learn how to check the status of pending maintenance updates, see Viewing pending maintenance updates.

  Pending maintenance actions in the `available` state don't block switchover. RDS for Oracle frequently releases operating system (OS) updates in the `available` state. These pending OS

updates won't block a switchover unless you schedule them for the next maintenance window, which puts them in the `next window` state.

> **Note**
>
> If you want to defer a scheduled maintenance action so that you can execute a switchover, choose **Actions** and then **Defer upgrade** in the RDS console. You can also prevent a switchover from being blocked by applying a pending maintenance action or moving the maintenance window to an interval before your switchover. For more information, see the re:Post article How to remove RDS pending maintenance items.

- The original standby database is in the replicating state.
- You aren't attempting to initiate a switchover when either the primary database or standby database is currently in a switchover lifecycle. If a replica database is reconfiguring after a switchover, Amazon RDS prevents you from initiating another switchover.

> **Note**
>
> A *bystander replica* is a replica in the Oracle Data Guard configuration that isn't the target of the switchover. Bystander replicas can be in any state during the switchover.

- The original standby database has a configuration that is as close as desired to the original primary database. Assume a scenario where the original primary and original standby databases have different options. After the switchover completes, Amazon RDS doesn't automatically reconfigure the new primary database to have the same options as the original primary database.
- You configure your desired Multi-AZ deployment before initiating a switchover. Amazon RDS doesn't manage Multi-AZ as part of the switchover. The Multi-AZ deployment remains as it is.

  Assume that db_maz is the primary database in a Multi-AZ deployment, and db_saz is a Single-AZ replica. You initiate a switchover from db_maz to db_saz. Afterward, db_maz is a Multi-AZ replica database, and db_saz is a Single-AZ primary database. The new primary database is now unprotected by a Multi-AZ deployment.
- In preparation for a cross-Region switchover, the primary database doesn't use the same option group as a DB instance outside of the replication configuration. For a cross-Region switchover to succeed, the current primary database and its read replicas must be the only DB instances

to use the option group of the current primary database. Otherwise, Amazon RDS prevents the switchover.

Initiating the Oracle Data Guard switchover

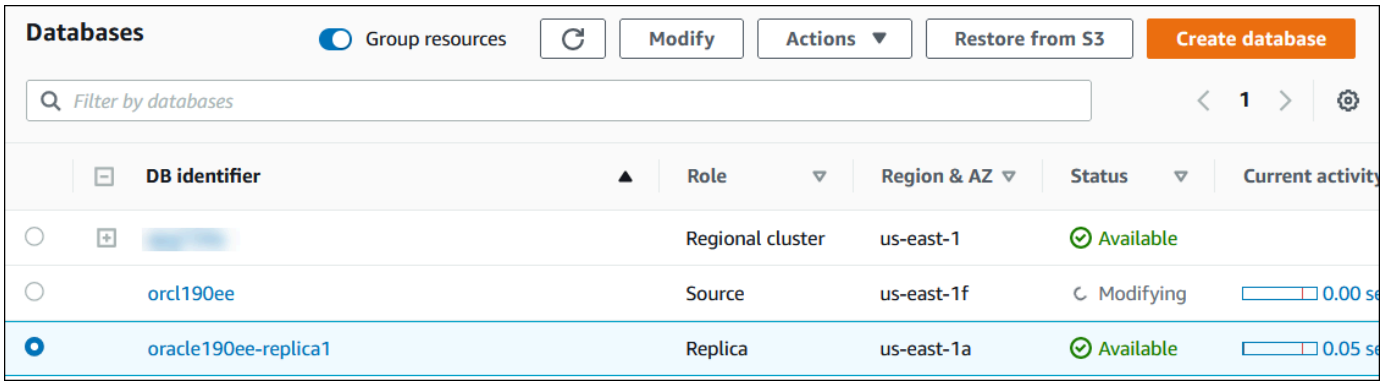
You can switch over an RDS for Oracle read replica to the primary role, and the former primary DB instance to a replica role.

Console

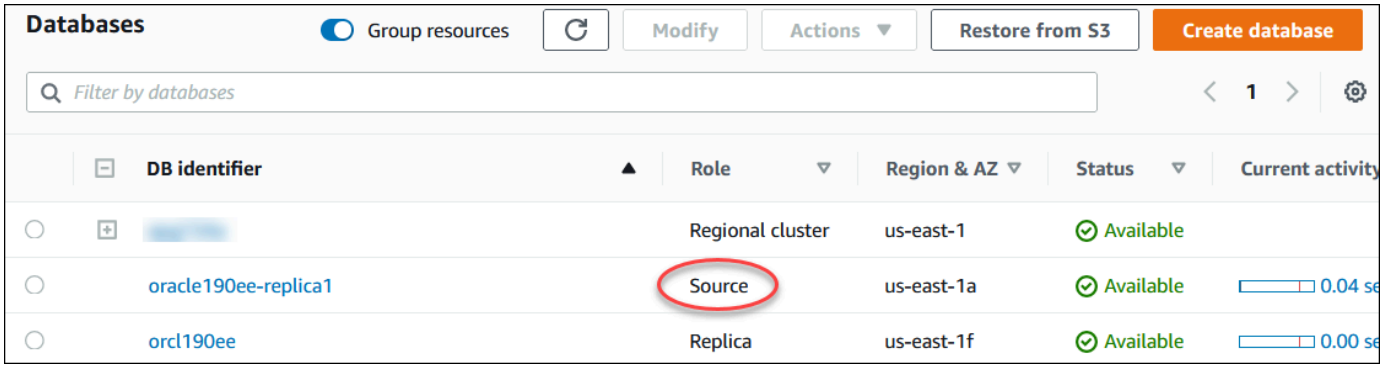
To switch over an Oracle read replica to the primary DB role

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the Amazon RDS console, choose **Databases**.

   The **Databases** pane appears. Each read replica shows **Replica** in the **Role** column.
3. Choose the read replica that you want to switch over to the primary role.
4. For **Actions**, choose **Switch over replica**.
5. Choose **I acknowledge**. Then choose **Switch over replica**.
6. On the **Databases** page, monitor the progress of the switchover.



When the switchover completes, the role of the switchover target changes from **Replica** to **Source**.



AWS CLI

To switch over an Oracle replica to the primary DB role, use the AWS CLI `switchover-read-replica` command. The following examples make the Oracle replica named *`replica-to-be-made-primary`* into the new primary database.

Example

For Linux, macOS, or Unix:

```
aws rds switchover-read-replica \
    --db-instance-identifier replica-to-be-made-primary
```

For Windows:

```
aws rds switchover-read-replica ^
    --db-instance-identifier replica-to-be-made-primary
```

RDS API

To switch over an Oracle replica to the primary DB role, call the Amazon RDS API `SwitchoverReadReplica` operation with the required parameter `DBInstanceIdentifier`. This parameter specifies the name of the Oracle replica that you want to assume the primary DB role.

Monitoring the Oracle Data Guard switchover

To check the status of your instances, use the AWS CLI command `describe-db-instances`. The following command checks the status of the DB instance *`orcl2`*. This database was a standby database before the switchover, but is the new primary database after the switchover.

```
aws rds describe-db-instances \
```

```
    --db-instance-identifier orcl2
```

To confirm that the switchover completed successfully, query `V$DATABASE.OPEN_MODE`. Check that the value for the new primary database is `READ WRITE`.

```
SELECT OPEN_MODE FROM V$DATABASE;
```

To look for switchover-related events, use the AWS CLI command `describe-events`. The following example looks for events on the *orcl2* instance.

```
aws rds describe-events \
    --source-identifier orcl2 \
    --source-type db-instance
```

Troubleshooting RDS for Oracle replicas

This section describes possible replication problems and solutions.

Topics

- Monitoring Oracle replication lag
- Troubleshooting Oracle replication failure after adding or modifying triggers

Monitoring Oracle replication lag

To monitor replication lag in Amazon CloudWatch, view the Amazon RDS `ReplicaLag` metric. For more information about replication lag time, see Monitoring read replication and Amazon CloudWatch metrics for Amazon RDS.

For a read replica, if the lag time is too long, query the following views:

- `V$ARCHIVED_LOG` – Shows which commits have been applied to the read replica.
- `V$DATAGUARD_STATS` – Shows a detailed breakdown of the components that make up the `ReplicaLag` metric.
- `V$DATAGUARD_STATUS` – Shows the log output from Oracle's internal replication processes.

For a mounted replica, if the lag time is too long, you can't query the `V$` views. Instead, do the following:

- Check the `ReplicaLag` metric in CloudWatch.
- Check the alert log file for the replica in the console. Look for errors in the recovery messages. The messages include the log sequence number, which you can compare to the primary sequence number. For more information, see Amazon RDS for Oracle database log files.

Troubleshooting Oracle replication failure after adding or modifying triggers

If you add or modify any triggers, and if replication fails afterward, the problem may be the triggers. Ensure that the trigger excludes the following user accounts, which are required by RDS for replication:

- User accounts with administrator privileges
- `SYS`
- `SYSTEM`
- `RDS_DATAGUARD`
- `rdsdb`

For more information, see Miscellaneous considerations for RDS for Oracle replicas.

Redo transport compression with RDS for Oracle

Use RDS for Oracle redo transport compression to improve the replication performance between your primary DB instance and standby replicas. This is particularly useful in environments that have limited network bandwidth or high-latency connections.

Obtaining a license for redo transport compression

Redo transport compression is part of the Oracle Advanced Compression option. To use redo transport compression, you need a valid license for the Oracle Advanced Compression option. For licensing information, contact your Oracle representative.

Configuring redo transport compression

To configure redo transport compression, you can use the `rds.replica.redo_compression` parameter. This parameter is available for Oracle versions 19c and 21c.

The `rds.replica.redo_compression` parameter accepts the following values:

- `DISABLE` – Default value that disables redo transport compression.
- `ENABLE` – Value that enables redo transport compression through the default algorithm ZLIB.
- `ZLIB` – Value that explicitly enables redo transport compression using the ZLIB algorithm, which provides good compression ratios.
- `LZO` – Value that explicitly enables redo transport compression using the LZO algorithm, which optimizes compression speed, particularly during decompression.

Performance considerations for redo transport compression

Compression and decompression operations consume CPU resources on both the primary and standby instances. If you use redo transport compression, consider instance resource usage and network conditions.

Related topics for redo transport compression

For more information on configuring redo transport compression, see the following resources:

- DB parameter groups for Amazon RDS DB instances
- RedoCompression in the Oracle Database 19c release notes

Adding options to Oracle DB instances

In Amazon RDS, an *option* is an additional feature. Following, you can find a description of options that you can add to Amazon RDS instances running the Oracle DB engine.

Topics

- Overview of Oracle DB options
- Amazon S3 integration
- Oracle Application Express (APEX)
- Amazon EFS integration
- Oracle Java virtual machine
- Oracle Enterprise Manager
- Oracle Label Security
- Oracle Locator
- Oracle native network encryption
- Oracle OLAP
- Oracle Secure Sockets Layer
- Oracle Spatial
- Oracle SQLT
- Oracle Statspack
- Oracle time zone
- Oracle time zone file autoupgrade
- Oracle Transparent Data Encryption
- Oracle UTL_MAIL
- Oracle XML DB

Overview of Oracle DB options

To enable options for your Oracle database, add them to an option group, and then associate the option group with your DB instance. For more information, see Working with option groups.

Topics

- Summary of Oracle Database options
- Options supported for different editions
- Memory requirements for specific options

Summary of Oracle Database options

You can add the following options for Oracle DB instances.

Option	Option ID
Amazon S3 integration	S3_INTEGRATION
Oracle Application Express (APEX)	APEX APEX-DEV
Oracle Enterprise Manager	OEM OEM_AGENT
Oracle Java virtual machine	JVM
Oracle Label Security	OLS
Oracle Locator	LOCATOR
Oracle native network encryption	NATIVE_NETWORK_ENCRYPTION
Oracle OLAP	OLAP
Oracle Secure Sockets Layer	SSL
Oracle Spatial	SPATIAL
Oracle SQLT	SQLT
Oracle Statspack	STATSPACK
Oracle time zone	Timezone

Option	Option ID
Oracle time zone file autoupgrade	TIMEZONE_FILE_AUTOUPGRADE
Oracle Transparent Data Encryption	TDE
Oracle UTL_MAIL	UTL_MAIL
Oracle XML DB	XMLDB

Options supported for different editions

RDS for Oracle prevents you from adding options to an edition if they aren't supported. To find out which RDS options are supported in different Oracle Database editions, use the command `aws rds describe-option-group-options`. The following example lists supported options for Oracle Database 19c Enterprise Edition.

```
aws rds describe-option-group-options \
    --engine-name oracle-ee \
    --major-engine-version 19
```

For more information, see describe-option-group-options in the *AWS CLI Command Reference*.

Memory requirements for specific options

Some options require additional memory to run on your DB instance. For example, Oracle Enterprise Manager Database Control uses about 300 MB of RAM. If you enable this option for a small DB instance, you might encounter performance problems due to memory constraints. You can adjust the Oracle parameters so that the database requires less RAM. Alternatively, you can scale up to a larger DB instance.

Amazon S3 integration

You can transfer files between your RDS for Oracle DB instance and an Amazon S3 bucket. You can use Amazon S3 integration with Oracle Database features such as Oracle Data Pump. For example, you can download Data Pump files from Amazon S3 to your RDS for Oracle DB instance. For more information, see Importing data into Oracle on Amazon RDS.

> **Note**
>
> Your DB instance and your Amazon S3 bucket must be in the same AWS Region.

Topics

- Configuring IAM permissions for RDS for Oracle integration with Amazon S3
- Adding the Amazon S3 integration option
- Transferring files between Amazon RDS for Oracle and an Amazon S3 bucket
- Troubleshooting Amazon S3 integration
- Removing the Amazon S3 integration option

Configuring IAM permissions for RDS for Oracle integration with Amazon S3

For RDS for Oracle to integrate with Amazon S3, your DB instance must have access to an Amazon S3 bucket. The Amazon VPC used by your DB instance doesn't need to provide access to the Amazon S3 endpoints.

RDS for Oracle supports transferring files between a DB instance in one account and an Amazon S3 bucket in a different account. Where additional steps are required, they are noted in the following sections.

Topics

- Step 1: Create an IAM policy for your Amazon RDS role
- Step 2: (Optional) Create an IAM policy for your Amazon S3 bucket
- Step 3: Create an IAM role for your DB instance and attach your policy
- Step 4: Associate your IAM role with your RDS for Oracle DB instance

Step 1: Create an IAM policy for your Amazon RDS role

In this step, you create an AWS Identity and Access Management (IAM) policy with the permissions required to transfer files between your Amazon S3 bucket and your RDS DB instance. This step assumes that you have already created an S3 bucket.

Before you create the policy, note the following pieces of information:

- The Amazon Resource Name (ARN) for your bucket
- The ARN for your AWS KMS key, if your bucket uses SSE-KMS or SSE-S3 encryption

> **Note**
>
> An RDS for Oracle DB instance can't access Amazon S3 buckets encrypted with SSE-C.

For more information, see Protecting data using server-side encryption in the *Amazon Simple Storage Service User Guide*.

Console

To create an IAM policy to allow Amazon RDS to access your Amazon S3 bucket

1. Open the IAM Management Console.
2. Under **Access management**, choose **Policies**.
3. Choose **Create Policy**.
4. On the **Visual editor** tab, choose **Choose a service**, and then choose **S3**.
5. For **Actions**, choose **Expand all**, and then choose the bucket permissions and object permissions required to transfer files from an Amazon S3 bucket to Amazon RDS. For example, do the following:
   - Expand **List**, and then select **ListBucket**.
   - Expand **Read**, and then select **GetObject**.
   - Expand **Write**, and then select **PutObject**, **DeleteObject**, **AbortMultipartUpload**, and **ListMultipartUploadParts**. The multipart upload permissions are required when uploading large files (100 MB or larger) to Amazon S3.

- Expand **Permissions management**, and then select **PutObjectAcl**. This permission is necessary if you plan to upload files to a bucket owned by a different account, and this account needs full control of the bucket contents.

Object permissions are permissions for object operations in Amazon S3. You must grant them for objects in a bucket, not the bucket itself. For more information, see Permissions for object operations.

6. Choose **Resources**, and then do the following:

   a. Choose **Specific**.

   b. For **bucket**, choose **Add ARN**. Enter your bucket ARN. The bucket name is filled in automatically. Then choose **Add**.

   c. If the **object** resource is shown, either choose **Add ARN** to add resources manually or choose **Any**.

   > **Note**
   >
   > You can set **Amazon Resource Name (ARN)** to a more specific ARN value to allow Amazon RDS to access only specific files or folders in an Amazon S3 bucket. For more information about how to define an access policy for Amazon S3, see Managing access permissions to your Amazon S3 resources.

7. (Optional) Choose **Add additional permissions** to add resources to the policy. For example, do the following:

   a. If your bucket is encrypted with a custom KMS key, select **KMS** for the service.

   b. For **Manual actions**, select the following:

      - **Encrypt**
      - **ReEncrypt from** and **ReEncrypt to**
      - **Decrypt**
      - **DescribeKey**
      - **GenerateDataKey**

   c. For **Resources**, choose **Specific**.

d. For **key**, choose **Add ARN**. Enter the ARN of your custom key as the resource, and then choose **Add**.

   For more information, see Protecting Data Using Server-Side Encryption with KMS keys Stored in AWS Key Management Service (SSE-KMS) in the *Amazon Simple Storage Service User Guide*.

e. If you want Amazon RDS to access to access other buckets, add the ARNs for these buckets. Optionally, you can also grant access to all buckets and objects in Amazon S3.

8. Choose **Next: Tags** and then **Next: Review**.
9. For **Name**, enter a name for your IAM policy, for example `rds-s3-integration-policy`. You use this name when you create an IAM role to associate with your DB instance. You can also add an optional **Description** value.
10. Choose **Create policy**.

AWS CLI

Create an AWS Identity and Access Management (IAM) policy that grants Amazon RDS access to an Amazon S3 bucket. After you create the policy, note the ARN of the policy. You need the ARN for a subsequent step.

Include the appropriate actions in the policy based on the type of access required:

- `GetObject` – Required to transfer files from an Amazon S3 bucket to Amazon RDS.
- `ListBucket` – Required to transfer files from an Amazon S3 bucket to Amazon RDS.
- `PutObject` – Required to transfer files from Amazon RDS to an Amazon S3 bucket.
- `AbortMultipartUpload` – Required for multipart uploads when transferring large files (100 MB or larger) from Amazon RDS to an Amazon S3 bucket.
- `ListMultipartUploadParts` – Required for multipart uploads when transferring large files (100 MB or larger) from Amazon RDS to an Amazon S3 bucket.

The following AWS CLI command creates an IAM policy named *`rds-s3-integration-policy`* with these options. It grants access to a bucket named *`amzn-s3-demo-bucket`*.

Example

For Linux, macOS, or Unix:

```
aws iam create-policy \
   --policy-name rds-s3-integration-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3integration",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket",
           "s3:PutObject",
           "s3:AbortMultipartUpload",
           "s3:ListMultipartUploadParts"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

The following example includes permissions for custom KMS keys.

```
aws iam create-policy \
   --policy-name rds-s3-integration-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3integration",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket",
           "s3:PutObject",
           "kms:Decrypt",
           "kms:Encrypt",
           "kms:ReEncrypt*",
           "kms:GenerateDataKey",
           "kms:DescribeKey",
         ],
         "Effect": "Allow",
```

```
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket",
          "arn:aws:s3:::amzn-s3-demo-bucket/*",
          "arn:aws:kms:::your-kms-arn"
        ]
      }
    ]
  }'
```

For Windows:

```
aws iam create-policy ^
   --policy-name rds-s3-integration-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3integration",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket",
           "s3:PutObject",
           "s3:AbortMultipartUpload",
           "s3:ListMultipartUploadParts"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

The following example includes permissions for custom KMS keys.

```
aws iam create-policy ^
   --policy-name rds-s3-integration-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3integration",
```

```
        "Action": [
          "s3:GetObject",
          "s3:ListBucket",
          "s3:PutObject",
          "kms:Decrypt",
          "kms:Encrypt",
          "kms:ReEncrypt",
          "kms:GenerateDataKey",
          "kms:DescribeKey",
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket",
          "arn:aws:s3:::amzn-s3-demo-bucket/*",
          "arn:aws:kms:::your-kms-arn"
        ]
      }
    ]
  }'
```

Step 2: (Optional) Create an IAM policy for your Amazon S3 bucket

This step is necessary only in the following conditions:

- You plan to upload files to an Amazon S3 bucket from one account (account A) and access them from a different account (account B).
- Account B owns the bucket.
- Account B needs full control of objects loaded into the bucket.

If the preceding conditions don't apply to you, skip to Step 3: Create an IAM role for your DB instance and attach your policy.

To create your bucket policy, make sure you have the following:

- The account ID for account A
- The user name for account A
- The ARN value for the Amazon S3 bucket in account B

Console

To create or edit a bucket policy

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the **Buckets** list, choose the name of the bucket that you want to create a bucket policy for or whose bucket policy you want to edit.
3. Choose **Permissions**.
4. Under **Bucket policy**, choose **Edit**. This opens the Edit bucket policy page.
5. On the **Edit bucket policy** page, explore **Policy examples** in the *Amazon S3 User Guide*, choose **Policy generator** to generate a policy automatically, or edit the JSON in the **Policy** section.

   If you choose **Policy generator**, the AWS Policy Generator opens in a new window:

   a. On the **AWS Policy Generator** page, in **Select Type of Policy**, choose **S3 Bucket Policy**.

   b. Add a statement by entering the information in the provided fields, and then choose **Add Statement**. Repeat for as many statements as you would like to add. For more information about these fields, see the IAM JSON policy elements reference in the *IAM User Guide*.

   > **Note**
   >
   > For convenience, the **Edit bucket policy** page displays the **Bucket ARN** (Amazon Resource Name) of the current bucket above the **Policy** text field. You can copy this ARN for use in the statements on the **AWS Policy Generator** page.

   c. After you finish adding statements, choose **Generate Policy**.

   d. Copy the generated policy text, choose **Close**, and return to the **Edit bucket policy** page in the Amazon S3 console.
6. In the **Policy** box, edit the existing policy or paste the bucket policy from the Policy generator. Make sure to resolve security warnings, errors, general warnings, and suggestions before you save your policy.

   JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ExamplePermissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/account-A-user"
      },
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-destination-bucket",
        "arn:aws:s3:::amzn-s3-demo-destination-bucket/*"
      ]
    }
  ]
}
```

7. Choose **Save changes**, which returns you to the Bucket Permissions page.

Step 3: Create an IAM role for your DB instance and attach your policy

This step assumes that you have created the IAM policy in Step 1: Create an IAM policy for your Amazon RDS role. In this step, you create a role for your RDS for Oracle DB instance and then attach your policy to the role.

Console

To create an IAM role to allow Amazon RDS to access an Amazon S3 bucket

1. Open the IAM Management Console.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Choose **AWS service**.
5. For **Use cases for other AWS services:**, choose **RDS** and then **RDS – Add Role to Database**. Then choose **Next**.

6. For **Search** under **Permissions policies**, enter the name of the IAM policy you created in Step 1: Create an IAM policy for your Amazon RDS role, and select the policy when it appears in the list. Then choose **Next**.
7. For **Role name**, enter a name for your IAM role, for example, `rds-s3-integration-role`. You can also add an optional **Description** value.
8. Choose **Create role**.

AWS CLI

To create a role and attach your policy to it

1. Create an IAM role that Amazon RDS can assume on your behalf to access your Amazon S3 buckets.

   We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based trust relationships to limit the service's permissions to a specific resource. This is the most effective way to protect against the confused deputy problem.

   You might use both global condition context keys and have the `aws:SourceArn` value contain the account ID. In this case, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same statement.

   - Use `aws:SourceArn` if you want cross-service access for a single resource.
   - Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

   In the trust relationship, make sure to use the `aws:SourceArn` global condition context key with the full Amazon Resource Name (ARN) of the resources accessing the role.

   The following AWS CLI command creates the role named *`rds-s3-integration-role`* for this purpose.

   Example

   For Linux, macOS, or Unix:

```
aws iam create-role \
   --role-name rds-s3-integration-role \
```

```
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                 "aws:SourceAccount": "my_account_ID",
                 "aws:SourceArn": "arn:aws:rds:Region:my_account_ID:db:dbname"
             }
         }
       }
     ]
   }'
```

For Windows:

```
aws iam create-role ^
   --role-name rds-s3-integration-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                 "aws:SourceAccount": "my_account_ID",
                 "aws:SourceArn": "arn:aws:rds:Region:my_account_ID:db:dbname"
             }
         }
       }
     ]
   }'
```

For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.

2. After the role is created, note the ARN of the role. You need the ARN for a subsequent step.
3. Attach the policy you created to the role you created.

   The following AWS CLI command attaches the policy to the role named *rds-s3-integration-role*.

   Example

   For Linux, macOS, or Unix:

```
aws iam attach-role-policy \
   --policy-arn your-policy-arn \
   --role-name rds-s3-integration-role
```

   For Windows:

```
aws iam attach-role-policy ^
   --policy-arn your-policy-arn ^
   --role-name rds-s3-integration-role
```

   Replace *your-policy-arn* with the policy ARN that you noted in a previous step.

Step 4: Associate your IAM role with your RDS for Oracle DB instance

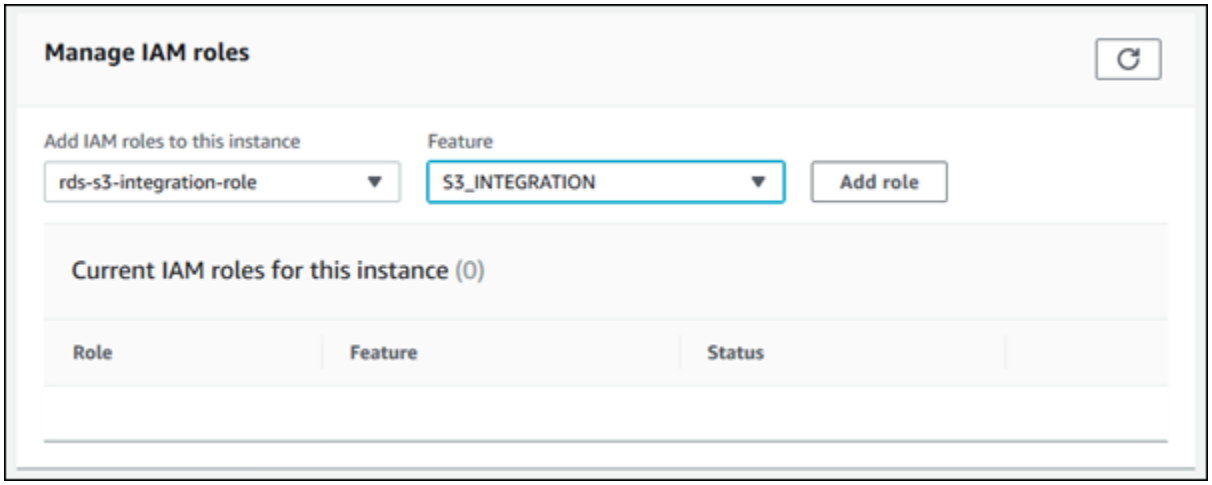
The last step in configuring permissions for Amazon S3 integration is associating your IAM role with your DB instance. Note the following requirements:

- You must have access to an IAM role with the required Amazon S3 permissions policy attached to it.
- You can only associate one IAM role with your RDS for Oracle DB instance at a time.
- Your DB instance must be in the **Available** state.

Console

To associate your IAM role with your RDS for Oracle DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Databases** from the navigation pane.
3. Choose the RDS for Oracle DB instance name to display its details.
4. On the **Connectivity & security** tab, scroll down to the **Manage IAM roles** section at the bottom of the page.
5. For **Add IAM roles to this instance**, choose the role that you created in Step 3: Create an IAM role for your DB instance and attach your policy.
6. For **Feature**, choose **S3_INTEGRATION**.



7. Choose **Add role**.

AWS CLI

The following AWS CLI command adds the role to an Oracle DB instance named *mydbinstance*.

Example

For Linux, macOS, or Unix:

```
aws rds add-role-to-db-instance \
   --db-instance-identifier mydbinstance \
   --feature-name S3_INTEGRATION \
   --role-arn your-role-arn
```

For Windows:

```
aws rds add-role-to-db-instance ^
   --db-instance-identifier mydbinstance ^
   --feature-name S3_INTEGRATION ^
   --role-arn your-role-arn
```

Replace *your-role-arn* with the role ARN that you noted in a previous step. `S3_INTEGRATION` must be specified for the `--feature-name` option.

Adding the Amazon S3 integration option

To integrate Amazon RDS for Oracle with Amazon S3, your DB instance must be associated with an option group that includes the `S3_INTEGRATION` option.

Console

To configure an option group for Amazon S3 integration

1. Create a new option group or identify an existing option group to which you can add the `S3_INTEGRATION` option.

   For information about creating an option group, see Creating an option group.
2. Add the `S3_INTEGRATION` option to the option group.

   For information about adding an option to an option group, see Adding an option to an option group.
3. Create a new RDS for Oracle DB instance and associate the option group with it, or modify an RDS for Oracle DB instance to associate the option group with it.

   For information about creating a DB instance, see Creating an Amazon RDS DB instance.

   For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

AWS CLI

To configure an option group for Amazon S3 integration

1. Create a new option group or identify an existing option group to which you can add the `S3_INTEGRATION` option.

For information about creating an option group, see Creating an option group.

2. Add the S3_INTEGRATION option to the option group.

   For example, the following AWS CLI command adds the S3_INTEGRATION option to an option group named **myoptiongroup**.

   Example

   For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
   --option-group-name myoptiongroup \
   --options OptionName=S3_INTEGRATION,OptionVersion=1.0
```

   For Windows:

```
aws rds add-option-to-option-group ^
   --option-group-name myoptiongroup ^
   --options OptionName=S3_INTEGRATION,OptionVersion=1.0
```

3. Create a new RDS for Oracle DB instance and associate the option group with it, or modify an RDS for Oracle DB instance to associate the option group with it.

   For information about creating a DB instance, see Creating an Amazon RDS DB instance.

   For information about modifying an RDS for Oracle DB instance, see Modifying an Amazon RDS DB instance.

Transferring files between Amazon RDS for Oracle and an Amazon S3 bucket

To transfer files between an RDS for Oracle DB instance and an Amazon S3 bucket, you can use the Amazon RDS package rdsadmin_s3_tasks. You can compress files with GZIP when uploading them, and decompress them when downloading.

Topics

- Requirements and limitations for file transfers
- Uploading files from your RDS for Oracle DB instance to an Amazon S3 bucket
- Downloading files from an Amazon S3 bucket to an Oracle DB instance

- Monitoring the status of a file transfer

Requirements and limitations for file transfers

Before transferring files between your DB instance and an Amazon S3 bucket, note the following:

- The `rdsadmin_s3_tasks` package transfers files located in a single directory. You can't include subdirectories in a transfer.
- The maximum object size in an Amazon S3 bucket is 5 TB.
- Tasks created by `rdsadmin_s3_tasks` run asynchronously.
- You can upload files from the Data Pump directory, such as `DATA_PUMP_DIR`, or any user-created directory. You can't upload files from a directory used by Oracle background processes, such as the `adump`, `bdump`, or `trace` directories.
- The download limit is 2000 files per procedure call for `download_from_s3`. If you need to download more than 2000 files from Amazon S3, split your download into separate actions, with no more than 2000 files per procedure call.
- If a file exists in your download folder, and you attempt to download a file with the same name, `download_from_s3` skips the download. To remove a file from the download directory, use the PL/SQL procedure UTL_FILE.FREMOVE.

Uploading files from your RDS for Oracle DB instance to an Amazon S3 bucket

To upload files from your DB instance to an Amazon S3 bucket, use the procedure `rdsadmin.rdsadmin_s3_tasks.upload_to_s3`. For example, you can upload Oracle Recovery Manager (RMAN) backup files or Oracle Data Pump files. For more information about working with objects, see Amazon Simple Storage Service User Guide. For more information about performing RMAN backups, see Performing common RMAN tasks for Oracle DB instances.

The `rdsadmin.rdsadmin_s3_tasks.upload_to_s3` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_bucket_name	VARCHAR2	–	required	The name of the Amazon S3 bucket to upload files to.

Parameter name	Data type	Default	Required	Description
p_directory_name	VARCHAR2	–	required	The name of the Oracle directory object to upload files from. The directory can be any user-created directory object or the Data Pump directory, such as DATA_PUMP_DIR . You can't upload files from a directory used by background processes, such as adump, bdump, and trace. **Note** You can only upload files from the specified directory. You can't upload files in subdirectories in the specified directory.

Parameter name	Data type	Default	Required	Description
p_s3_prefix	VARCHAR2	–	required	An Amazon S3 file name prefix that files are uploaded to. An empty prefix uploads all files to the top level in the specified Amazon S3 bucket and doesn't add a prefix to the file names. For example, if the prefix is folder_1/oradb , files are uploaded to folder_1. In this case, the oradb prefix is added to each file.
p_prefix	VARCHAR2	–	required	A file name prefix that file names must match to be uploaded. An empty prefix uploads all files in the specified directory.
p_compression_level	NUMBER	0	optional	The level of GZIP compression. Valid values range from 0 to 9: • 0 – No compression • 1 – Fastest compression • 9 – Highest compression

Parameter name	Data type	Default	Required	Description
p_bucket_owner_full_control	VARCHAR2	–	optional	The access control setting for the bucket. The only valid values are null or FULL_CONTROL . This setting is required only if you upload files from one account (account A) into a bucket owned by a different account (account B), and account B needs full control of the files.

The return value for the rdsadmin.rdsadmin_s3_tasks.upload_to_s3 procedure is a task ID.

The following example uploads all of the files in the *DATA_PUMP_DIR* directory to the Amazon S3 bucket named *amzn-s3-demo-bucket*. The files aren't compressed.

```
SELECT rdsadmin.rdsadmin_s3_tasks.upload_to_s3(
      p_bucket_name    =>  'amzn-s3-demo-bucket',
      p_prefix         =>  '',
      p_s3_prefix      =>  '',
      p_directory_name =>  'DATA_PUMP_DIR')
   AS TASK_ID FROM DUAL;
```

The following example uploads all of the files with the prefix *db* in the *DATA_PUMP_DIR* directory to the Amazon S3 bucket named *amzn-s3-demo-bucket*. Amazon RDS applies the highest level of GZIP compression to the files.

```
SELECT rdsadmin.rdsadmin_s3_tasks.upload_to_s3(
      p_bucket_name       =>  'amzn-s3-demo-bucket',
      p_prefix            =>  'db',
      p_s3_prefix         =>  '',
      p_directory_name    =>  'DATA_PUMP_DIR',
      p_compression_level =>  9)
```

```
   AS TASK_ID FROM DUAL;
```

The following example uploads all of the files in the *DATA_PUMP_DIR* directory to the Amazon S3 bucket named *amzn-s3-demo-bucket*. The files are uploaded to a `dbfiles` folder. In this example, the GZIP compression level is *1*, which is the fastest level of compression.

```
SELECT rdsadmin.rdsadmin_s3_tasks.upload_to_s3(
      p_bucket_name       =>  'amzn-s3-demo-bucket',
      p_prefix            =>  '',
      p_s3_prefix         =>  'dbfiles/',
      p_directory_name    =>  'DATA_PUMP_DIR',
      p_compression_level =>  1)
   AS TASK_ID FROM DUAL;
```

The following example uploads all of the files in the *DATA_PUMP_DIR* directory to the Amazon S3 bucket named *amzn-s3-demo-bucket*. The files are uploaded to a `dbfiles` folder and `ora` is added to the beginning of each file name. No compression is applied.

```
SELECT rdsadmin.rdsadmin_s3_tasks.upload_to_s3(
      p_bucket_name    =>  'amzn-s3-demo-bucket',
      p_prefix         =>  '',
      p_s3_prefix      =>  'dbfiles/ora',
      p_directory_name =>  'DATA_PUMP_DIR')
   AS TASK_ID FROM DUAL;
```

The following example assumes that the command is run in account A, but account B requires full control of the bucket contents. The command `rdsadmin_s3_tasks.upload_to_s3` transfers all files in the *DATA_PUMP_DIR* directory to the bucket named *s3bucketOwnedByAccountB*. Access control is set to `FULL_CONTROL` so that account B can access the files in the bucket. The GZIP compression level is *6*, which balances speed and file size.

```
SELECT rdsadmin.rdsadmin_s3_tasks.upload_to_s3(
      p_bucket_name               =>  's3bucketOwnedByAccountB',
      p_prefix                    =>  '',
      p_s3_prefix                 =>  '',
      p_directory_name            =>  'DATA_PUMP_DIR',
      p_bucket_owner_full_control =>  'FULL_CONTROL',
      p_compression_level         =>  6)
   AS TASK_ID FROM DUAL;
```

In each example, the SELECT statement returns the ID of the task in a VARCHAR2 data type.

You can view the result by displaying the task's output file.

```
SELECT text FROM table(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-task-id.log'));
```

Replace *task-id* with the task ID returned by the procedure.

> **Note**
>
> Tasks are executed asynchronously.

Downloading files from an Amazon S3 bucket to an Oracle DB instance

To download files from an Amazon S3 bucket to an RDS for Oracle instance, use the Amazon RDS procedure `rdsadmin.rdsadmin_s3_tasks.download_from_s3`.

The `download_from_s3` procedure has the following parameters.

Parameter name	Data type	Default	Required	Description
p_bucket_name	VARCHAR	–	Required	The name of the Amazon S3 bucket to download files from.
p_directory_name	VARCHAR	–	Required	The name of the Oracle directory object to download files to. The directory can be any user-created directory object or the Data Pump directory, such as DATA_PUMP_DIR .
p_error_on_zero_downloads	VARCHAR	FALSE	Optional	A flag that determines whether the task raises an error when no objects in the Amazon S3 bucket match the prefix. If this parameter is not set or set to FALSE (default), the task prints a message that no objects were found,

Parameter name	Data type	Default	Required	Description
				but doesn't raise an exception or fail. If this parameter is TRUE, the task raises an exception and fails. Examples of prefix specifications that can fail match tests are spaces in prefixes, as in `' import/test9.log'` , and case mismatches, as in `test9.log` and `test9.LOG` .

Parameter name	Data type	Default	Required	Description
p_s3_prefix	VARCHAR	–	Required	A file name prefix that file names must match to be downloaded. An empty prefix downloads all of the top level files in the specified Amazon S3 bucket, but not the files in folders in the bucket. The procedure downloads Amazon S3 objects only from the first level folder that matches the prefix. Nested directory structures matching the specified prefix are not downloaded. For example, suppose that an Amazon S3 bucket has the folder structure folder_1/folder_2/folder_3 . You specify the 'folder_1/folder_2/' prefix. In this case, only the files in folder_2 are downloaded, not the files in folder_1 or folder_3. If, instead, you specify the 'folder_1/folder_2' prefix, all files in folder_1 that match the 'folder_2' prefix are downloaded, and no files in folder_2 are downloaded.
p_decompression_format	VARCHAR	–	Optional	The decompression format. Valid values are NONE for no decompression and GZIP for decompression.

The return value for the rdsadmin.rdsadmin_s3_tasks.download_from_s3 procedure is a task ID.

The following example downloads all files in the Amazon S3 bucket named *amzn-s3-demo-bucket* to the *DATA_PUMP_DIR* directory. The files aren't compressed, so no decompression is applied.

```
SELECT rdsadmin.rdsadmin_s3_tasks.download_from_s3(
      p_bucket_name    =>  'amzn-s3-demo-bucket',
      p_directory_name =>  'DATA_PUMP_DIR')
   AS TASK_ID FROM DUAL;
```

The following example downloads all of the files with the prefix *db* in the Amazon S3 bucket named *amzn-s3-demo-bucket* to the *DATA_PUMP_DIR* directory. The files are compressed with GZIP, so decompression is applied. The parameter `p_error_on_zero_downloads` turns on prefix error checking, so if the prefix doesn't match any files in the bucket, the task raises and exception and fails.

```
SELECT rdsadmin.rdsadmin_s3_tasks.download_from_s3(
      p_bucket_name               =>  'amzn-s3-demo-bucket',
      p_s3_prefix                 =>  'db',
      p_directory_name            =>  'DATA_PUMP_DIR',
      p_decompression_format      =>  'GZIP',
      p_error_on_zero_downloads   =>  'TRUE')
   AS TASK_ID FROM DUAL;
```

The following example downloads all of the files in the folder *myfolder/* in the Amazon S3 bucket named *amzn-s3-demo-bucket* to the *DATA_PUMP_DIR* directory. Use the `p_s3_prefix` parameter to specify the Amazon S3 folder. The uploaded files are compressed with GZIP, but aren't decompressed during the download.

```
SELECT rdsadmin.rdsadmin_s3_tasks.download_from_s3(
      p_bucket_name          =>  'amzn-s3-demo-bucket',
      p_s3_prefix            =>  'myfolder/',
      p_directory_name       =>  'DATA_PUMP_DIR',
      p_decompression_format =>  'NONE')
   AS TASK_ID FROM DUAL;
```

The following example downloads the file *mydumpfile.dmp* in the Amazon S3 bucket named *amzn-s3-demo-bucket* to the *DATA_PUMP_DIR* directory. No decompression is applied.

```
SELECT rdsadmin.rdsadmin_s3_tasks.download_from_s3(
```

```
      p_bucket_name    =>  'amzn-s3-demo-bucket',
      p_s3_prefix      =>  'mydumpfile.dmp',
      p_directory_name =>  'DATA_PUMP_DIR')
   AS TASK_ID FROM DUAL;
```

In each example, the SELECT statement returns the ID of the task in a VARCHAR2 data type.

You can view the result by displaying the task's output file.

```
SELECT text FROM table(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-task-id.log'));
```

Replace *task-id* with the task ID returned by the procedure.

> **Note**
>
> Tasks are executed asynchronously.
> You can use the UTL_FILE.FREMOVE Oracle procedure to remove files from a directory. For more information, see FREMOVE procedure in the Oracle documentation.

Monitoring the status of a file transfer

File transfer tasks publish Amazon RDS events when they start and when they complete. The event message contains the task ID for the file transfer. For information about viewing events, see Viewing Amazon RDS events.

You can view the status of an ongoing task in a bdump file. The bdump files are located in the /rdsdbdata/log/trace directory. Each bdump file name is in the following format.

```
dbtask-task-id.log
```

Replace *task-id* with the ID of the task that you want to monitor.

> **Note**
>
> Tasks are executed asynchronously.

You can use the `rdsadmin.rds_file_util.read_text_file` stored procedure to view the contents of bdump files. For example, the following query returns the contents of the *dbtask-1234567890123-1234.log* bdump file.

```
SELECT text FROM
 table(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-1234567890123-1234.log'));
```

The following sample shows the log file for a failed transfer.

```
TASK_ID




                                                
```

Troubleshooting Amazon S3 integration

For troubleshooting tips, see the AWS re:Post article How do troubleshoot issues when I integrate Amazon RDS for Oracle with Amazon S3?.

Removing the Amazon S3 integration option

You can remove Amazon S3 integration option from a DB instance.

To remove the Amazon S3 integration option from a DB instance, do one of the following:

- To remove the Amazon S3 integration option from multiple DB instances, remove the `S3_INTEGRATION` option from the option group to which the DB instances belong. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
- To remove the Amazon S3 integration option from a single DB instance, modify the instance and specify a different option group that doesn't include the `S3_INTEGRATION` option. You can specify the default (empty) option group or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Oracle Application Express (APEX)

Amazon RDS supports Oracle Application Express (APEX) through the use of the `APEX` and `APEX-DEV` options. You can deploy Oracle APEX as a runtime environment or as a full development environment for web-based applications. Using Oracle APEX, you can build applications entirely within the web browser. For more information, see Oracle application Express in the Oracle documentation.

Topics

- Oracle APEX components
- Requirements and limitations
- Setting up Oracle APEX and Oracle Rest Data Services (ORDS)
- Configuring Oracle Rest Data Services (ORDS)
- Upgrading and removing Oracle APEX

Oracle APEX components

Oracle APEX consists of the following main components:

- A *repository* that stores the metadata for Oracle APEX applications and components. The repository consists of tables, indexes, and other objects that are installed in your Amazon RDS DB instance.
- A *listener* that manages HTTP communications with Oracle APEX clients. The listener resides on a separate host such as an Amazon EC2 instance, an on-premises server at your company, or your desktop computer. The listener accepts incoming connections from web browsers, forwards them to the Amazon RDS DB instance for processing, and then sends results from the repository back to the browsers.

  RDS for Oracle supports the following types of listeners:

  - For Oracle APEX version 5.0 and later, use Oracle REST Data Services (ORDS) version 19.1 and higher. We recommend that you use the latest supported version of Oracle APEX and ORDS. This documentation describes older versions for backwards compatibility only.
  - For Oracle APEX version 4.1.1, you can use Oracle APEX Listener version 1.1.4.
  - You can use Oracle HTTP Server and `mod_plsql` listeners.

> **Note**
>
> Amazon RDS doesn't support the Oracle XML DB HTTP server with the embedded PL/SQL gateway as a listener for Oracle APEX. In general, Oracle recommends against using the embedded PL/SQL gateway for applications that run on the internet.

For more information about these listener types, see About choosing a web listener in the Oracle documentation.

When you add the APEX and APEX-DEV options to your RDS for Oracle DB instance, Amazon RDS installs the Oracle APEX repository only. Install your listener on a separate host.

Requirements and limitations

The following topic lists the requirements and limitations for Oracle APEX and ORDS.

Oracle APEX version requirements

The APEX option uses storage on the DB instance class for your DB instance. Following are the supported versions and approximate storage requirements for Oracle APEX.

Oracle APEX version	Storage requirements	Supported Oracle Database versions	Notes
Oracle APEX version 24.2.v1	114 MiB	All	This version includes patch 37885097: PSE BUNDLE FOR APEX 24.2 (PSES ON TOP OF 24.2.0), PATCH_VERSION 4.
Oracle APEX version 24.1.v1	112 MiB	All	This version includes patch 36695709: PSE BUNDLE FOR APEX 24.1 (PSES ON TOP OF 24.1.0), PATCH_VERSION 3. If you need exactly the same APEX images version to install on your EC2 instance, download patch 37544819: 24.1.3 PSE BUNDLE FOR APEX 24.1 (PSES ON TOP OF 24.1.0).

Oracle APEX version	Storage requirements	Supported Oracle Database versions	Notes
Oracle APEX version 23.2.v1	110 MiB	All	This version includes patch 35895964: PSE BUNDLE FOR APEX 23.2 (PSES ON TOP OF 23.2.0), PATCH_VERSION 6. If you need exactly the same APEX images version to install on your EC2 instance, download patch 37593125: 23.2.6 PSE BUNDLE FOR APEX 23.2 (PSES ON TOP OF 23.2.0).
Oracle APEX version 23.1.v1	106 MiB	All	This version includes patch 35283657: PSE BUNDLE FOR APEX 23.1 (PSES ON TOP OF 23.1.0), PATCH_VERSION 2.
Oracle APEX version 22.2.v1	106 MiB	All	This version includes patch 34628174: PSE BUNDLE FOR APEX 22.2 (PSES ON TOP OF 22.2.0), PATCH_VERSION 4.
Oracle APEX version 22.1.v1	124 MiB	All	This version includes patch 34020981: PSE BUNDLE FOR APEX 22.1 (PSES ON TOP OF 22.1.0), PATCH_VERSION 6.
Oracle APEX version 21.2.v1	125 MiB	All	This version includes patch 33420059: PSE BUNDLE FOR APEX 21.2 (PSES ON TOP OF 21.2.0), PATCH_VERSION 8.
Oracle APEX version 21.1.v1	125 MiB	All	This version includes patch 32598392: PSE BUNDLE FOR APEX 21.1, PATCH_VERSION 3.

Oracle APEX version	Storage requirements	Supported Oracle Database versions	Notes
Oracle APEX version 20.2.v1	148 MiB	All except Oracle Database 21c	This version includes patch 32006852: PSE BUNDLE FOR APEX 20.2, PATCH_VERSION 2020.11.12. You can see the patch number and date by running the following query: `SELECT PATCH_VERSION, PATCH_NUMBER FROM APEX_PATCHES;`
Oracle APEX version 20.1.v1	173 MiB	All except Oracle Database 21c	This version includes patch 30990551: PSE BUNDLE FOR APEX 20.1, PATCH_VERSION 2020.07.15.
Oracle APEX version 19.2.v1	149 MiB	All except Oracle Database 21c	
Oracle APEX version 19.1.v1	148 MiB	All except Oracle Database 21c	

For downloadable Oracle APEX .zip files, see Oracle APEX Prior Release Archives on the Oracle website.

Oracle APEX and ORDS prerequisites

Note the following prerequisites for using Oracle APEX and ORDS:

- Your system must use the Java Runtime Environment (JRE).
- Your Oracle client installation must include the following:

- SQL*Plus or SQL Developer for administration tasks
- Oracle Net Services for configuring connections to your RDS for Oracle DB instance

Oracle APEX limitations

You can't modify the APEX_*version* user account, which is managed by Amazon RDS. Thus, you can't apply database profiles or enforce password rules on this user. The profiles and password settings for APEX_*version* are predefined by Oracle and AWS and are designed to meet the security requirements for Amazon RDS.

Setting up Oracle APEX and Oracle Rest Data Services (ORDS)

The following topic lists the steps required to set up Oracle APEX and ORDS

Topics

- Adding the APEX and APEX-DEV options to your DB instance
- Unlocking the public user account on your DB instance
- Configuring RESTful services for Oracle APEX
- Preparing to install ORDS on a separate host
- Setting up Oracle APEX listener

Adding the APEX and APEX-DEV options to your DB instance

To add the APEX and APEX-DEV options to your RDS for Oracle DB instance, do the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the APEX and APEX-DEV options to the option group.
3. Associate the option group with your DB instance.

When you add the APEX and APEX-DEV options, a brief outage occurs while your DB instance is automatically restarted.

> **Note**
>
> APEX_MAIL is available when the APEX option is installed. The execute privilege for the APEX_MAIL package is granted to PUBLIC so you don't need the APEX administrative account to use it.

To add the APEX and APEX-DEV options to a DB instance

1. Determine the option group that you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   a. For **Engine**, choose the Oracle edition that you want to use. The APEX and APEX-DEV options are supported on all editions.

   b. For **Major engine version**, choose the version of your DB instance.

   For more information, see Creating an option group.

2. Add the options to the option group. If you want to deploy only the Oracle APEX runtime environment, add only the APEX option. To deploy the full development environment, add both the APEX and APEX-DEV options.

   For **Version**, choose the version of Oracle APEX that you want to use.

   > **Important**
   >
   > If you add the APEX or APEX-DEV options to an existing option group that is already attached to one or more DB instances, a brief outage occurs. During this outage, all the DB instances are automatically restarted.

   For more information about adding options, see Adding an option to an option group.

3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.

- For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. When you add the APEX or APEX-DEV options to an existing DB instance, a brief outage occurs while your DB instance is automatically restarted. For more information, see Modifying an Amazon RDS DB instance.

Unlocking the public user account on your DB instance

After you have installed the APEX or APEX-DEV options your DB instance, make sure to do the following:

1. Change the password for the APEX_PUBLIC_USER account.
2. Unlock the account.

You can do this by using the Oracle SQL*Plus command line utility. Connect to your DB instance as the master user, and issue the following commands. Replace new_password with a password of your choice.

```
ALTER USER APEX_PUBLIC_USER IDENTIFIED BY new_password;
ALTER USER APEX_PUBLIC_USER ACCOUNT UNLOCK;
```

Configuring RESTful services for Oracle APEX

To configure RESTful services in Oracle APEX (not needed for Oracle APEX 4.1.1.V1), use SQL*Plus to connect to your DB instance as the master user. After you do this, run the rdsadmin.rdsadmin_run_apex_rest_config stored procedure. When you run the stored procedure, you provide passwords for the following users:

- APEX_LISTENER
- APEX_REST_PUBLIC_USER

The stored procedure runs the apex_rest_config.sql script, which creates new database accounts for these users.

> **Note**
>
> Configuration isn't required for Oracle APEX version 4.1.1.v1. For this Oracle APEX version only, you don't need to run the stored procedure.

The following command runs the stored procedure.

```
EXEC rdsadmin.rdsadmin_run_apex_rest_config('apex_listener_password',
 'apex_rest_public_user_password');
```

Preparing to install ORDS on a separate host

Install ORDS on a separate host such as an Amazon EC2 instance, an on-premises server at your company, or your desktop computer. The examples in this section,assume that your host runs Linux and is named `myapexhost.example.com`.

Before you can install ORDS, you need to create a nonprivileged OS user, and then download and unzip the Oracle APEX installation file.

To prepare for ORDS installation

1. Log in to `myapexhost.example.com` as `root`.
2. Create a nonprivileged OS user to own the listener installation. The following command creates a new user named *apexuser*.

   ```
   useradd -d /home/apexuser apexuser
   ```

   The following command assigns a password to the new user.

   ```
   passwd apexuser;
   ```

3. Log in to `myapexhost.example.com` as `apexuser`, and download the Oracle APEX installation file from Oracle to your `/home/apexuser` directory:

   - http://www.oracle.com/technetwork/developer-tools/apex/downloads/index.html
   - Oracle application Express prior release archives

4. Unzip the file in the `/home/apexuser` directory.

   ```
   unzip apex_version.zip
   ```

   After you unzip the file, there is an `apex` directory in the `/home/apexuser` directory.

5. While you are still logged into `myapexhost.example.com` as `apexuser`, download the Oracle REST Data Services file from Oracle to your `/home/apexuser` directory: http://www.oracle.com/technetwork/developer-tools/apex-listener/downloads/index.html.

Setting up Oracle APEX listener

> **Note**
>
> Oracle APEX Listener is deprecated.

Amazon RDS for Oracle continues to support Oracle APEX version 4.1.1 and Oracle APEX Listener version 1.1.4. We recommend that you use the latest supported versions of Oracle APEX and ORDS.

Install Oracle APEX Listener on a separate host such as an Amazon EC2 instance, an on-premises server at your company, or your desktop computer. We assume that the name of your host is `myapexhost.example.com`, and that your host is running Linux.

Preparing to install Oracle APEX listener

Before you can install Oracle APEX Listener, you need to create a nonprivileged OS user, and then download and unzip the Oracle APEX installation file.

To prepare for Oracle APEX listener installation

1. Log in to `myapexhost.example.com` as `root`.
2. Create a nonprivileged OS user to own the listener installation. The following command creates a new user named *apexuser*.

   ```
   useradd -d /home/apexuser apexuser
   ```

   The following command assigns a password to the new user.

   ```
   passwd apexuser;
   ```

3. Log in to `myapexhost.example.com` as `apexuser`, and download the Oracle APEX installation file from Oracle to your `/home/apexuser` directory:

   - http://www.oracle.com/technetwork/developer-tools/apex/downloads/index.html

- Oracle application Express prior release archives

4. Unzip the file in the /home/apexuser directory.

```
unzip apex_<version>.zip
```

After you unzip the file, there is an apex directory in the /home/apexuser directory.

5. While you are still logged into myapexhost.example.com as apexuser, download the Oracle APEX Listener file from Oracle to your /home/apexuser directory.

Installing and configuring Oracle APEX listener

Before you can use Oracle APEX, you need to download the apex.war file, use Java to install Oracle APEX Listener, and then start the listener.

To install and configure Oracle APEX listener

1. Create a new directory based on Oracle APEX Listener and open the listener file.

Run the following code:

```
mkdir /home/apexuser/apexlistener
cd /home/apexuser/apexlistener
unzip ../apex_listener.version.zip
```

2. Run the following code.

```
java -Dapex.home=./apex -Dapex.images=/home/apexuser/apex/images -Dapex.erase -
jar ./apex.war
```

3. Enter information for the program prompts following:

- The APEX Listener Administrator user name. The default is *adminlistener*.
- A password for the APEX Listener Administrator.
- The APEX Listener Manager user name. The default is *managerlistener*.
- A password for the APEX Listener Administrator.

The program prints a URL that you need to complete the configuration, as follows.

```
INFO: Please complete configuration at: http://localhost:8080/apex/
listenerConfigure
Database is not yet configured
```

4. Leave Oracle APEX Listener running so that you can use Oracle Application Express. When you have finished this configuration procedure, you can run the listener in the background.
5. From your web browser, go to the URL provided by the Oracle APEX Listener program. The Oracle Application Express Listener administration window appears. Enter the following information:

   - **Username** – `APEX_PUBLIC_USER`
   - **Password** – the password for *APEX_PUBLIC_USER*. This password is the one that you specified earlier when you configured the Oracle APEX repository. For more information, see Unlocking the public user account on your DB instance.
   - **Connection type** – Basic
   - **Hostname** – the endpoint of your Amazon RDS DB instance, such as `mydb.f9rbfa893tft.us-east-1.rds.amazonaws.com`.
   - **Port** – 1521
   - **SID** – the name of the database on your Amazon RDS DB instance, such as `mydb`.
6. Choose **Apply**. The Oracle APEX administration window appears.
7. Set a password for the Oracle APEX `admin` user. To do this, use SQL*Plus to connect to your DB instance as the master user, and then run the following commands.

   ```
   EXEC rdsadmin.rdsadmin_util.grant_apex_admin_role;
   grant APEX_ADMINISTRATOR_ROLE to master;
   @/home/apexuser/apex/apxchpwd.sql
   ```

   Replace *`master`* with your master user name. When the `apxchpwd.sql` script prompts you, enter a new `admin` password.
8. Return to the Oracle APEX administration window in your browser and choose **Administration**. Next, choose **Application Express Internal Administration**. When you are prompted for credentials, enter the following information:

   - **User name** – `admin`
   - **Password** – the password you set using the `apxchpwd.sql` script

Choose **Login**, and then set a new password for the `admin` user.

Your listener is now ready for use.

Configuring Oracle Rest Data Services (ORDS)

The following topic lists the configuration options for ORDS 21 and 22:

Topics

- Installing and configuring ORDS 21 and lower
- Installing and configuring ORDS 22 and higher

Installing and configuring ORDS 21 and lower

You are now ready to install and configure Oracle Rest Data Services (ORDS) for use with Oracle APEX. For Oracle APEX version 5.0 and later, use ORDS versions 19.1 to 21. To learn how to install ORDS 22 and higher, see Installing and configuring ORDS 22 and higher.

Install the listener on a separate host such as an Amazon EC2 instance, an on-premises server at your company, or your desktop computer. For the examples in this section, we assume that the name of your host is `myapexhost.example.com`, and that your host is running Linux.

To install and configure ORDS 21 and lower for use with Oracle APEX

1. Go to Oracle REST data services, and examine the Readme. Make sure that you have the required version of Java installed.
2. Create a new directory for your ORDS installation.

   ```
   mkdir /home/apexuser/ORDS
   cd /home/apexuser/ORDS
   ```

3. Download the file `ords.version.number.zip` from Oracle REST data services.
4. Unzip the file into the `/home/apexuser/ORDS` directory.
5. If you're installing ORDS in a multitenant database, add the following line to the file `/home/apexuser/ORDS/params/ords_params.properties`:

```
pdb.disable.lockdown=false
```

6. Grant the master user the required privileges to install ORDS.

   After the options for Oracle APEX are installed, give the master user the required privileges to install the ORDS schema. You can do this by connecting to the database and running the following commands. Replace *MASTER_USER* with the uppercase name of your master user.

   > **Important**
   >
   > When you enter the user name, use uppercase unless you created the user with a case-sensitive identifier. For example, if you run `CREATE USER myuser` or `CREATE USER MYUSER`, the data dictionary stores `MYUSER`. However, if you use double quotes in `CREATE USER "MyUser"`, the data dictionary stores `MyUser`. For more information, see Granting SELECT or EXECUTE privileges to SYS objects.

```
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_OBJECTS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_ROLE_PRIVS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_TAB_COLUMNS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_CONS_COLUMNS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_CONSTRAINTS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_OBJECTS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_PROCEDURES', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_TAB_COLUMNS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_TABLES', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_VIEWS', 'MASTER_USER', 'SELECT',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('WPIUTL', 'MASTER_USER', 'EXECUTE',
 true);
```

```
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_SESSION', 'MASTER_USER',
 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_UTILITY', 'MASTER_USER',
 'EXECUTE', true);
```

> **Note**
>
> These commands apply to ORDS version 19.1 and later.

7. Install the ORDS schema using the downloaded ords.war file.

```
java -jar ords.war install advanced
```

The program prompts you for the following information. The default values are in brackets. For more information, see Introduction to Oracle REST data services in the Oracle documentation.

- Enter the location to store configuration data:

  Enter */home/apexuser/ORDS*. This is the location of the ORDS configuration files.

- Specify the database connection type to use. Enter number for [1] Basic [2] TNS [3] Custom URL [1]:

  Choose the desired connection type.

- Enter the name of the database server [localhost]: *DB_instance_endpoint*

  Choose the default or enter the correct value.

- Enter the database listener port [1521]: *DB_instance_port*

  Choose the default or enter the correct value.

- Enter 1 to specify the database service name, or 2 to specify the database SID [1]:

  Choose 2 to specify the database SID.

- Database SID [xe]

  Choose the default or enter the correct value.

- Enter 1 if you want to verify/install Oracle REST Data Services schema or 2 to skip this step [1]:

Choose 1. This step creates the Oracle REST Data Services proxy user named ORDS_PUBLIC_USER.

- Enter the database password for ORDS_PUBLIC_USER:

  Enter the password, and then confirm it.

- Requires to login with administrator privileges to verify Oracle REST Data Services schema.

  Enter the administrator user name: *master_user*

  Enter the database password for *master_user*: *master_user_password*

  Confirm the password: *master_user_password*

  > **Note**
  >
  > Specify a password other than the prompt shown here as a security best practice.

- Enter the default tablespace for ORDS_METADATA [SYSAUX].

  Enter the temporary tablespace for ORDS_METADATA [TEMP].

  Enter the default tablespace for ORDS_PUBLIC_USER [USERS].

  Enter the temporary tablespace for ORDS_PUBLIC_USER [TEMP].

- Enter 1 if you want to use PL/SQL Gateway or 2 to skip this step. If you're using Oracle Application Express or migrating from mod_plsql, you must enter 1 [1].

  Choose the default.

- Enter the PL/SQL Gateway database user name [APEX_PUBLIC_USER]

  Choose the default.

- Enter the database password for APEX_PUBLIC_USER:

  Enter the password, and then confirm it.

- Enter 1 to specify passwords for Application Express RESTful Services database users (APEX_LISTENER, APEX_REST_PUBLIC_USER) or 2 to skip this step [1]:

  Choose 2 for APEX 4.1.1.V1; choose 1 for all other APEX versions.

- [Not needed for APEX 4.1.1.v1] Database password for APEX_LISTENER

  Enter the password (if required), and then confirm it.

- [Not needed for APEX 4.1.1.v1] Database password for APEX_REST_PUBLIC_USER

  Enter the password (if required), and then confirm it.

- Enter a number to select a feature to enable:

  Enter 1 to enable all features: SQL Developer Web, REST Enabled SQL, and Database API.

- Enter 1 if you wish to start in standalone mode or 2 to exit [1]:

  Enter 1.

- Enter the APEX static resources location:

  If you unzipped APEX installation files into `/home/apexuser`, enter `/home/apexuser/apex/images`. Otherwise, enter *`unzip_path`*`/apex/images`, where *`unzip_path`* is the directory where you unzipped the file.

- Enter 1 if using HTTP or 2 if using HTTPS [1]:

  If you enter 1, specify the HTTP port. If you enter 2, specify the HTTPS port and the SSL host name. The HTTPS option prompts you to specify how you will provide the certificate:

  - Enter 1 to use the self-signed certificate.
  - Enter 2 to provide your own certificate. If you enter 2, specify the path for the SSL certificate and the path for the SSL certificate private key.

8. Set a password for the APEX `admin` user. To do this, use SQL*Plus to connect to your DB instance as the master user, and then run the following commands.

```
EXEC rdsadmin.rdsadmin_util.grant_apex_admin_role;
grant APEX_ADMINISTRATOR_ROLE to master;
@/home/apexuser/apex/apxchpwd.sql
```

   Replace *`master`* with your master user name. When the `apxchpwd.sql` script prompts you, enter a new `admin` password.

9. Start the ORDS listener. Run the following code.

```
java -jar ords.war
```

The first time you start ORDS, you are prompted to provide the location of the APEX Static resources. This images folder is located in the `/apex/images` directory in the installation directory for APEX.

10. Return to the Oracle APEX administration window in your browser and choose **Administration**. Next, choose **Application Express Internal Administration**. When you are prompted for credentials, enter the following information:

    - **User name** – `admin`
    - **Password** – the password you set using the `apxchpwd.sql` script

    Choose **Login**, and then set a new password for the `admin` user.

Your listener is now ready for use.

Installing and configuring ORDS 22 and higher

You are now ready to install and configure Oracle Rest Data Services (ORDS) for use with Oracle APEX. For the examples in this section, we assume that the name of your separate host is `myapexhost.example.com`, and that your host is running Linux. The instructions for ORDS 22 differ from the instructions for previous releases.

To install and configure ORDS 22 and higher for use with Oracle APEX

1. Go to Oracle REST data services, and examine the Readme for the ORDS version that you plan to download. Make sure that you have the required version of Java installed.
2. Create a new directory for your ORDS installation.

   ```
   mkdir /home/apexuser/ORDS
   cd /home/apexuser/ORDS
   ```

3. Download the file `ords.`*`version.number`*`.zip` or `ords-latest.zip` from Oracle REST data services.
4. Unzip the file into the `/home/apexuser/ORDS` directory.
5. Grant the master user the required privileges to install ORDS.

After the APEX option is installed, give the master user the required privileges to install the ORDS schema. You can do this by logging in to the database and running the following commands. Replace *MASTER_USER* with the uppercase name of your master user.

> **Important**
>
> When you enter the user name, use uppercase unless you created the user with a case-sensitive identifier. For example, if you run `CREATE USER myuser` or `CREATE USER MYUSER`, the data dictionary stores `MYUSER`. However, if you use double quotes in `CREATE USER "MyUser"`, the data dictionary stores `MyUser`. For more information, see Granting SELECT or EXECUTE privileges to SYS objects.

```
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_OBJECTS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_ROLE_PRIVS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_TAB_COLUMNS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_CONS_COLUMNS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_CONSTRAINTS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_OBJECTS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_PROCEDURES', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_TAB_COLUMNS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_TABLES', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('USER_VIEWS', 'MASTER_USER', 'SELECT',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('WPIUTL', 'MASTER_USER', 'EXECUTE',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_SESSION', 'MASTER_USER',
 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_UTILITY', 'MASTER_USER',
 'EXECUTE', true);
```

```
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_LOB', 'MASTER_USER', 'EXECUTE',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_ASSERT', 'MASTER_USER',
 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_OUTPUT', 'MASTER_USER',
 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_SCHEDULER', 'MASTER_USER',
 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('HTP', 'MASTER_USER', 'EXECUTE',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('OWA', 'MASTER_USER', 'EXECUTE',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('WPG_DOCLOAD', 'MASTER_USER',
 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_CRYPTO', 'MASTER_USER',
 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_METADATA', 'MASTER_USER',
 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_SQL', 'MASTER_USER', 'EXECUTE',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('UTL_SMTP', 'MASTER_USER', 'EXECUTE',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBMS_NETWORK_ACL_ADMIN',
 'MASTER_USER', 'EXECUTE', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('SESSION_PRIVS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_USERS', 'MASTER_USER', 'SELECT',
 true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_NETWORK_ACL_PRIVILEGES',
 'MASTER_USER', 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_NETWORK_ACLS', 'MASTER_USER',
 'SELECT', true);
exec rdsadmin.rdsadmin_util.grant_sys_object('DBA_REGISTRY', 'MASTER_USER',
 'SELECT', true);
```

> **Note**
>
> The preceding commands apply to ORDS 22 and later.

6. Install the ORDS schema using the downloaded `ords` script. Specify directories to contain configuration files and log files. Oracle Corporation recommends not placing these directories inside the directory that contains the ORDS product software.

```
mkdir -p /home/apexuser/ords_config /home/apexuser/ords_logs

/home/apexuser/ORDS/bin/ords \
  --config /home/apexuser/ords_config \
  install --interactive --log-folder /home/apexuser/ords_logs
```

For DB instances running the container database (CDB) architecture, use ORDS 23.3 or higher and pass the `--pdb-skip-disable-lockdown` argument when installing ORDS.

```
/home/apexuser/ORDS/bin/ords \
  --config /home/apexuser/ords_config \
  install --interactive --log-folder /home/apexuser/ords_logs --pdb-skip-disable-
lockdown
```

The program prompts you for the following information. The default values are in brackets. For more information, see Introduction to Oracle REST data services in the Oracle documentation.

- `Choose the type of installation:`

  Choose **2** to install ORDS schemas in the database and create a database connection pool in the local ORDS configuration files.

- `Specify the database connection type to use. Enter number for [1] Basic [2] TNS [3] Custom URL:`

  Choose the desired connection type. This example assumes that you choose **1**.

- `Enter the name of the database server [localhost]:` ***`DB_instance_endpoint`***

  Choose the default or enter the correct value.

- `Enter the database listener port [1521]:` ***`DB_instance_port`***

  Choose the default **1521** or enter the correct value.

- `Enter the database service name [orcl]:`

  Enter the database name used by your RDS for Oracle DB instance.

- `Provide database user name with administrator privileges`

Enter the master user name for your RDS for Oracle DB instance.

- `Enter the database password for [username]:`

  Enter the master user password for your RDS for Oracle DB instance.

- `Enter the default tablespace for ORDS_METADATA and ORDS_PUBLIC_USER [SYSAUX]:`
- `Enter the temporary tablespace for ORDS_METADATA [TEMP]. Enter the default tablespace for ORDS_PUBLIC_USER [USERS]. Enter the temporary tablespace for ORDS_PUBLIC_USER [TEMP].`
- `Enter a number to select additional feature(s) to enable [1]:`
- `Enter a number to configure and start ORDS in standalone mode [1]:`

  Choose **2** to skip starting ORDS immediately in standalone mode.

- `Enter a number to select the protocol [1] HTTP`
- `Enter the HTTP port [8080]:`
- `Enter the APEX static resources location:`

  Enter the path to the Oracle APEX installation files (`/home/apexuser/apex/images`).

7. Set a password for the Oracle APEX `admin` user. To do this, use SQL*Plus to connect to your DB instance as the master user, and then run the following commands.

```
EXEC rdsadmin.rdsadmin_util.grant_apex_admin_role;
grant APEX_ADMINISTRATOR_ROLE to master;
@/home/apexuser/apex/apxchpwd.sql
```

   Replace *`master`* with your master user name. When the `apxchpwd.sql` script prompts you, enter a new `admin` password.

8. Run ORDS in standalone mode using the `ords` script with the `serve` command. For production deployments, consider using supported Java EE application servers such as Apache Tomcat or Oracle WebLogic Server. For more information, see Deploying and Monitoring Oracle REST Data Services in the Oracle Database documentation.

```
/home/apexuser/ORDS/bin/ords \
  --config /home/apexuser/ords_config serve \
  --port 8193 \
```

```
  --apex-images /home/apexuser/apex/images
```

If ORDS is running but unable to access the Oracle APEX installation, you might see the following error, particularly on non-CDB instances.

```
The procedure named apex_admin could not be accessed, it may not be declared,
 or the user executing this request may not have been granted execute privilege
 on the procedure, or a function specified by security.requestValidationFunction
 configuration property has prevented access.
```

To fix this error, change the request validation function used by ORDS by running the `ords` script with the `config` command. By default, ORDS uses the `ords_util.authorize_plsql_gateway` procedure, which is only supported on CDB instances. For non-CDB instances, you can change this procedure to the `wwv_flow_epg_include_modules.authorize` package. See the Oracle Database documentation and Oracle Support for best practices on configuring the proper request validation function for your use case.

9. Return to the Oracle APEX administration window in your browser and choose **Administration**. Next, choose **Application Express Internal Administration**. When you are prompted for credentials, enter the following information:

   - **User name** – `admin`
   - **Password** – the password you set using the `apxchpwd.sql` script

   Choose **Login**, and then set a new password for the `admin` user.

Your listener is now ready for use.

Upgrading and removing Oracle APEX

To upgrade or remove Oracle APEX, follow the instructions in this topic:

Topics

- Upgrading the Oracle APEX version
- Removing the APEX and APEX-DEV options

Upgrading the Oracle APEX version

> **Important**
>
> Back up your DB instance before you upgrade Oracle APEX. For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS and Testing an Oracle DB upgrade.

To upgrade Oracle APEX with your DB instance, do the following:

- Create a new option group for the upgraded version of your DB instance.
- Add the upgraded versions of the `APEX` and `APEX-DEV` options to the new option group. Be sure to include any other options that your DB instance uses. For more information, see Option group considerations.
- When you upgrade your DB instance, specify the new option group for your upgraded DB instance.

After you upgrade your version of Oracle APEX, the Oracle APEX schema for the previous version might still exist in your database. If you don't need it anymore, you can drop the old Oracle APEX schema from your database after you upgrade.

If you upgrade the Oracle APEX version and RESTful services were not configured in the previous Oracle APEX version, we recommend that you configure RESTful services. For more information, see Configuring RESTful services for Oracle APEX.

In some cases when you plan to do a major version upgrade of your DB instance, you might find that you're using an Oracle APEX version that isn't compatible with your target database version. In these cases, you can upgrade your version of Oracle APEX before you upgrade your DB instance. Upgrading Oracle APEX first can reduce the amount of time that it takes to upgrade your DB instance.

> **Note**
>
> After upgrading Oracle APEX, install and configure a listener for use with the upgraded version. For instructions, see Setting up Oracle APEX listener.

Removing the APEX and APEX-DEV options

You can remove the APEX and APEX-DEV options from a DB instance. To remove these options from your DB instance, do one of the following:

- To remove the APEX and APEX-DEV options from multiple DB instances, remove the options from the option group they belong to. This change affects all DB instances that use the option group. When you remove the options from an option group that is attached to multiple DB instances, a brief outage occurs while the DB instances are restarted.

  For more information, see Removing an option from an option group.
- To remove the APEX and APEX-DEV options from a single DB instance, modify the DB instance and specify a different option group that doesn't include these options. You can specify the default (empty) option group, or a different custom option group. When you remove the options, a brief outage occurs while your DB instance is automatically restarted.

  For more information, see Modifying an Amazon RDS DB instance.

When you remove the APEX and APEX-DEV options from a DB instance, the APEX schema is removed from your database.

Amazon EFS integration

Amazon Elastic File System (Amazon EFS) provides serverless, fully elastic file storage so that you can share file data without provisioning or managing storage capacity and performance. With Amazon EFS, you can create a file system and then mount it in your VPC through the NFS versions 4.0 and 4.1 (NFSv4) protocol. Then you can use the EFS file system like any other POSIX-compliant file system. For general information, see What is Amazon Elastic File System? and the AWS blog Integrate Amazon RDS for Oracle with Amazon EFS.

Topics

- Overview of Amazon EFS integration
- Configuring network permissions for RDS for Oracle integration with Amazon EFS
- Configuring IAM permissions for RDS for Oracle integration with Amazon EFS
- Adding the EFS_INTEGRATION option
- Configuring Amazon EFS file system permissions
- Transferring files between RDS for Oracle and an Amazon EFS file system
- Removing the EFS_INTEGRATION option
- Troubleshooting Amazon EFS integration

Overview of Amazon EFS integration

With Amazon EFS, you can transfer files between your RDS for Oracle DB instance and an EFS file system. For example, you can use EFS to support the following use cases:

- Share a file system between applications and multiple database servers.
- Create a shared directory for migration-related files, including transportable tablespace data files. For more information, see Migrating using Oracle transportable tablespaces.
- Store and share archived redo log files without allocating additional storage space on the server.
- Use Oracle Database utilities such as `UTL_FILE` to read and write files.

Advantages to Amazon EFS integration

When you choose an EFS file system over alternative data transfer solutions, you get the following benefits:

- You can transfer Oracle Data Pump files between Amazon EFS and your RDS for Oracle DB instance. You don't need to copy these files locally because Data Pump imports directly from the EFS file system. For more information, see Importing data into Oracle on Amazon RDS.
- Data migration is faster than using a database link.
- You avoid allocating storage space on your RDS for Oracle DB instance to hold the files.
- An EFS file systems can automatically scale storage without requiring you to provision it.
- Amazon EFS integration has no minimum fees or setup costs. You pay only for what you use.
- Amazon EFS integration supports two forms of encryption: encryption of data in transit and encryption at rest. Encryption of data in transit is enabled by default using TLS version 1.2. You can enable encryption of data at rest when creating an Amazon EFS file system. For more information, see Encrypting data at rest in the *Amazon Elastic File System User Guide*.

Requirements for Amazon EFS integration

Make sure that you meet the following requirements:

- Your database must run database version 19.0.0.0.ru-2022-07.rur-2022-07.r1 or higher.
- Your DB instance and your EFS file system must be in the same AWS Region, VPC, and AWS account. RDS for Oracle doesn't support cross-account and cross-Region access for EFS.
- Your VPC must have both **DNS Resolution** and **DNS Hostnames** enabled. For more information, see DNS attributes in your VPC in the *Amazon Virtual Private Cloud User Guide*.
- If you use a DNS name in the `mount` command, make sure your VPC configured to use the DNS server provided by Amazon. Custom DNS servers aren't supported.
- You must use non-RDS solutions to back up your EFS file system. RDS for Oracle doesn't support automated backups or manual DB snapshots of an EFS file system. For more information, see Backing up your Amazon EFS file systems.

Configuring network permissions for RDS for Oracle integration with Amazon EFS

For RDS for Oracle to integrate with Amazon EFS, make sure that your DB instance has network access to an EFS file system. For more information, see Controlling network access to Amazon EFS file systems for NFS clients in the *Amazon Elastic File System User Guide*.

Topics

- Controlling network access with security groups

- Controlling network access with file system policies

Controlling network access with security groups

You can control your DB instance access to EFS file systems using network layer security mechanisms such as VPC security groups. To allow access to an EFS file system for your DB instance, make sure that your EFS file system meets the following requirements:

- An EFS mount target exists in every Availability Zone used by an RDS for Oracle DB instance.

  An *EFS mount target* provides an IP address for an NFSv4 endpoint at which you can mount an EFS file system. You mount your file system using its DNS name, which resolves to the IP address of the EFS mount target in the used by the Availability Zone of your DB instance.

  You can configure DB instances in different AZs to use the same EFS file system. For Multi-AZ, you need a mount point for each AZ in your deployment. You might need to move a DB instance to a different AZ. For these reasons, we recommend that you create an EFS mount point in each AZ in your VPC. By default, when you create a new EFS file system using the console, RDS creates mount targets for all AZs.
- A security group is attached to the mount target.
- The security group has an inbound rule to allow the network subnet or security group of the RDS for Oracle DB instance on TCP/2049 (Type NFS).

For more information, see Creating Amazon EFS file systems and Creating and managing EFS mount targets and security groups in the *Amazon Elastic File System User Guide*.

Controlling network access with file system policies

Amazon EFS integration with RDS for Oracle works with the default (empty) EFS file system policy. The default policy doesn't use IAM to authenticate. Instead, it grants full access to any anonymous client that can connect to the file system using a mount target. The default policy is in effect whenever a user-configured file system policy isn't in effect, including at file system creation. For more information, see Default EFS file system policy in the *Amazon Elastic File System User Guide*.

To strengthen access to your EFS file system for all clients, including RDS for Oracle, you can configure IAM permissions. In this approach, you create a file system policy. For more information, see Creating file system policies in the *Amazon Elastic File System User Guide*.

Configuring IAM permissions for RDS for Oracle integration with Amazon EFS

By default, Amazon EFS integration feature doesn't use an IAM role: the `USE_IAM_ROLE` option setting is `FALSE`. To integrate RDS for Oracle with Amazon EFS and an IAM role, your DB instance must have IAM permissions to access an Amazon EFS file system.

Topics

- Step 1: Create an IAM role for your DB instance and attach your policy
- Step 2: Create a file system policy for your Amazon EFS file system
- Step 3: Associate your IAM role with your RDS for Oracle DB instance

Step 1: Create an IAM role for your DB instance and attach your policy

In this step, you create a role for your RDS for Oracle DB instance to allow Amazon RDS to access your EFS file system.

Console

To create an IAM role to allow Amazon RDS access to an EFS file system

1. Open the IAM Management Console.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. For **AWS service**, choose **RDS**.
5. For **Select your use case**, choose **RDS – Add Role to Database**.
6. Choose **Next**.
7. Don't add any permissions policies. Choose **Next**.
8. Set **Role name** to a name for your IAM role, for example `rds-efs-integration-role`. You can also add an optional **Description** value.
9. Choose **Create role**.

AWS CLI

To limit the service's permissions to a specific resource, we recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based trust relationships. This is the most effective way to protect against the confused deputy problem.

You might use both global condition context keys and have the `aws:SourceArn` value contain the account ID. In this case, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same statement.

- Use `aws:SourceArn` if you want cross-service access for a single resource.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

In the trust relationship, make sure to use the `aws:SourceArn` global condition context key with the full Amazon Resource Name (ARN) of the resources accessing the role.

The following AWS CLI command creates the role named *`rds-efs-integration-role`* for this purpose.

Example

For Linux, macOS, or Unix:

```
aws iam create-role \
   --role-name rds-efs-integration-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                 "aws:SourceAccount": my_account_ID,
                 "aws:SourceArn": "arn:aws:rds:Region:my_account_ID:db:dbname"
             }
         }
       }
     ]
   }'
```

For Windows:

```
aws iam create-role ^
   --role-name rds-efs-integration-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                 "aws:SourceAccount": my_account_ID,
                 "aws:SourceArn": "arn:aws:rds:Region:my_account_ID:db:dbname"
             }
         }
       }
     ]
   }'
```

For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.

Step 2: Create a file system policy for your Amazon EFS file system

In this step, you create a file system policy for your EFS file system.

To create or edit an EFS file system policy

1. Open the EFS Management Console.
2. Choose **File Systems**.
3. On the **File systems** page, choose the file system that you want to edit or create a file system policy for. The details page for that file system is displayed.
4. Choose the **File system policy** tab.

   If the policy is empty, then the default EFS file system policy is in use. For more information, see Default EFS file system policy in the *Amazon Elastic File System User Guide*.
5. Choose **Edit**. The **File system policy** page appears.
6. In **Policy editor**, enter a policy such as the following, and then choose **Save**.

JSON

```
{
    "Version":"2012-10-17",
    "Id": "ExamplePolicy01",
    "Statement": [
        {
            "Sid": "ExampleStatement01",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/rds-efs-integration-
role"
            },
            "Action": [
                "elasticfilesystem:ClientMount",
                "elasticfilesystem:ClientWrite",
                "elasticfilesystem:ClientRootAccess"
            ],
            "Resource": "arn:aws:elasticfilesystem:us-
east-1:123456789012:file-system/fs-1234567890abcdef0"
        }
    ]
}
```

Step 3: Associate your IAM role with your RDS for Oracle DB instance

In this step, you associate your IAM role with your DB instance. Be aware of the following requirements:

- You must have access to an IAM role with the required Amazon EFS permissions policy attached to it.
- You can associate only one IAM role with your RDS for Oracle DB instance at a time.
- The status of your instance must be **Available**.

For more information, see Identity and access management for Amazon EFS in the *Amazon Elastic File System User Guide*.

Console

To associate your IAM role with your RDS for Oracle DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Databases**.
3. If your database instance is unavailable, choose **Actions** and then **Start**. When the instance status shows **Started**, go to the next step.
4. Choose the Oracle DB instance name to display its details.
5. On the **Connectivity & security** tab, scroll down to the **Manage IAM roles** section at the bottom of the page.
6. Choose the role to add in the **Add IAM roles to this instance** section.
7. For **Feature**, choose **EFS_INTEGRATION**.
8. Choose **Add role**.

AWS CLI

The following AWS CLI command adds the role to an Oracle DB instance named *mydbinstance*.

Example

For Linux, macOS, or Unix:

```
aws rds add-role-to-db-instance \
   --db-instance-identifier mydbinstance \
   --feature-name EFS_INTEGRATION \
   --role-arn your-role-arn
```

For Windows:

```
aws rds add-role-to-db-instance ^
   --db-instance-identifier mydbinstance ^
   --feature-name EFS_INTEGRATION ^
   --role-arn your-role-arn
```

Replace *your-role-arn* with the role ARN that you noted in a previous step. `EFS_INTEGRATION` must be specified for the `--feature-name` option.

Adding the EFS_INTEGRATION option

To integrate Amazon RDS for Oracle with Amazon EFS, your DB instance must be associated with an option group that includes the `EFS_INTEGRATION` option.

Multiple Oracle DB instances that belong to the same option group share the same EFS file system. Different DB instances can access the same data, but access can be divided by using different Oracle directories. For more information see Transferring files between RDS for Oracle and an Amazon EFS file system.

Console

To configure an option group for Amazon EFS integration

1. Create a new option group or identify an existing option group to which you can add the `EFS_INTEGRATION` option.

   For information about creating an option group, see Creating an option group.
2. Add the `EFS_INTEGRATION` option to the option group. You need to specify the `EFS_ID` file system ID and set the `USE_IAM_ROLE` flag.

   For more information, see Adding an option to an option group.
3. Associate the option group with your DB instance in either of the following ways:

   - Create a new Oracle DB instance and associate the option group with it. For information about creating a DB instance, see Creating an Amazon RDS DB instance.
   - Modify an Oracle DB instance to associate the option group with it. For information about modifying an Oracle DB instance, see Modifying an Amazon RDS DB instance.

AWS CLI

To configure an option group for EFS integration

1. Create a new option group or identify an existing option group to which you can add the `EFS_INTEGRATION` option.

   For information about creating an option group, see Creating an option group.
2. Add the `EFS_INTEGRATION` option to the option group.

For example, the following AWS CLI command adds the `EFS_INTEGRATION` option to an option group named **`myoptiongroup`**.

Example

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
   --option-group-name myoptiongroup \
   --options "OptionName=EFS_INTEGRATION,OptionSettings=\
   [{Name=EFS_ID,Value=fs-1234567890abcdef0},{Name=USE_IAM_ROLE,Value=TRUE}]"
```

For Windows:

```
aws rds add-option-to-option-group ^
   --option-group-name myoptiongroup ^
   --options "OptionName=EFS_INTEGRATION,OptionSettings=^
   [{Name=EFS_ID,Value=fs-1234567890abcdef0},{Name=USE_IAM_ROLE,Value=TRUE}]"
```

3. Associate the option group with your DB instance in either of the following ways:

- Create a new Oracle DB instance and associate the option group with it. For information about creating a DB instance, see Creating an Amazon RDS DB instance.
- Modify an Oracle DB instance to associate the option group with it. For information about modifying an Oracle DB instance, see Modifying an Amazon RDS DB instance.

Configuring Amazon EFS file system permissions

By default, only the root user (UID `0`) has read, write, and execute permissions for a newly created EFS file system. For other users to modify the file system, the root user must explicitly grant them access. The user for the RDS for Oracle DB instance is in the `others` category. For more information, see Working with users, groups, and permissions at the Network File System (NFS) Level in the *Amazon Elastic File System User Guide*.

To allow your RDS for Oracle DB instance to read and write files on an EFS file system, do the following:

- Mount an EFS file system locally on your Amazon EC2 or on-premises instance.
- Configure fine grain permissions.

For example, to grant `other` users permissions to write to the EFS file system root, run `chmod 777` on this directory. For more information, see Example Amazon EFS file system use cases and permissions in the *Amazon Elastic File System User Guide*.

Transferring files between RDS for Oracle and an Amazon EFS file system

To transfer files between an RDS for Oracle instance and an Amazon EFS file system, create at least one Oracle directory and configure EFS file system permissions to control DB instance access.

Topics

- Creating an Oracle directory
- Transferring data to and from an EFS file system: examples

Creating an Oracle directory

To create an Oracle directory, use the procedure `rdsadmin.rdsadmin_util.create_directory_efs`. The procedure has the following parameters.

Parameter name	Data type	Defaul	Requir	Description
`p_directory_name`	VARCHAR2	–	Yes	The name of the Oracle directory.
`p_path_on_efs`	VARCHAR2	–	Yes	The path on the EFS file system. The prefix of the path name uses the pattern `/rdsefs-`*`fsid`*`/`, where *`fsid`* is a placeholder for your EFS file system ID. For example, if your EFS file system is named `fs-1234567890abcdef0` , and you create a subdirectory on this file system named `mydir`, you could specify the following value: `/rdsefs-fs-1234567890abcdef0/mydir`

Assume that you create a subdirectory named `/datapump1` on the EFS file system `fs-1234567890abcdef0`. The following example creates an Oracle directory `DATA_PUMP_DIR_EFS` that points to the `/datapump1` directory on the EFS file system. The file system path value for the `p_path_on_efs` parameter is prefixed with the string `/rdsefs-`.

```
BEGIN
  rdsadmin.rdsadmin_util.create_directory_efs(
    p_directory_name => 'DATA_PUMP_DIR_EFS',
    p_path_on_efs    => '/rdsefs-fs-1234567890abcdef0/datapump1');
END;
/
```

Transferring data to and from an EFS file system: examples

The following example uses Oracle Data Pump to export the table named `MY_TABLE` to file `datapump.dmp`. This file resides on an EFS file system.

```
DECLARE
  v_hdnl NUMBER;
BEGIN
  v_hdnl := DBMS_DATAPUMP.OPEN(operation => 'EXPORT', job_mode => 'TABLE',
 job_name=>null);
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'datapump.dmp',
    directory => 'DATA_PUMP_DIR_EFS',
    filetype  => dbms_datapump.ku$_file_type_dump_file);
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'datapump-exp.log',
    directory => 'DATA_PUMP_DIR_EFS',
    filetype  => dbms_datapump.ku$_file_type_log_file);
  DBMS_DATAPUMP.METADATA_FILTER(v_hdnl,'NAME_EXPR','IN (''MY_TABLE'')');
  DBMS_DATAPUMP.START_JOB(v_hdnl);
END;
/
```

The following example uses Oracle Data Pump to import the table named `MY_TABLE` from file `datapump.dmp`. This file resides on an EFS file system.

```
DECLARE
```

```
  v_hdnl NUMBER;
BEGIN
  v_hdnl := DBMS_DATAPUMP.OPEN(
    operation => 'IMPORT',
    job_mode  => 'TABLE',
    job_name  => null);
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'datapump.dmp',
    directory => 'DATA_PUMP_DIR_EFS',
    filetype  => dbms_datapump.ku$_file_type_dump_file );
  DBMS_DATAPUMP.ADD_FILE(
    handle    => v_hdnl,
    filename  => 'datapump-imp.log',
    directory => 'DATA_PUMP_DIR_EFS',
    filetype  => dbms_datapump.ku$_file_type_log_file);
  DBMS_DATAPUMP.METADATA_FILTER(v_hdnl,'NAME_EXPR','IN (''MY_TABLE'')');
  DBMS_DATAPUMP.START_JOB(v_hdnl);
END;
/
```

For more information, see Importing data into Oracle on Amazon RDS.

Removing the EFS_INTEGRATION option

The steps for removing the `EFS_INTEGRATION` option depend on whether you're removing the option from multiple DB instances or a single instance.

Number of DB instances	Action	Related information
Multiple	Remove the `EFS_INTEGRATION` option from the option group to which the DB instances belong. This change affects all instances that use the option group.	Removing an option from an option group
Single	Modify the DB instance and specify a different option group that doesn't include the `EFS_INTEGRATION` option. You can specify the default (empty) option group or a different custom option group.	Modifying an Amazon RDS DB instance

After you remove the EFS_INTEGRATION option, you can optionally delete the EFS file system that was connected to your DB instances.

Troubleshooting Amazon EFS integration

Your RDS for Oracle DB instance monitors the connectivity to an Amazon EFS file system. When monitoring detects an issue, it might try to correct the issue and publish an event in the RDS console. For more information, see Viewing Amazon RDS events.

Use the information in this section to help you diagnose and fix common issues when you work with Amazon EFS integration.

Notification	Description	Action
`The EFS for RDS Oracle instance` *instance_name* `isn't available on the primary host. NFS port 2049 of your EFS isn't reachable.`	The DB instance can't communicate with the EFS file system.	Make sure of the following: • The EFS file system exists. • The security group that is attached to the EFS mount target has an inbound rule to allow the security group or network subnet of the RDS for Oracle DB instance on TCP/2049 (Type NFS).
`The EFS isn't reachable.`	An error occurred during the installation of the `EFS_INTEGRATION` option.	Make sure of the following: • The EFS file system exists. • The security group that is attached to the EFS mount target has an inbound rule to allow the security group or network subnet of the RDS for Oracle DB instance on TCP/2049 (Type NFS). • The `enableDnsSupport` attribute is turned on for your VPC.

Notification	Description	Action
		• You are using the Amazon provided DNS server in your VPC. Amazon EFS integrati on doesn't work with a custom DHCP DNS.
`The associated role with your DB instance wasn't found.`	An error occurred during the installation of the `EFS_INTEGRATION` option.	Make sure that you associated an IAM role with your RDS for Oracle DB instance.
`The associated role with your DB instance wasn't found.`	An error occurred during the installation of the `EFS_INTEGRATION` option. RDS for Oracle was restored from a DB snapshot with the `USE_IAM_ROLE` option setting of `TRUE`.	Make sure that you associated an IAM role with your RDS for Oracle DB instance.
`The associated role with your DB instance wasn't found.`	An error occurred during the installation of the `EFS_INTEGRATION` option. RDS for Oracle was created from an all-in-one CloudForm ation template with the `USE_IAM_ROLE` option setting of `TRUE`.	As a workaround, complete the following steps: 1. Create a DB instance with the IAM role and default option group. 2. On a subsequent stack update, add the custom option group with the `EFS_INTEGRATION` option.
`PLS-00302: component 'CREATE_DIRECTORY_ EFS' must be declared`	This error can occur when you're using a version of RDS for Oracle that doesn't support Amazon EFS.	Make sure that you are using RDS for Oracle DB instance version 19.0.0.0.ru-2022-0 7.rur-2022-07.r1 or higher.

Notification	Description	Action
`Read access of your EFS is denied. Check your file system policy.`	Your DB instance can't read the EFS file system.	Make sure that your EFS file system allows read access through the IAM role or on the EFS file system level.
N/A	Your DB instance can't write to the EFS file system.	Take the following steps: 1. Make sure that your EFS file system is mounted on an Amazon EC2 instance. 2. Give the `others` group write access to your RDS user. The simplest technique is to run the `chmod 777` command on the top directory of the EFS file system.

Oracle Java virtual machine

Amazon RDS supports Oracle Java Virtual Machine (JVM) through the use of the `JVM` option. Oracle Java provides a SQL schema and functions that facilitate Oracle Java features in an Oracle database. For more information, see Introduction to Java in Oracle database in the Oracle documentation. You can use Oracle JVM with all versions of Oracle Database 21c (21.0.0) and Oracle Database 19c (19.0.0).

Considerations for Oracle JVM

Java implementation in Amazon RDS has a limited set of permissions. The master user is granted the `RDS_JAVA_ADMIN` role, which grants a subset of the privileges granted by the `JAVA_ADMIN` role. To list the privileges granted to the `RDS_JAVA_ADMIN` role, run the following query on your DB instance:

```
SELECT * FROM dba_java_policy
   WHERE grantee IN ('RDS_JAVA_ADMIN', 'PUBLIC')
   AND enabled = 'ENABLED'
   ORDER BY type_name, name, grantee;
```

Prerequisites for Oracle JVM

The following are prerequisites for using Oracle Java:

- Your DB instance must be of a large enough class. Oracle Java isn't supported for the db.t3.small DB instance classes. For more information, see DB instance classes.
- Your DB instance must have **Auto Minor Version Upgrade** enabled. This option enables your DB instance to receive minor DB engine version upgrades automatically when they become available. Amazon RDS uses this option to update your DB instance to the latest Oracle Patch Set Update (PSU) or Release Update (RU). For more information, see Modifying an Amazon RDS DB instance.

Best practices for Oracle JVM

The following are best practices for using Oracle Java:

- For maximum security, use the `JVM` option with Secure Sockets Layer (SSL). For more information, see Oracle Secure Sockets Layer.

- Configure your DB instance to restrict network access. For more information, see Scenarios for accessing a DB instance in a VPC and Working with a DB instance in a VPC.
- Update the configuration of your HTTPS endpoints to support TLSv1.2 if you meet the following conditions:
  - You use Oracle Java Virtual Machine (JVM) to connect an HTTPS endpoint over TLSv1 or TLSv1.1 protocols.
  - Your endpoint doesn't support the TLSv1.2 protocol.
  - You haven't applied the April 2021 release update to your Oracle DB.

  By updating your endpoint configuration, you ensure that the connectivity of the JVM to the HTTPS endpoint will continue to work. For more information about TLS changes in the Oracle JRE and JDK, see Oracle JRE and JDK Cryptographic Roadmap.

Adding the Oracle JVM option

The following is the general process for adding the `JVM` option to a DB instance:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with the DB instance.

There is a brief outage while the `JVM` option is added. After you add the option, you don't need to restart your DB instance. As soon as the option group is active, Oracle Java is available.

> **Note**
>
> During this outage, password verification functions are disabled briefly. You can also expect to see events related to password verification functions during the outage. Password verification functions are enabled again before the Oracle DB instance is available.

To add the JVM option to a DB instance

1. Determine the option group that you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

- For **Engine**, choose the DB engine used by the DB instance (**oracle-ee**, **oracle-se**, **oracle-se1**, or **oracle-se2**).
- For **Major engine version**, choose the version of your DB instance.

For more information, see Creating an option group.

2. Add the **JVM** option to the option group. For more information about adding options, see Adding an option to an option group.
3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.
4. Grant the required permissions to users.

   The Amazon RDS master user has the permissions to use the `JVM` option by default. If other users require these permissions, connect to the DB instance as the master user in a SQL client and grant the permissions to the users.

   The following example grants the permissions to use the `JVM` option to the `test_proc` user.

```
create user test_proc identified by password;
CALL dbms_java.grant_permission('TEST_PROC',
 'oracle.aurora.security.JServerPermission', 'LoadClassInPackage.*', '');
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

After the user is granted the permissions, the following query should return output.

```
select * from dba_java_policy where grantee='TEST_PROC';
```

> **Note**
>
> The Oracle user name is case-sensitive, and it usually has all uppercase characters.

Removing the Oracle JVM option

You can remove the `JVM` option from a DB instance. There is a brief outage while the option is removed. After you remove the `JVM` option, you don't need to restart your DB instance.

> **Warning**
>
> Removing the `JVM` option can result in data loss if the DB instance is using data types that were enabled as part of the option. Back up your data before proceeding. For more information, see Backing up, restoring, and exporting data.

To remove the `JVM` option from a DB instance, do one of the following:

- Remove the `JVM` option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
- Modify the DB instance and specify a different option group that doesn't include the `JVM` option. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Oracle Enterprise Manager

Amazon RDS supports Oracle Enterprise Manager (OEM). OEM is the Oracle product line for integrated management of enterprise information technology.

Amazon RDS supports OEM on Oracle Database 19c non-CDBs or CDBs. The following table describes the supported OEM options.

Option	Option ID	Supported OEM releases
OEM Database Express	`OEM`	OEM Database Express 19c
OEM Management Agent	`OEM_AGENT`	OEM Cloud Control for 13c

Note

You can use OEM Database or OEM Management Agent, but not both.

Oracle Enterprise Manager Database Express

Amazon RDS supports Oracle Enterprise Manager Database Express (EM Express) through the use of the OEM option. Amazon RDS supports EM Express for Oracle Database 19c using the CDB or non-CDB architecture.

EM Express is a web-based database management tool included in your database and is only available when it is open. It supports key performance management and basic database administration functions. For more information, see Introduction to Oracle Enterprise Manager Database Express in the Oracle Database documentation.

> **Note**
>
> EM Express isn't supported on the db.t3.small DB instance class. For more information about DB instance classes, see RDS for Oracle DB instance classes.

OEM option settings

Amazon RDS supports the following settings for the OEM option.

Option setting	Valid values	Description
Port	An integer value	The port on the RDS for Oracle DB instance that listens for EM Express. The default is 5500.
Security Groups	—	A security group that has access to **Port**.

Step 1: Adding the OEM option

The general process for adding the OEM option to a DB instance is the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with your DB instance.

When you add the OEM option, a brief outage occurs while your DB instance is automatically restarted.

To add the OEM option to a DB instance

1. Determine the option group you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   a. For **Engine** choose the oracle edition for your DB instance.

   b. For **Major engine version** choose the version of your DB instance.

   For more information, see Creating an option group.

2. Add the OEM option to the option group, and configure the option settings. For more information about adding options, see Adding an option to an option group. For more information about each setting, see OEM option settings.

   > **Note**
   >
   > If you add the OEM option to an existing option group that is already attached to one or more DB instances, a brief outage occurs while all the DB instances are automatically restarted.

3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, apply the option group by modifying the instance and attaching the new option group. When you add the OEM option, a brief outage occurs while your DB instance is automatically restarted. For more information, see Modifying an Amazon RDS DB instance.

> **Note**
>
> You can also use the AWS CLI to add the OEM option. For examples, see Adding an option to an option group.

Step 2: (CDB only) Unlocking the DBSNMP user account

If your DB instance uses the CDB architecture, you must log in to EM Express as DBSNMP. in a CDB, DBSNMP is a common user. By default, this account is locked. If your DB instance doesn't use the CDB architecture, skip this step.

To unlock the DBSNMP user account in a CDB instance

1. In SQL*Plus or another Oracle SQL application, log in to your DB instance as your master user.
2. Run the following stored procedure to unlock the DBSNMP account:

   ```
   EXEC rdsadmin.rdsadmin_util.reset_oem_agent_password('new_password');
   ```

   If you receive an error stating that the procedure doesn't exist, reboot your CDB instance to install it automatically. For more information, see Rebooting a DB instance.

Step 3: Accessing EM Express through your browser

When you access EM Express from your web browser, a login window appears that prompts you for a user name and password.

To access EM Express through your browser

1. Identify the endpoint and EM Express port for your Amazon RDS DB instance. For information about finding the endpoint for your Amazon RDS DB instance, see Finding the endpoint of your RDS for Oracle DB instance.
2. Enter a URL in your browser locator bar using the following format.

   ```
   https://endpoint.rds.amazonaws.com:port/em
   ```

   For example, if the endpoint for your Amazon RDS DB instance is `mydb.a1bcde234fgh.us-east-1.rds.amazonaws.com`, and your EM Express port is 1158, then use the following URL to access EM Express.

   ```
   https://mydb.f9rbfa893tft.us-east-1.rds.amazonaws.com:1158/em
   ```

3. When prompted for your login details, do one of the following actions, depending on your database architecture:

Your database is a non-CDB.

Type the master user name and master password for your DB instance.

Your database is a CDB.

Enter `DBSNMP` for the user and the `DBSNMP` password. Leave the `Container` field empty.

Modifying OEM Database settings

After you enable OEM Database, you can modify the Security Groups setting for the option.

You can't modify the OEM port number after you have associated the option group with a DB instance. To change the OEM port number for a DB instance, do the following:

1. Create a new option group.
2. Add the OEM option with the new port number to the new option group.
3. Remove the existing option group from the DB instance.
4. Add the new option group to the DB instance.

For more information about how to modify option settings, see Modifying an option setting. For more information about each setting, see OEM option settings.

Running OEM Database Express tasks

You can use Amazon RDS procedures to run certain OEM Database Express tasks. By running these procedures, you can do the tasks listed following.

> **Note**
>
> OEM Database Express tasks run asynchronously.

Tasks

- Switching the website front end for OEM Database Express to Adobe Flash
- Switching the website front end for OEM Database Express to Oracle JET

Switching the website front end for OEM Database Express to Adobe Flash

> **Note**
>
> This task is available only for Oracle Database 19c non-CDBs.

Starting with Oracle Database 19c, Oracle has deprecated the former OEM Database Express user interface, which was based on Adobe Flash. Instead, OEM Database Express now uses an interface built with Oracle JET. If you experience difficulties with the new interface, you can switch back to the deprecated Flash-based interface. Difficulties you might experience with the new interface include being stuck on a `Loading` screen after logging in to OEM Database Express. You might also miss certain features that were present in the Flash-based version of OEM Database Express.

To switch the OEM Database Express website front end to Adobe Flash, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_tasks.em_express_frontend_to_flash`. This procedure is equivalent to the `execemx emx` SQL command.

Security best practices discourage the use of Adobe Flash. Although you can revert to the Flash-based OEM Database Express, we recommend the use of the JET-based OEM Database Express websites if possible. If you revert to using Adobe Flash and want to switch back to using Oracle JET, use the `rdsadmin.rdsadmin_oem_tasks.em_express_frontend_to_jet` procedure. After an Oracle database upgrade, a newer version of Oracle JET might resolve JET-related issues in OEM Database Express. For more information about switching to Oracle JET, see Switching the website front end for OEM Database Express to Oracle JET.

> **Note**
>
> Running this task from the source DB instance for a read replica also causes the read replica to switch its OEM Database Express website front ends to Adobe Flash.

The following procedure invocation creates a task to switch the OEM Database Express website to Adobe Flash and returns the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_tasks.em_express_frontend_to_flash() as TASK_ID from DUAL;
```

You can view the result by displaying the task's output file.

```
SELECT text FROM table(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-task-id.log'));
```

Replace *task-id* with the task ID returned by the procedure. For more information about the Amazon RDS procedure `rdsadmin.rds_file_util.read_text_file`, see Reading files in a DB instance directory

You can also view the contents of the task's output file in the AWS Management Console by searching the log entries in the **Logs & events** section for the `task-id`.

Switching the website front end for OEM Database Express to Oracle JET

> **Note**
>
> This task is available only for Oracle Database 19c non-CDBs.

To switch the OEM Database Express website front end to Oracle JET, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_tasks.em_express_frontend_to_jet`. This procedure is equivalent to the `execemx omx` SQL command.

By default, the OEM Database Express websites for Oracle DB instances running 19c or later use Oracle JET. If you used the `rdsadmin.rdsadmin_oem_tasks.em_express_frontend_to_flash` procedure to switch the OEM Database Express website front end to Adobe Flash, you can switch back to Oracle JET. To do this, use the `rdsadmin.rdsadmin_oem_tasks.em_express_frontend_to_jet` procedure. For more information about switching to Adobe Flash, see Switching the website front end for OEM Database Express to Adobe Flash.

> **Note**
>
> Running this task from the source DB instance for a read replica also causes the read replica to switch its OEM Database Express website front ends to Oracle JET.

The following procedure invocation creates a task to switch the OEM Database Express website to Oracle JET and returns the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_tasks.em_express_frontend_to_jet() as TASK_ID from DUAL;
```

You can view the result by displaying the task's output file.

```
SELECT text FROM table(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-task-id.log'));
```

Replace *task-id* with the task ID returned by the procedure. For more information about the Amazon RDS procedure `rdsadmin.rds_file_util.read_text_file`, see Reading files in a DB instance directory

You can also view the contents of the task's output file in the AWS Management Console by searching the log entries in the **Logs & events** section for the `task-id`.

Removing the OEM Database option

You can remove the OEM option from a DB instance. When you remove the OEM option, a brief outage occurs while your instance is automatically restarted. Therefore, after you remove the OEM option, you don't need to restart your DB instance.

To remove the OEM option from a DB instance, do one of the following:

- Remove the OEM option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
- Modify the DB instance and specify a different option group that doesn't include the OEM option. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Oracle Management Agent for Enterprise Manager Cloud Control

Oracle Enterprise Manager (OEM) Management Agent is a software component that monitors targets running on hosts and communicates that information to the middle-tier Oracle Management Service (OMS). Amazon RDS supports Management Agent through the use of the `OEM_AGENT` option.

For more information, see Overview of Oracle Enterprise Manager cloud control 12c and Overview of Oracle Enterprise Manager cloud control 13c in the Oracle documentation.

Topics

- Requirements for Management Agent
- OMS host communication prerequisites
- Limitations for Management Agent
- Option settings for Management Agent
- Enabling the Management Agent option for your DB instance
- Removing the Management Agent option
- Performing database tasks with the Management Agent

Requirements for Management Agent

Following are general requirements for using Management Agent:

- Your DB instance must run Oracle Database 19c (19.0.0.0). You can use either the CDB or non-CDB architecture.
- You must use an Oracle Management Service (OMS) that is configured to connect to your DB instance. Note the following OMS requirements:
  - Management Agent versions 13.5.0.0.v2 and 13.5.0.0.v3 require OMS version 13.5.0.23 or 24.1.
  - Management Agent version 13.5.0.0.v1 requires OMS version 13.5.0.0 or 24.1.
  - Management Agent versions 13.4.0.9.v1 and 13.4.0.9.v2 require OMS version 13.4.0.9 or later and patch 32198287.
- In most cases, you must configure your VPC to allow connections from OMS to your DB instance. If you aren't familiar with Amazon Virtual Private Cloud (Amazon VPC), we recommend that you complete the steps in Tutorial: Create a VPC for use with a DB instance (IPv4 only) before continuing.

- You can use Management Agent with Oracle Enterprise Manager Cloud Control for 12c and 13c. Ensure that you have sufficient storage space for your OEM release:
  - At least 8.5 GiB for OEM 13c Release 5
  - At least 8.5 GiB for OEM 13c Release 4
  - At least 8.5 GiB for OEM 13c Release 3
  - At least 5.5 GiB for OEM 13c Release 2
  - At least 4.5 GiB for OEM 13c Release 1
  - At least 2.5 GiB for OEM 12c
- If you're using Management Agent versions `OEM_AGENT 13.2.0.0.v3` and `13.3.0.0.v2`, and if you want to use TCPS connectivity, follow the instructions in Configuring third party CA certificates for communication with target databases in the Oracle documentation. Also, update the JDK on your OMS by following the instructions in the Oracle document with the Oracle Doc ID 2241358.1. This step ensures that OMS supports all the cipher suites that the database supports.

  > **Note**
  >
  > TCPS connectivity between the Management Agent and the DB instance is supported for Management Agent `OEM_AGENT 13.2.0.0.v3`, `13.3.0.0.v2`, `13.4.0.9.v1`, and higher versions.

OMS host communication prerequisites

Make sure that your OMS host and your Amazon RDS DB instance can communicate. Do the following:

- To connect from the Management Agent to your OMS host when your OMS host is behind a firewall, add the IP addresses of your DB instances to the firewall. Make sure the firewall for the OMS allows the following network traffic:

  From the OMS host to your DB instance

  Configure a one-way firewall rule that allows traffic from the OMS host to the database listener port (default 1521) and the OEM Agent port (default 3872).

From your DB instance to the OMS host

Configure a one-way firewall rule that allows traffic from the DB instance to the OMS HTTP port (default 4903).

- To connect from your OMS to the Management Agent, if your OMS has a publicly resolvable host name, add the OMS address to a security group. Your security group must have inbound rules that allow access to the DB listener port and the Management Agent port. For an example of creating a security and adding inbound rules, see Tutorial: Create a VPC for use with a DB instance (IPv4 only).
- To connect from your OMS to the Management Agent, if your OMS doesn't have a publicly resolvable host name, use one of the following:
  - If your OMS is hosted on an Amazon Elastic Compute Cloud (Amazon EC2) instance in a private VPC, you can set up VPC peering to connect from OMS to Management Agent. For more information, see A DB instance in a VPC accessed by an EC2 instance in a different VPC.
  - If your OMS is hosted on-premises, you can set up a VPN connection to allow access from OMS to Management Agent. For more information, see A DB instance in a VPC accessed by a client application through the internet or VPN connections.
- To connect OEM Management Agent version 13.5.0.0 (v1-v3) to a 24.1 OMS host, set `MINIMUM_TLS_VERSION` to use the TLS 1.2 protocol `TLSv1.2` in your configuration options.

Limitations for Management Agent

Following are some limitations to using Management Agent:

- You can't provide custom Oracle Management Agent images.
- Administrative tasks such as job execution and database patching, that require host credentials, aren't supported.
- Host metrics and the process list aren't guaranteed to reflect the actual system state. Thus, you shouldn't use OEM to monitor the root file system or mount point file system. For more information about monitoring the operating system, see Monitoring OS metrics with Enhanced Monitoring.
- Autodiscovery isn't supported. You must manually add database targets.
- OMS module availability depends on your database edition. For example, the database performance diagnosis and tuning module is only available for Oracle Database Enterprise Edition.

- Management Agent consumes additional memory and computing resources. If you experience performance problems after enabling the `OEM_AGENT` option, we recommend that you scale up to a larger DB instance class. For more information, see DB instance classes and Modifying an Amazon RDS DB instance.
- The user running the `OEM_AGENT` on the Amazon RDS host doesn't have operating system access to the alert log. Thus, you can't collect metrics for `DB Alert Log` and `DB Alert Log Error Status` in OEM.

Option settings for Management Agent

Amazon RDS supports the following settings for the Management Agent option.

Option setting	Require	Valid values	Description

Version (`AGENT_VERSION`)	Yes	`13.5.0.0.v3`
		`13.5.0.0.v2`
		`13.5.0.0.v1`
		`13.4.0.9.v2`
		`13.4.0.9.v1`
		`13.3.0.0.v2`
		`13.3.0.0.v1`
		`13.2.0.0.v3`
			The version of the Management Agent software. The minimum supported version is `13.1.0.0.v1` .
			The AWS CLI option name is `OptionVersion` .
			Note
			In the AWS GovCloud (US) Regions, 13.1 versions aren't available.

Option setting	Require	Valid values	Description

		`13.2.0.0.v2`
		
		`13.2.0.0.v1`
		
		`13.1.0.0.v1`	

Port (`AGENT_PORT`)	Yes	An integer value	The port on the DB instance that listens for the OMS host. The default is 3872. Your OMS host must belong to a security group that has access to this port. The AWS CLI option name is `Port`.
Security Groups	Yes	Existing security groups	A security group that has access to **Port**. Your OMS host must belong to this security group. The AWS CLI option name is `VpcSecurityGroupMemberships` or `DBSecurityGroupMemberships`.
OMS_HOST	Yes	A string value, for example *`my.example.oms`*	The publicly accessible host name or IP address of the OMS. The AWS CLI option name is `OMS_HOST`.

Option setting	Require	Valid values	Description
OMS_PORT	Yes	An integer value	The HTTPS upload port on the OMS Host that listens for the Management Agent. To determine the HTTPS upload port, connect to the OMS host, and run the following command (which requires the SYSMAN password): `emctl status oms -details` The AWS CLI option name is `OMS_PORT`.
AGENT_REGISTRATION_PASSWORD	Yes	A string value	The password that the Management Agent uses to authenticate itself with the OMS. We recommend that you create a persistent password in your OMS before enabling the `OEM_AGENT` option. With a persistent password you can share a single Management Agent option group among multiple Amazon RDS databases. The AWS CLI option name is `AGENT_REGISTRATION_PASSWORD`.
ALLOW_TLS_ONLY	No	`true`, `false` (default)	A value that configures the OEM Agent to support only the `TLSv1` protocol while the agent listens as a server. This setting is no longer supported. Management Agent versions 13.1.0.0. v1 and higher support Transport Layer Security (TLS) by default.

Option setting	Require	Valid values	Description
MINIMUM_TLS_VERSION	No	`TLSv1` (default), `TLSv1.2`	A value that specifies the minimum TLS version supported by the OEM Agent while the agent listens as a server. Desupported agent versions only support the `TLSv1` setting. To connect 13.5.0.0 (v1-v3) to a 24.1 OMS host, set this to `TLSv1.2`.
TLS_CIPHER_SUITE	No	See Option settings for Management Agent.	A value that specifies the TLS cipher suite used by the OEM Agent while the agent listens as a server.

The following table lists the TLS cipher suites that the Management Agent option supports.

Cipher suite	Agent version supported	FedRAMP compliant
TLS_RSA_WITH_AES_128_CBC_SHA	All	No
TLS_RSA_WITH_AES_128_CBC_SHA256	13.1.0.0.v1 and higher	No
TLS_RSA_WITH_AES_256_CBC_SHA	13.2.0.0.v3 and higher	No
TLS_RSA_WITH_AES_256_CBC_SHA256	13.2.0.0.v3 and higher	No
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	13.2.0.0.v3 and higher	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	13.2.0.0.v3 and higher	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	13.2.0.0.v3 and higher	Yes

Cipher suite	Agent version supported	FedRAMP compliant
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	13.2.0.0.v3 and higher	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	13.4.0.9.v1 and higher	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	13.4.0.9.v1 and higher	Yes

Certificate compatibility with cipher suites

RDS for Oracle supports both RSA and Elliptic Curve Digital Signature Algorithm (ECDSA) certificates. When you configure the OEM Agent option for your DB instance, you must ensure that the cipher suites you specify in the `TLS_CIPHER_SUITE` option setting are compatible with the certificate type used by your DB instance.

The following table shows the compatibility between certificate types and cipher suites:

Certificate type	Compatible cipher suites	Incompatible cipher suites
RSA certificates (rds-ca-2019, rds-ca-rsa2048-g1, rds-ca-rsa4096-g1)	TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

Certificate type	Compatible cipher suites	Incompatible cipher suites
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	
ECDSA certificates (rds-ca-ecc384-g1)	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

When you specify a cipher suite in the `TLS_CIPHER_SUITE` option setting, make sure it is compatible with the certificate type used by your DB instance. If you attempt to associate an option group with an OEM Agent option that contains a cipher suite incompatible with the certificate type of a DB instance, the operation will fail with an error message indicating the incompatibility.

Enabling the Management Agent option for your DB instance

To enable the Management Agent option, use the following steps:

Topics

- Step 1: Adding the Management Agent option to your DB instance
- Step 2: Unlocking the DBSNMP user account
- Step 3: Adding your targets to the Management Agent console

Step 1: Adding the Management Agent option to your DB instance

To add the Management Agent option to your DB instance, do the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with the DB instance.

If you encounter errors, check My Oracle Support documents for information about resolving specific problems.

After you add the Management Agent option, you don't need to restart your DB instance. As soon as the option group is active, the OEM Agent is active.

If your OMS host is using an untrusted third-party certificate, Amazon RDS returns the following error.

```
You successfully installed the OEM_AGENT option. Your OMS host is using an untrusted
 third party certificate.
Configure your OMS host with the trusted certificates from your third party.
```

If this error is returned, the Management Agent option isn't enabled until the problem is corrected. For information about correcting the problem, see the My Oracle Support document 2202569.1.

Console

To add the Management Agent option to your DB instance

1. Determine the option group you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   a. For **Engine** choose the oracle edition for your DB instance.

   b. For **Major engine version** choose the version of your DB instance.

   For more information, see Creating an option group.
2. Add the **OEM_AGENT** option to the option group, and configure the option settings. For more information about adding options, see Adding an option to an option group. For more information about each setting, see Option settings for Management Agent.
3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

AWS CLI

The following example uses the AWS CLI add-option-to-option-group command to add the `OEM_AGENT` option to an option group called `myoptiongroup`.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name "myoptiongroup" \
    --options
 OptionName=OEM_AGENT,OptionVersion=13.1.0.0.v1,Port=3872,VpcSecurityGroupMemberships=sg-123456
{Name=OMS_PORT,Value=4903},{Name=AGENT_REGISTRATION_PASSWORD,Value=password}] \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name "myoptiongroup" ^
    --options
 OptionName=OEM_AGENT,OptionVersion=13.1.0.0.v1,Port=3872,VpcSecurityGroupMemberships=sg-123456
{Name=OMS_PORT,Value=4903},{Name=AGENT_REGISTRATION_PASSWORD,Value=password}] ^
    --apply-immediately
```

Step 2: Unlocking the DBSNMP user account

The Management Agent uses the DBSNMP user account to connect to the database and report issues to Oracle Enterprise Manager. In a CDB, DBSNMP is a common user. This user account is necessary for both the Management Agent and OEM Database Express. By default, this account is locked. The procedure for unlocking this account differs depending on whether your database uses the non-CDB or CDB architecture.

To unlock the DBSNMP user account

1. In SQL*Plus or another Oracle SQL application, log in to your DB instance as your master user.
2. Do either of the following actions, depending on the database architecture:

   Your database is a non-CDB.

   Run the following SQL statement:

   ```
   ALTER USER dbsnmp IDENTIFIED BY new_password ACCOUNT UNLOCK;
   ```

   Your database is a CDB.

   Run the following stored procedure to unlock the DBSNMP account:

   ```
   EXEC rdsadmin.rdsadmin_util.reset_oem_agent_password('new_password');
   ```

   If you receive an error stating that the procedure doesn't exist, reboot your CDB instance to install it automatically. For more information, see Rebooting a DB instance.

Step 3: Adding your targets to the Management Agent console

To add a DB instance as a target, make sure you know the endpoint and port. For information about finding the endpoint for your Amazon RDS DB instance, see Finding the endpoint of your

RDS for Oracle DB instance. If your database uses the CDB architecture, then add the CDB$ROOT container separately as a target.

To add targets to the Management Agent console

1. In your OMS console, choose **Setup**, **Add Target**, **Add Targets Manually**.
2. Choose **Add Targets Declaratively by Specifying Target Monitoring Properties**.
3. For **Target Type**, choose **Database Instance**.
4. For **Monitoring Agent**, choose the agent with the identifier that is the same as your RDS DB instance identifier.
5. Choose **Add Manually**.
6. Enter the endpoint for your Amazon RDS DB instance, or choose it from the host name list. Make sure that the specified host name matches the endpoint of the Amazon RDS DB instance.
7. Specify the following database properties:
   - For **Target name**, enter a name.
   - For **Database system name**, enter a name.
   - For **Monitor username**, enter **dbsnmp**.
   - For **Monitor password**, enter the password from Step 2: Unlocking the DBSNMP user account.
   - For **Role**, enter **normal**.
   - For **Oracle home path**, enter **/oracle**.
   - For **Listener Machine name**, the agent identifier already appears.
   - For **Port**, enter the database port. The RDS default port is 1521.
   - For **Database name**, enter the name of your database. If your database is a CDB, this name is RDSCDB.
8. Choose **Test Connection**.
9. Choose **Next**. The target database appears in your list of monitored resources.

Removing the Management Agent option

You can remove the OEM Agent from a DB instance. After you remove the OEM Agent, you don't need to restart your DB instance.

To remove the OEM Agent from a DB instance, do one of the following:

- Remove the OEM Agent option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
- Modify the DB instance and specify a different option group that doesn't include the OEM Agent option. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Performing database tasks with the Management Agent

You can use Amazon RDS procedures to run certain EMCTL commands on the Management Agent. By running these procedures, you can do the tasks listed following.

> **Note**
>
> Tasks are executed asynchronously.

Tasks

- Securing the Management Agent
- Getting the status of the Management Agent
- Restarting the Management Agent
- Listing the targets monitored by the Management Agent
- Listing the collection threads monitored by the Management Agent
- Clearing the Management Agent state
- Making the Management Agent upload its OMS
- Pinging the OMS
- Viewing the status of an ongoing task

Securing the Management Agent

To secure the Management Agent, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_agent_tasks.secure_oem_agent`. This procedure is equivalent to running the `emctl secure agent` command.

The following procedure creates a task to secure the Management Agent and returns the ID of the task:

```
SELECT rdsadmin.rdsadmin_oem_agent_tasks.secure_oem_agent as TASK_ID from DUAL;
```

To display the output file for the task and view the result, see Viewing the status of an ongoing task.

Getting the status of the Management Agent

To get the status of the Management Agent, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_agent_tasks.get_status_oem_agent`. This procedure is equivalent to the `emctl status agent` command.

The following procedure creates a task to get the Management Agent's status and returns the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_agent_tasks.get_status_oem_agent() as TASK_ID from DUAL;
```

To display the output file for the task and view the result, see Viewing the status of an ongoing task.

Restarting the Management Agent

To restart the Management Agent, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_agent_tasks.restart_oem_agent`. This procedure is equivalent to running the `emctl stop agent` and `emctl start agent` commands.

The following procedure creates a task to restart the Management Agent and returns the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_agent_tasks.restart_oem_agent as TASK_ID from DUAL;
```

To display the output file for the task and view the result, see Viewing the status of an ongoing task.

Listing the targets monitored by the Management Agent

To list the targets monitored by the Management Agent, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_agent_tasks.list_targets_oem_agent`. This procedure is equivalent to running the `emctl config agent listtargets` command.

The following procedure creates a task to list the targets monitored by the Management Agent and returns the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_agent_tasks.list_targets_oem_agent as TASK_ID from DUAL;
```

To display the output file for the task and view the result, see Viewing the status of an ongoing task.

Listing the collection threads monitored by the Management Agent

To list of all the running, ready, and scheduled collection threads monitored by the Management Agent, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_agent_tasks.list_clxn_threads_oem_agent`. This procedure is equivalent to the `emctl status agent scheduler` command.

The following procedure creates a task to list the collection threads and returns the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_agent_tasks.list_clxn_threads_oem_agent() as TASK_ID from
 DUAL;
```

To display the output file for the task and view the result, see Viewing the status of an ongoing task.

Clearing the Management Agent state

To clear the Management Agent's state, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_agent_tasks.clearstate_oem_agent`. This procedure is equivalent to running the `emctl clearstate agent` command.

The following procedure creates a task that clears the Management Agent's state and returns the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_agent_tasks.clearstate_oem_agent() as TASK_ID from DUAL;
```

To display the output file for the task and view the result, see Viewing the status of an ongoing task.

Making the Management Agent upload its OMS

To make the Management Agent upload the Oracle Management Server (OMS) associated with it, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_agent_tasks.upload_oem_agent`. This procedure is equivalent to running the `emclt upload agent` command.

The following procedure creates a task that makes the Management Agent upload its associated OMS and return the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_agent_tasks.upload_oem_agent() as TASK_ID from DUAL;
```

To display the output file for the task and view the result, see Viewing the status of an ongoing task.

Pinging the OMS

To ping the Management Agent's OMS, run the Amazon RDS procedure `rdsadmin.rdsadmin_oem_agent_tasks.ping_oms_oem_agent`. This procedure is equivalent to running the `emctl pingOMS` command.

The following procedure creates a task that pings the Management Agent's OMS and returns the ID of the task.

```
SELECT rdsadmin.rdsadmin_oem_agent_tasks.ping_oms_oem_agent() as TASK_ID from DUAL;
```

To display the output file for the task and view the result, see Viewing the status of an ongoing task.

Viewing the status of an ongoing task

You can view the status of an ongoing task in a bdump file. The bdump files are located in the `/rdsdbdata/log/trace` directory. Each bdump file name is in the following format.

```
dbtask-task-id.log
```

When you want to monitor a task, replace *`task-id`* with the ID of the task that you want to monitor.

To view the contents of bdump files, run the Amazon RDS procedure `rdsadmin.rds_file_util.read_text_file`. The following query returns the contents of the `dbtask-1546988886389-2444.log` bdump file.

```
SELECT text FROM
 table(rdsadmin.rds_file_util.read_text_file('BDUMP','dbtask-1546988886389-2444.log'));
```

For more information about the Amazon RDS procedure `rdsadmin.rds_file_util.read_text_file`, see Reading files in a DB instance directory.

Oracle Label Security

Amazon RDS supports Oracle Label Security for the Enterprise Edition of Oracle Database through the use of the OLS option.

Most database security controls access at the object level. Oracle Label Security provides fine-grained control of access to individual table rows. For example, you can use Label Security to enforce regulatory compliance with a policy-based administration model. You can use Label Security policies to control access to sensitive data, and restrict access to only users with the appropriate clearance level. For more information, see Introduction to Oracle Label Security in the Oracle documentation.

Topics

- Requirements for Oracle Label Security
- Considerations when using Oracle Label Security
- Adding the Oracle Label Security option
- Troubleshooting

Requirements for Oracle Label Security

Familiarize yourself with the following requirements for Oracle Label Security:

- Your DB instance must use the Bring Your Own License model. For more information, see RDS for Oracle licensing options.
- You must have a valid license for Oracle Enterprise Edition with Software Update License and Support.
- Your Oracle license must include the Label Security option.

Considerations when using Oracle Label Security

To use Oracle Label Security, you create policies that control access to specific rows in your tables. For more information, see Creating an Oracle Label Security policy in the Oracle documentation.

Consider the following:

- Oracle Label Security is a permanent and persistent option. Because the option is permanent, you can't remove it from an option group. If you add Oracle Label Security to an option group

and associate it with your DB instance, you can later associate a different option group with your DB instance, but this group must also contain the Oracle Label Security option.

- When you work with Label Security, you perform all actions as the LBAC_DBA role. The master user for your DB instance is granted the LBAC_DBA role. You can grant the LBAC_DBA role to other users so that they can administer Label Security policies.
- Make sure to grant access to the OLS_ENFORCEMENT package to any new users who require access to Oracle Label Security. To grant access to the OLS_ENFORCEMENT package, connect to the DB instance as the master user and run the following SQL statement:

```
GRANT ALL ON LBACSYS.OLS_ENFORCEMENT TO username;
```

- You can configure Label Security through Oracle Enterprise Manager (OEM) Cloud Control. Amazon RDS supports OEM Cloud Control through the Management Agent option. For more information, see Oracle Management Agent for Enterprise Manager Cloud Control.

Adding the Oracle Label Security option

The general process for adding the Oracle Label Security option to a DB instance is the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.

> **Important**
>
> Oracle Label Security is a permanent and persistent option.

3. Associate the option group with the DB instance.

After you add the Label Security option, as soon as the option group is active, Label Security is active.

To add the label security option to a DB instance

1. Determine the option group you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   a. For **Engine**, choose **oracle-ee**.

b. For **Major engine version**, choose the version of your DB instance.

For more information, see Creating an option group.

2. Add the **OLS** option to the option group. For more information about adding options, see Adding an option to an option group.

> **Important**
>
> If you add Label Security to an existing option group that is already attached to one or more DB instances, all the DB instances are restarted.

3. Apply the option group to a new or existing DB instance:

- For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. When you add the Label Security option to an existing DB instance, a brief outage occurs while your DB instance is automatically restarted. For more information, see Modifying an Amazon RDS DB instance.

Troubleshooting

The following are issues you might encounter when you use Oracle Label Security.

Issue	Troubleshooting suggestions
When you try to create a policy, you see an error message similar to the following : `insufficient authorization for the SYSDBA package`.	A known issue with Oracle's Label Security feature prevents users with usernames of 16 or 24 characters from running Label Security commands. You can create a new user with a different number of characters, grant LBAC_DBA to the new user, log in as the new user, and run the OLS commands as the new user. For additional information, contact Oracle Support.

Oracle Locator

Amazon RDS supports Oracle Locator through the use of the LOCATOR option. Oracle Locator provides capabilities that are typically required to support internet and wireless service-based applications and partner-based GIS solutions. Oracle Locator is a limited subset of Oracle Spatial. For more information, see Oracle Locator in the Oracle documentation.

> **Important**
>
> If you use Oracle Locator, Amazon RDS automatically updates your DB instance to the latest Oracle PSU if there are security vulnerabilities with a Common Vulnerability Scoring System (CVSS) score of 9+ or other announced security vulnerabilities.

Supported database releases for Oracle Locator

RDS for Oracle supports Oracle Locator for Oracle Database 19c. Oracle Locator isn't supported for Oracle Database 21c, but its functionality is available in the Oracle Spatial option. Formerly, the Spatial option required additional licenses. Oracle Locator represented a subset of Oracle Spatial features and didn't require additional licenses. In 2019, Oracle announced that all Oracle Spatial features were included in the Enterprise Edition and Standard Edition 2 licenses without additional cost. Consequently, the Oracle Spatial option no longer required additional licensing. For more information, see Machine Learning, Spatial and Graph - No License Required! in the Oracle Database Insider blog.

Prerequisites for Oracle Locator

The following are prerequisites for using Oracle Locator:

- Your DB instance must be of sufficient class. Oracle Locator is not supported for the db.t3.small DB instance classes. For more information, see RDS for Oracle DB instance classes.
- Your DB instance must have **Auto Minor Version Upgrade** enabled. This option enables your DB instance to receive minor DB engine version upgrades automatically when they become available and is required for any options that install the Oracle Java Virtual Machine (JVM). Amazon RDS uses this option to update your DB instance to the latest Oracle Patch Set Update (PSU) or Release Update (RU). For more information, see Modifying an Amazon RDS DB instance.

Best practices for Oracle Locator

The following are best practices for using Oracle Locator:

- For maximum security, use the LOCATOR option with Secure Sockets Layer (SSL). For more information, see Oracle Secure Sockets Layer.
- Configure your DB instance to restrict access to your DB instance. For more information, see Scenarios for accessing a DB instance in a VPC and Working with a DB instance in a VPC.

Adding the Oracle Locator option

The following is the general process for adding the LOCATOR option to a DB instance:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with the DB instance.

If Oracle Java Virtual Machine (JVM) is *not* installed on the DB instance, there is a brief outage while the LOCATOR option is added. There is no outage if Oracle Java Virtual Machine (JVM) is already installed on the DB instance. After you add the option, you don't need to restart your DB instance. As soon as the option group is active, Oracle Locator is available.

> **Note**
>
> During this outage, password verification functions are disabled briefly. You can also expect to see events related to password verification functions during the outage. Password verification functions are enabled again before the Oracle DB instance is available.

To add the LOCATOR option to a DB instance

1. Determine the option group that you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:
   a. For **Engine**, choose the oracle edition for your DB instance.
   b. For **Major engine version**, choose the version of your DB instance.

For more information, see Creating an option group.

2. Add the **LOCATOR** option to the option group. For more information about adding options, see Adding an option to an option group.
3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

Using Oracle Locator

After you enable the Oracle Locator option, you can begin using it. You should only use Oracle Locator features. Don't use any Oracle Spatial features unless you have a license for Oracle Spatial.

For a list of features that are supported for Oracle Locator, see Features Included with Locator in the Oracle documentation.

For a list of features that are not supported for Oracle Locator, see Features Not Included with Locator in the Oracle documentation.

Removing the Oracle Locator option

After you drop all objects that use data types provided by the `LOCATOR` option, you can remove the option from a DB instance. If Oracle Java Virtual Machine (JVM) is *not* installed on the DB instance, there is a brief outage while the `LOCATOR` option is removed. There is no outage if Oracle Java Virtual Machine (JVM) is already installed on the DB instance. After you remove the `LOCATOR` option, you don't need to restart your DB instance.

To drop the `LOCATOR` option

1. Back up your data.

> **Warning**
>
> If the instance uses data types that were enabled as part of the option, and if you remove the LOCATOR option, you can lose data. For more information, see Backing up, restoring, and exporting data.

2. Check whether any existing objects reference data types or features of the LOCATOR option.

   If LOCATOR options exist, the instance can get stuck when applying the new option group that doesn't have the LOCATOR option. You can identify the objects by using the following queries:

```
SELECT OWNER, SEGMENT_NAME, TABLESPACE_NAME, BYTES/1024/1024 mbytes
FROM   DBA_SEGMENTS
WHERE  SEGMENT_TYPE LIKE '%TABLE%'
AND    (OWNER, SEGMENT_NAME) IN
       (SELECT DISTINCT OWNER, TABLE_NAME
        FROM   DBA_TAB_COLUMNS
        WHERE  DATA_TYPE='SDO_GEOMETRY'
        AND    OWNER <> 'MDSYS')
ORDER BY 1,2,3,4;

SELECT OWNER, TABLE_NAME, COLUMN_NAME
FROM   DBA_TAB_COLUMNS
WHERE  DATA_TYPE = 'SDO_GEOMETRY'
AND    OWNER <> 'MDSYS'
ORDER BY 1,2,3;
```

3. Drop any objects that reference data types or features of the LOCATOR option.
4. Do one of the following:

   - Remove the LOCATOR option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
   - Modify the DB instance and specify a different option group that doesn't include the LOCATOR option. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Oracle native network encryption

Amazon RDS supports Oracle native network encryption (NNE). With the `NATIVE_NETWORK_ENCRYPTION` option, you can encrypt data as it moves to and from a DB instance. Amazon RDS supports NNE for all editions of Oracle Database.

A detailed discussion of Oracle native network encryption is beyond the scope of this guide, but you should understand the strengths and weaknesses of each algorithm and key before you decide on a solution for your deployment. For information about the algorithms and keys that are available through Oracle native network encryption, see Configuring network data encryption in the Oracle documentation. For more information about AWS security, see the AWS security center.

> **Note**
>
> You can use Native Network Encryption or Secure Sockets Layer, but not both. For more information, see Oracle Secure Sockets Layer.

Topics

- NATIVE_NETWORK_ENCRYPTION option settings
- Adding the NATIVE_NETWORK_ENCRYPTION option
- Setting NNE values in the sqlnet.ora
- Modifying NATIVE_NETWORK_ENCRYPTION option settings
- Removing the NATIVE_NETWORK_ENCRYPTION option

NATIVE_NETWORK_ENCRYPTION option settings

You can specify encryption requirements on both the server and the client. The DB instance can act as a client when, for example, it uses a database link to connect to another database. You might want to avoid forcing encryption on the server side. For example, you might not want to force all client communications to use encryption because the server requires it. In this case, you can force encryption on the client side using the `SQLNET.*CLIENT` options.

Amazon RDS supports the following settings for the `NATIVE_NETWORK_ENCRYPTION` option.

Note

When you use commas to separate values for an option setting, don't put a space after the comma.

Option setting	Valid values	Default values	Description
SQLNET.ALLOW_WEAK_CRYPTO_CLIENTS	TRUE, FALSE	TRUE	The behavior of the server when a client using a non-secure cipher attempts to connect to the database. If TRUE, clients can connect even if they aren't patched with the July 2021 PSU. If the setting is FALSE, clients can connect to the database only when they are patched with the July 2021 PSU. Before you set SQLNET.ALLOW_WEAK_CRYPTO_CLIENTS to FALSE, make sure that the following conditions are met: • SQLNET.ENCRYPTION_TYPES_SERVER and SQLNET.ENCRYPTION_TYPES_CLIENT have one matching encryption method that is not DES, 3DES, or RC4 (all key lengths). • SQLNET.CHECKSUM_TYPES_SERVER and SQLNET.CHECKSUM_TYPES_CLIENT have one matching secure checksumming method that is not MD5. • The client is patched with the July 2021 PSU. If the client isn't patched,

Option setting	Valid values	Default values	Description
			the client loses the connection and receives the ORA-12269 error.

Option setting	Valid values	Default values	Description
SQLNET.ALLOW_WEAK_CRYPTO	TRUE, FALSE	TRUE	The behavior of the server when a client using a non-secure cipher attempts to connect to the database. The following ciphers are considered not secure: • DES encryption method (all key lengths) • 3DES encryption method (all key lengths) • RC4 encryption method (all key lengths) • MD5 checksumming method If the setting is TRUE, clients can connect when they use the preceding non-secure ciphers. If the setting is FALSE, the database prevents clients from connecting when they use the preceding non-secure ciphers. Before you set SQLNET.ALLOW_WEAK_CRYPTO to FALSE, make sure that the following conditions are met: • SQLNET.ENCRYPTION_TYPES_SERVER and SQLNET.ENCRYPTION_TYPES_CLIENT have one matching encryption method that is not DES, 3DES, or RC4 (all key lengths).

Option setting	Valid values	Default values	Description
			• SQLNET.CHECKSUM_TYPES_SERVER and SQLNET.CHECKSUM_TYPES_CLIENT have one matching secure checksumming method that is not MD5. • The client is patched with the July 2021 PSU. If the client isn't patched, the client loses the connection and receives the ORA-12269 error.
SQLNET.CRYPTO_CHECKSUM_CLIENT	Accepted Rejected Requested Required	Requested	The data integrity behavior when a DB instance connects to the client, or a server acting as a client. When a DB instance uses a database link, it acts as a client. Requested indicates that the client doesn't require the DB instance to perform a checksum.
SQLNET.CRYPTO_CHECKSUM_SERVER	Accepted Rejected Requested Required	Requested	The data integrity behavior when a client, or a server acting as a client, connects to the DB instance. When a DB instance uses a database link, it acts as a client. Requested indicates that the DB instance doesn't require the client to perform a checksum.

Option setting	Valid values	Default values	Description
SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT	SHA256, SHA384, SHA512, SHA1, MD5	SHA256, SHA384, SHA512	A list of checksum algorithms. You can specify either one value or a comma-separated list of values. If you use a comma, don't insert a space after the comma; otherwise, you receive an InvalidParameterValue error. This parameter and SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER must have a common cipher.
SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER	SHA256, SHA384, SHA512, SHA1, MD5	SHA256, SHA384, SHA512, SHA1, MD5	A list of checksum algorithms. You can specify either one value or a comma-separated list of values. If you use a comma, don't insert a space after the comma; otherwise, you receive an InvalidParameterValue error. This parameter and SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT must have a common cipher.
SQLNET.ENCRYPTION_CLIENT	Accepted, Rejected, Requested, Required	Requested	The encryption behavior of the client when a client, or a server acting as a client, connects to the DB instance. When a DB instance uses a database link, it acts as a client. Requested indicates that the client does not require traffic from the server to be encrypted.

Option setting	Valid values	Default values	Description
SQLNET.ENCRYPTION_SERVER	Accepted Rejected Requested , Required	Requested	The encryption behavior of the server when a client, or a server acting as a client, connects to the DB instance. When a DB instance uses a database link, it acts as a client. Requested indicates that the DB instance does not require traffic from the client to be encrypted.

Option setting	Valid values	Default values	Description
SQLNET.ENCRYPTION_TYPES_CLIENT	RC4_256, AES256, AES192, 3DES168, RC4_128, AES128, 3DES112, RC4_56, DES, RC4_40, DES40	RC4_256, AES256, AES192, 3DES168, RC4_128, AES128, 3DES112, RC4_56, DES, RC4_40, DES40	A list of encryption algorithms used by the client. The client attempts to decrypt the server input by trying each algorithm in order, proceeding until an algorithm succeeds or the end of the list is reached. Amazon RDS uses the following default list from Oracle. RDS starts with RC4_256 and proceeds down the list in order. You can change the order or limit the algorithms that the DB instance will accept. 1. RC4_256: RSA RC4 (256-bit key size) 2. AES256: AES (256-bit key size) 3. AES192: AES (192-bit key size) 4. 3DES168: 3-key Triple-DES (112-bit effective key size) 5. RC4_128: RSA RC4 (128-bit key size) 6. AES128: AES (128-bit key size) 7. 3DES112: 2-key Triple-DES (80-bit effective key size) 8. RC4_56: RSA RC4 (56-bit key size) 9. DES: Standard DES (56-bit key size) 10. RC4_40: RSA RC4 (40-bit key size) 11. DES40: DES40 (40-bit key size) You can specify either one value or a comma-separated list of values. If you a comma, don't insert a space after

Option setting	Valid values	Default values	Description
			the comma; otherwise, you receive an `InvalidParameterValue` error. This parameter and `SQLNET.SQLNET.ENCRYPTION_TYPES_SERVER` must have a common cipher.

Option setting	Valid values	Default values	Description
SQLNET.ENCRYPTION_TYPES_SERVER	RC4_256, AES256, AES192, 3DES168, RC4_128, AES128, 3DES112, RC4_56, DES, RC4_40, DES40	RC4_256, AES256, AES192, 3DES168, RC4_128, AES128, 3DES112, RC4_56, DES, RC4_40, DES40	A list of encryption algorithms used by the DB instance. The DB instance uses each algorithm, in order, to attempt to decrypt the client input until an algorithm succeeds or until the end of the list is reached. Amazon RDS uses the following default list from Oracle. You can change the order or limit the algorithms that the client will accept. 1. RC4_256: RSA RC4 (256-bit key size) 2. AES256: AES (256-bit key size) 3. AES192: AES (192-bit key size) 4. 3DES168: 3-key Triple-DES (112-bit effective key size) 5. RC4_128: RSA RC4 (128-bit key size) 6. AES128: AES (128-bit key size) 7. 3DES112: 2-key Triple-DES (80-bit effective key size) 8. RC4_56: RSA RC4 (56-bit key size) 9. DES: Standard DES (56-bit key size) 10RC4_40: RSA RC4 (40-bit key size) 11DES40: DES40 (40-bit key size) You can specify either one value or a comma-separated list of values. If you a comma, don't insert a space after the comma; otherwise, you receive an InvalidParameterValue error.

Option setting	Valid values	Default values	Description
			This parameter and SQLNET.SQLNET.ENCRYPTION_TYPES_SERVER must have a common cipher.

Adding the NATIVE_NETWORK_ENCRYPTION option

The general process for adding the NATIVE_NETWORK_ENCRYPTION option to a DB instance is the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with the DB instance.

When the option group is active, NNE is active.

To add the NATIVE_NETWORK_ENCRYPTION option to a DB instance using the AWS Management Console

1. For **Engine**, choose the Oracle edition that you want to use. NNE is supported on all editions.
2. For **Major engine version**, choose the version of your DB instance.

   For more information, see Creating an option group.
3. Add the **NATIVE_NETWORK_ENCRYPTION** option to the option group. For more information about adding options, see Adding an option to an option group.

   > **Note**
   >
   > After you add the **NATIVE_NETWORK_ENCRYPTION** option, you don't need to restart your DB instances. As soon as the option group is active, NNE is active.

4. Apply the option group to a new or existing DB instance:

- For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
- For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. After you add the **NATIVE_NETWORK_ENCRYPTION** option, you don't need to restart your DB instance. As soon as the option group is active, NNE is active. For more information, see Modifying an Amazon RDS DB instance.

Setting NNE values in the sqlnet.ora

With Oracle native network encryption, you can set network encryption on the server side and client side. The client is the computer used to connect to the DB instance. You can specify the following client settings in the sqlnet.ora:

- `SQLNET.ALLOW_WEAK_CRYPTO`
- `SQLNET.ALLOW_WEAK_CRYPTO_CLIENTS`
- `SQLNET.CRYPTO_CHECKSUM_CLIENT`
- `SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT`
- `SQLNET.ENCRYPTION_CLIENT`
- `SQLNET.ENCRYPTION_TYPES_CLIENT`

For information, see Configuring network data encryption and integrity for Oracle servers and clients in the Oracle documentation.

Sometimes, the DB instance rejects a connection request from an application. For example, a rejection can occur when the encryption algorithms on the client and on the server don't match. To test Oracle native network encryption, add the following lines to the sqlnet.ora file on the client:

```
DIAG_ADR_ENABLED=off
TRACE_DIRECTORY_CLIENT=/tmp
TRACE_FILE_CLIENT=nettrace
TRACE_LEVEL_CLIENT=16
```

When a connection is attempted, the preceding lines generate a trace file on the client called `/tmp/nettrace*`. The trace file contains information about the connection. For more information about connection-related issues when you are using Oracle Native Network Encryption, see About negotiating encryption and integrity in the Oracle Database documentation.

Modifying NATIVE_NETWORK_ENCRYPTION option settings

After you enable the NATIVE_NETWORK_ENCRYPTION option, you can modify its settings. Currently, you can modify NATIVE_NETWORK_ENCRYPTION option settings only with the AWS CLI or RDS API. You can't use the console. The following example modifies two settings in the option.

```
aws rds add-option-to-option-group \
    --option-group-name my-option-group \
    --options
 "OptionName=NATIVE_NETWORK_ENCRYPTION,OptionSettings=[{Name=SQLNET.CRYPTO_CHECKSUM_TYPES_SERVE
{Name=SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER,Value=SHA256}]" \
    --apply-immediately
```

To learn how to modify option settings using the CLI, see AWS CLI. For more information about each setting, see NATIVE_NETWORK_ENCRYPTION option settings.

Topics

- Modifying CRYPTO_CHECKSUM_* values
- Modifying ALLOW_WEAK_CRYPTO* settings

Modifying CRYPTO_CHECKSUM_* values

If you modify **NATIVE_NETWORK_ENCRYPTION** option settings, make sure that the following option settings have at least one common cipher:

- SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER
- SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT

The following example shows a scenario in which you modify SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER. The configuration is valid because the CRYPTO_CHECKSUM_TYPES_CLIENT and CRYPTO_CHECKSUM_TYPES_SERVER both use SHA256.

Option setting	Values before modification	Values after modification
SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT	**SHA256**, SHA384, SHA512	No change

Option setting	Values before modification	Values after modification
SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER	**SHA256**, SHA384, SHA512, SHA1, MD5	SHA1,MD5, **SHA256**

For another example, assume that you want to modify SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER from its default setting to SHA1,MD5. In this case, make sure you set SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT to SHA1 or MD5. These algorithms aren't included in the default values for SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT.

Modifying ALLOW_WEAK_CRYPTO* settings

To set the SQLNET.ALLOW_WEAK_CRYPTO* options from the default value to FALSE, make sure that the following conditions are met:

- SQLNET.ENCRYPTION_TYPES_SERVER and SQLNET.ENCRYPTION_TYPES_CLIENT have one matching secure encryption method. A method is considered secure if it's not DES, 3DES, or RC4 (all key lengths).
- SQLNET.CHECKSUM_TYPES_SERVER and SQLNET.CHECKSUM_TYPES_CLIENT have one matching secure checksumming method. A method is considered secure if it's not MD5.
- The client is patched with the July 2021 PSU. If the client isn't patched, the client loses the connection and receives the ORA-12269 error.

The following example shows sample NNE settings. Assume that you want to set SQLNET.ENCRYPTION_TYPES_SERVER and SQLNET.ENCRYPTION_TYPES_CLIENT to FALSE, thereby blocking non-secure connections. The checksum option settings meet the prerequisites because they both have SHA256. However, SQLNET.ENCRYPTION_TYPES_CLIENT and SQLNET.ENCRYPTION_TYPES_SERVER use the DES, 3DES, and RC4 encryption methods, which are non-secure. Therefore, to set the SQLNET.ALLOW_WEAK_CRYPTO* options to FALSE, first set SQLNET.ENCRYPTION_TYPES_SERVER and SQLNET.ENCRYPTION_TYPES_CLIENT to a secure encryption method such as AES256.

Option setting	Values
SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT	SHA256, SHA384, SHA512
SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER	SHA1,MD5,SHA256
SQLNET.ENCRYPTION_TYPES_CLIENT	RC4_256, 3DES168, DES40
SQLNET.ENCRYPTION_TYPES_SERVER	RC4_256, 3DES168, DES40

Removing the NATIVE_NETWORK_ENCRYPTION option

You can remove NNE from a DB instance.

To remove the NATIVE_NETWORK_ENCRYPTION option from a DB instance, do one of the following:

- To remove the option from multiple DB instances, remove the NATIVE_NETWORK_ENCRYPTION option from the option group they belong to. This change affects all DB instances that use the option group. After you remove the NATIVE_NETWORK_ENCRYPTION option, you don't need to restart your DB instances. For more information, see Removing an option from an option group.
- To remove the option from a single DB instance, modify the DB instance and specify a different option group that doesn't include the NATIVE_NETWORK_ENCRYPTION option. You can specify the default (empty) option group, or a different custom option group. After you remove the NATIVE_NETWORK_ENCRYPTION option, you don't need to restart your DB instance. For more information, see Modifying an Amazon RDS DB instance.

Oracle OLAP

Amazon RDS supports Oracle OLAP through the use of the OLAP option. This option provides On-line Analytical Processing (OLAP) for Oracle DB instances. You can use Oracle OLAP to analyze large amounts of data by creating dimensional objects and cubes in accordance with the OLAP standard. For more information, see the Oracle documentation.

> **Important**
>
> If you use Oracle OLAP, Amazon RDS automatically updates your DB instance to the latest Oracle PSU if there are security vulnerabilities with a Common Vulnerability Scoring System (CVSS) score of 9+ or other announced security vulnerabilities.

Amazon RDS supports Oracle OLAP for the Enterprise Edition of Oracle Database 19c and higher.

Prerequisites for Oracle OLAP

The following are prerequisites for using Oracle OLAP:

- You must have an Oracle OLAP license from Oracle. For more information, see Licensing Information in the Oracle documentation.
- Your DB instance must be of a sufficient instance class. Oracle OLAP isn't supported for the db.t3.small DB instance classes. For more information, see RDS for Oracle DB instance classes.
- Your DB instance must have **Auto Minor Version Upgrade** enabled. This option enables your DB instance to receive minor DB engine version upgrades automatically when they become available and is required for any options that install the Oracle Java Virtual Machine (JVM). Amazon RDS uses this option to update your DB instance to the latest Oracle Patch Set Update (PSU) or Release Update (RU). For more information, see Modifying an Amazon RDS DB instance.
- Your DB instance must not have a user named OLAPSYS. If it does, the OLAP option installation fails.

Best practices for Oracle OLAP

The following are best practices for using Oracle OLAP:

- For maximum security, use the OLAP option with Secure Sockets Layer (SSL). For more information, see Oracle Secure Sockets Layer.

- Configure your DB instance to restrict access to your DB instance. For more information, see Scenarios for accessing a DB instance in a VPC and Working with a DB instance in a VPC.

Adding the Oracle OLAP option

The following is the general process for adding the OLAP option to a DB instance:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with the DB instance.

If Oracle Java Virtual Machine (JVM) is *not* installed on the DB instance, there is a brief outage while the OLAP option is added. There is no outage if Oracle Java Virtual Machine (JVM) is already installed on the DB instance. After you add the option, you don't need to restart your DB instance. As soon as the option group is active, Oracle OLAP is available.

To add the OLAP option to a DB instance

1. Determine the option group that you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   - For **Engine**, choose the Oracle edition for your DB instance.
   - For **Major engine version**, choose the version of your DB instance.

   For more information, see Creating an option group.
2. Add the **OLAP** option to the option group. For more information about adding options, see Adding an option to an option group.
3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

Using Oracle OLAP

After you enable the Oracle OLAP option, you can begin using it. For a list of features that are supported for Oracle OLAP, see the Oracle documentation.

Removing the Oracle OLAP option

After you drop all objects that use data types provided by the OLAP option, you can remove the option from a DB instance. If Oracle Java Virtual Machine (JVM) is *not* installed on the DB instance, there is a brief outage while the OLAP option is removed. There is no outage if Oracle Java Virtual Machine (JVM) is already installed on the DB instance. After you remove the OLAP option, you don't need to restart your DB instance.

To drop the OLAP option

1. Back up your data.

   > **Warning**
   >
   > If the instance uses data types that were enabled as part of the option, and if you remove the OLAP option, you can lose data. For more information, see Backing up, restoring, and exporting data.

2. Check whether any existing objects reference data types or features of the OLAP option.
3. Drop any objects that reference data types or features of the OLAP option.
4. Do one of the following:

   - Remove the OLAP option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
   - Modify the DB instance and specify a different option group that doesn't include the OLAP option. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Oracle Secure Sockets Layer

To enable SSL encryption for an RDS for Oracle DB instance, add the Oracle SSL option to the option group associated with the DB instance. Amazon RDS uses a second port, as required by Oracle, for SSL connections. This approach allows both clear text and SSL-encrypted communication to occur at the same time between a DB instance and SQL*Plus. For example, you can use the port with clear text communication to communicate with other resources inside a VPC while using the port with SSL-encrypted communication to communicate with resources outside the VPC.

> **Note**
>
> You can use either SSL or Native Network Encryption (NNE) on the same RDS for Oracle DB instance, but not both. If you use SSL encryption, make sure to turn off any other connection encryption. For more information, see Oracle native network encryption.

SSL/TLS and NNE are no longer part of Oracle Advanced Security. In RDS for Oracle, you can use SSL encryption with all licensed editions of the following database versions:

- Oracle Database 21c (21.0.0)
- Oracle Database 19c (19.0.0)

Topics

- TLS versions for the Oracle SSL option
- Cipher suites for the Oracle SSL option
- FIPS support
- Certificate compatibility with cipher suites
- Adding the SSL option
- Configuring SQL*Plus to use SSL with an RDS for Oracle DB instance
- Connecting to an RDS for Oracle DB instance using SSL
- Setting up an SSL connection over JDBC
- Enforcing a DN match with an SSL connection
- Troubleshooting SSL connections

TLS versions for the Oracle SSL option

Amazon RDS for Oracle supports Transport Layer Security (TLS) versions 1.0 and 1.2. When you add a new Oracle SSL option, set `SQLNET.SSL_VERSION` explicitly to a valid value. The following values are allowed for this option setting:

- `"1.0"` – Clients can connect to the DB instance using TLS version 1.0 only. For existing Oracle SSL options, `SQLNET.SSL_VERSION` is set to `"1.0"` automatically. You can change the setting if necessary.
- `"1.2"` – Clients can connect to the DB instance using TLS 1.2 only.
- `"1.2 or 1.0"` – Clients can connect to the DB instance using either TLS 1.2 or 1.0.

Cipher suites for the Oracle SSL option

Amazon RDS for Oracle supports multiple SSL cipher suites. By default, the Oracle SSL option is configured to use the `SSL_RSA_WITH_AES_256_CBC_SHA` cipher suite. To specify a different cipher suite to use over SSL connections, use the `SQLNET.CIPHER_SUITE` option setting.

You can specify multiple values for `SQLNET.CIPHER_SUITE`. This technique is useful if you have database links between your DB instances and decide to update your cipher suites.

The following table summarizes SSL support for RDS for Oracle in all editions of Oracle Database 19c and 21c.

Cipher suite (SQLNET.CIPHER_SUITE)	TLS version support (SQLNET.SSL_VERSION)	FIPS support	FedRAMP compliant
SSL_RSA_WITH_AES_256_CBC_SHA (default)	1.0 and 1.2	Yes	No
SSL_RSA_WITH_AES_256_CBC_SHA256	1.2	Yes	No
SSL_RSA_WITH_AES_256_GCM_SHA384	1.2	Yes	No
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	1.2	Yes	Yes

Cipher suite (SQLNET.CIPHER_SUITE)	TLS version support (SQLNET.SSL_VERSION)	FIPS support	FedRAMP compliant
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	1.2	Yes	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	1.2	Yes	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	1.2	Yes	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	1.2	Yes	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	1.2	Yes	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	1.2	Yes	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	1.2	Yes	Yes

FIPS support

RDS for Oracle allows you to use the Federal Information Processing Standard (FIPS) standard for 140-2. FIPS 140-2 is a United States government standard that defines cryptographic module security requirements. You turn on the FIPS standard by setting `FIPS.SSLFIPS_140` to `TRUE` for the Oracle SSL option. When FIPS 140-2 is configured for SSL, the cryptographic libraries encrypt data between the client and the RDS for Oracle DB instance.

Clients must use the cipher suite that is FIPS-compliant. When establishing a connection, the client and RDS for Oracle DB instance negotiate which cipher suite to use when transmitting messages back and forth. The table in Cipher suites for the Oracle SSL option shows the FIPS-compliant SSL cipher suites for each TLS version. For more information, see Oracle database FIPS 140-2 settings in the Oracle Database documentation.

Certificate compatibility with cipher suites

RDS for Oracle supports both RSA and Elliptic Curve Digital Signature Algorithm (ECDSA) certificates. When you configure SSL for your DB instance, you must ensure that the cipher suites you specify in the `SQLNET.CIPHER_SUITE` option setting are compatible with the certificate type used by your DB instance.

The following table shows the compatibility between certificate types and cipher suites:

Certificate type	Compatible cipher suites	Incompatible cipher suites
RSA certificates (rds-ca-2019, rds-ca-rsa2048-g1, rds-ca-rsa4096-g1)	SSL_RSA_WITH_AES_256_CBC_SHA SSL_RSA_WITH_AES_256_CBC_SHA256 SSL_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
ECDSA certificates (rds-ca-ecc384-g1)	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_CBC_SHA

Certificate type	Compatible cipher suites	Incompatible cipher suites
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	SSL_RSA_WITH_AES_256_CBC_SHA256 SSL_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

When you specify multiple cipher suites in the `SQLNET.CIPHER_SUITE` option setting, make sure to include at least one cipher suite that is compatible with the certificate type used by your DB instance. If you're using an option group with multiple DB instances that have different certificate types, include at least one cipher suite for each certificate type.

If you attempt to associate an option group with an SSL option that contains only cipher suites incompatible with the certificate type of a DB instance, the operation will fail with an error message indicating the incompatibility.

Adding the SSL option

To use SSL, your RDS for Oracle DB instance must be associated with an option group that includes the `SSL` option.

Console

To add the SSL option to an option group

1. Create a new option group or identify an existing option group to which you can add the SSL option.

   For information about creating an option group, see Creating an option group.

2. Add the SSL option to the option group.

   If you want to use only FIPS-verified cipher suites for SSL connections, set the option `FIPS.SSLFIPS_140` to `TRUE`. For information about the FIPS standard, see FIPS support.

   For information about adding an option to an option group, see Adding an option to an option group.

3. Create a new RDS for Oracle DB instance and associate the option group with it, or modify an RDS for Oracle DB instance to associate the option group with it.

   For information about creating an DB instance, see Creating an Amazon RDS DB instance.

   For information about modifying an DB instance, see Modifying an Amazon RDS DB instance.

AWS CLI

To add the SSL option to an option group

1. Create a new option group or identify an existing option group to which you can add the SSL option.

   For information about creating an option group, see Creating an option group.

2. Add the SSL option to the option group.

   Specify the following option settings:

   - `Port` – The SSL port number
   - `VpcSecurityGroupMemberships` – The VPC security group for which the option is enabled
   - `SQLNET.SSL_VERSION` – The TLS version that client can use to connect to the DB instance

For example, the following AWS CLI command adds the SSL option to an option group named `ora-option-group`.

Example

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group --option-group-name ora-option-group \
  --options
 'OptionName=SSL,Port=2484,VpcSecurityGroupMemberships="sg-68184619",OptionSettings=[{Name=
```

For Windows:

```
aws rds add-option-to-option-group --option-group-name ora-option-group ^
  --options
 'OptionName=SSL,Port=2484,VpcSecurityGroupMemberships="sg-68184619",OptionSettings=[{Name=
```

3. Create a new RDS for Oracle DB instance and associate the option group with it, or modify an RDS for Oracle DB instance to associate the option group with it.

   For information about creating an DB instance, see Creating an Amazon RDS DB instance.

   For information about modifying an DB instance, see Modifying an Amazon RDS DB instance.

Configuring SQL*Plus to use SSL with an RDS for Oracle DB instance

Before you can connect to an RDS for Oracle DB instance that uses the Oracle SSL option, you must configure SQL*Plus before connecting.

> **Note**
>
> To allow access to the DB instance from the appropriate clients, ensure that your security groups are configured correctly. For more information, see Controlling access with security groups. Also, these instructions are for SQL*Plus and other clients that directly use an Oracle home. For JDBC connections, see Setting up an SSL connection over JDBC.

To configure SQL*Plus to use SSL to connect to an RDS for Oracle DB instance

1. Set the ORACLE_HOME environment variable to the location of your Oracle home directory.

   The path to your Oracle home directory depends on your installation. The following example sets the ORACLE_HOME environment variable.

   ```
   prompt>export ORACLE_HOME=/home/user/app/user/product/19.0.0/dbhome_1
   ```

   For information about setting Oracle environment variables, see SQL*Plus environment variables in the Oracle documentation, and also see the Oracle installation guide for your operating system.

2. Append $ORACLE_HOME/lib to the LD_LIBRARY_PATH environment variable.

   The following is an example that sets the LD_LIBRARY_PATH environment variable.

   ```
   prompt>export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
   ```

3. Create a directory for the Oracle wallet at $ORACLE_HOME/ssl_wallet.

   The following is an example that creates the Oracle wallet directory.

   ```
   prompt>mkdir $ORACLE_HOME/ssl_wallet
   ```

4. Download the certificate bundle .pem file that works for all AWS Regions and put the file in the ssl_wallet directory. For information, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

5. In the $ORACLE_HOME/network/admin directory, modify or create the tnsnames.ora file and include the following entry.

   ```
   net_service_name =
     (DESCRIPTION =
       (ADDRESS_LIST =
         (ADDRESS =
           (PROTOCOL = TCPS)
           (HOST = endpoint)
           (PORT = ssl_port_number)
         )
       )
       (CONNECT_DATA =
   ```

```
      (SID = database_name)
    )
    (SECURITY =
      (SSL_SERVER_CERT_DN =
"C=US,ST=Washington,L=Seattle,O=Amazon.com,OU=RDS,CN=endpoint")
    )
  )
```

6. In the same directory, modify or create the sqlnet.ora file and include the following parameters.

> **Note**
>
> To communicate with entities over a TLS secured connection, Oracle requires a wallet with the necessary certificates for authentication. You can use Oracle's ORAPKI utility to create and maintain Oracle wallets, as shown in step 7. For more information, see Setting up Oracle wallet using ORAPKI in the Oracle documentation.

```
WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY =
 $ORACLE_HOME/ssl_wallet)))
SSL_CLIENT_AUTHENTICATION = FALSE
SSL_VERSION = 1.0
SSL_CIPHER_SUITES = (SSL_RSA_WITH_AES_256_CBC_SHA)
SSL_SERVER_DN_MATCH = ON
```

> **Note**
>
> You can set SSL_VERSION to a higher value if your DB instance supports it.

7. Run the following command to create the Oracle wallet.

```
prompt>orapki wallet create -wallet $ORACLE_HOME/ssl_wallet -auto_login_only
```

8. Extract each certificate in the .pem bundle file into a separate .pem file using an OS utility.
9. Add each certificate to your wallet using separate `orapki` commands, replacing *`certificate-pem-file`* with the absolute file name of the .pem file.

```
prompt>orapki wallet add -wallet $ORACLE_HOME/ssl_wallet -trusted_cert -cert
```

```
        certificate-pem-file -auto_login_only
```

For more information, see Rotating your SSL/TLS certificate.

Connecting to an RDS for Oracle DB instance using SSL

After you configure SQL*Plus to use SSL as described previously, you can connect to the RDS for Oracle DB instance with the SSL option. Optionally, you can first export the TNS_ADMIN value that points to the directory that contains the tnsnames.ora and sqlnet.ora files. Doing so ensures that SQL*Plus can find these files consistently. The following example exports the TNS_ADMIN value.

```
export TNS_ADMIN=${ORACLE_HOME}/network/admin
```

Connect to the DB instance. For example, you can connect using SQL*Plus and a *<net_service_name>* in a tnsnames.ora file.

```
sqlplus mydbuser@net_service_name
```

You can also connect to the DB instance using SQL*Plus without using a tnsnames.ora file by using the following command.

```
sqlplus 'mydbuser@(DESCRIPTION = (ADDRESS = (PROTOCOL = TCPS)(HOST = endpoint) (PORT
 = ssl_port_number))(CONNECT_DATA = (SID = database_name)))'
```

You can also connect to the RDS for Oracle DB instance without using SSL. For example, the following command connects to the DB instance through the clear text port without SSL encryption.

```
sqlplus 'mydbuser@(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = endpoint) (PORT
 = port_number))(CONNECT_DATA = (SID = database_name)))'
```

If you want to close Transmission Control Protocol (TCP) port access, create a security group with no IP address ingresses and add it to the instance. This addition closes connections over the TCP port, while still allowing connections over the SSL port that are specified from IP addresses within the range permitted by the SSL option security group.

Setting up an SSL connection over JDBC

To use an SSL connection over JDBC, you must create a keystore, trust the Amazon RDS root CA certificate, and use the code snippet specified following.

To create the keystore in JKS format, you can use the following command. For more information about creating the keystore, see the Creating a keystore in the Oracle documentation. For reference information, see keytool in the *Java Platform, Standard Edition Tools Reference*.

```
keytool -genkey -alias client -validity 365 -keyalg RSA -keystore clientkeystore
```

Take the following steps to trust the Amazon RDS root CA certificate.

To trust the Amazon RDS root CA certificate

1. Download the certificate bundle .pem file that works for all AWS Regions and put the file in the ssl_wallet directory.

   For information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
2. Extract each certificate in the .pem file into a separate file using an OS utility.
3. Convert each certificate to .der format using a separate `openssl` command, replacing *`certificate-pem-file`* with the name of the certificate .pem file (without the .pem extension).

   ```
   openssl x509 -outform der -in certificate-pem-file.pem -out certificate-pem-file.der
   ```
4. Import each certificate into the keystore using the following command.

   ```
   keytool -import -alias rds-root -keystore clientkeystore.jks -file certificate-pem-file.der
   ```

   For more information, see Rotating your SSL/TLS certificate.
5. Confirm that the key store was created successfully.

   ```
   keytool -list -v -keystore clientkeystore.jks
   ```

Enter the keystore password when you are prompted for it.

The following code example shows how to set up the SSL connection using JDBC.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class OracleSslConnectionTest {
    private static final String DB_SERVER_NAME = "dns-name-provided-by-amazon-rds";
    private static final Integer SSL_PORT = "ssl-option-port-configured-in-option-
group";
    private static final String DB_SID = "oracle-sid";
    private static final String DB_USER = "user-name";
    private static final String DB_PASSWORD = "password";
    // This key store has only the prod root ca.
    private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
    private static final String KEY_STORE_PASS = "keystore-password";

    public static void main(String[] args) throws SQLException {
        final Properties properties = new Properties();
        final String connectionString = String.format(
                "jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS)(HOST=%s)(PORT=
%d))(CONNECT_DATA=(SID=%s)))",
                DB_SERVER_NAME, SSL_PORT, DB_SID);
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);
        properties.put("oracle.jdbc.J2EE13Compliant", "true");
        properties.put("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        properties.put("javax.net.ssl.trustStoreType", "JKS");
        properties.put("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);
        final Connection connection = DriverManager.getConnection(connectionString,
 properties);
        // If no exception, that means handshake has passed, and an SSL connection can
 be opened
    }
}
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Enforcing a DN match with an SSL connection

You can use the Oracle parameter SSL_SERVER_DN_MATCH to enforce that the distinguished name (DN) for the database server matches its service name. If you enforce the match verifications, then SSL ensures that the certificate is from the server. If you don't enforce the match verification, then SSL performs the check but allows the connection, regardless if there is a match. If you do not enforce the match, you allow the server to potentially fake its identity.

To enforce DN matching, add the DN match property and use the connection string specified below.

Add the property to the client connection to enforce DN matching.

```
properties.put("oracle.net.ssl_server_dn_match", "TRUE");
```

Use the following connection string to enforce DN matching when using SSL.

```
final String connectionString = String.format(
    "jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS)(HOST=%s)(PORT=%d))" +
    "(CONNECT_DATA=(SID=%s))" +
    "(SECURITY = (SSL_SERVER_CERT_DN =
\"C=US,ST=Washington,L=Seattle,O=Amazon.com,OU=RDS,CN=%s\")))",
    DB_SERVER_NAME, SSL_PORT, DB_SID, DB_SERVER_NAME);
```

Troubleshooting SSL connections

You might query your database and receive the ORA-28860 error.

```
ORA-28860: Fatal SSL error
28860. 00000 - "Fatal SSL error"
*Cause: An error occurred during the SSL connection to the peer. It is likely that this
 side sent data which the peer rejected.
*Action: Enable tracing to determine the exact cause of this error.
```

This error occurs when the client attempts to connect using a version of TLS that the server doesn't support. To avoid this error, edit the sqlnet.ora and set `SSL_VERSION` to the correct TLS version. For more information, see Oracle Support Document 2748438.1 in My Oracle Support.

Oracle Spatial

Amazon RDS supports Oracle Spatial through the use of the `SPATIAL` option. Oracle Spatial provides a SQL schema and functions that facilitate the storage, retrieval, update, and query of collections of spatial data in an Oracle database. For more information, see Spatial Concepts in the Oracle documentation. Amazon RDS supports Oracle Spatial in all editions of all supported releases.

How Spatial Patch Bundles (SPBs) work

Every quarter, RDS for Oracle releases new minor engine versions for every supported major engine. A Release Update (RU) engine version incorporates bug fixes from Oracle by including the RU patches for the specified quarter. A Spatial Patch Bundle (SPB) engine version contains RU patches plus patches specific to Oracle Spatial. For example, 19.0.0.0.ru-2025-01.spb-1.r1 is a minor engine version that contains the RU patches in engine version 19.0.0.0.ru-2025-01.rur-2025-01.r1 plus Spatial patches. SPBs are supported only for Oracle Database 19c.

SPBs function in the same way as RUs, although they are named differently. An RU uses the naming format 19.0.0.0.ru-2025-01.rur-2025-01.r1. An SPB name includes the text "spb," as in 19.0.0.0.ru-2025-01.spb-1.r1. Typically, an SPB is released 2–3 weeks after its corresponding quarterly RU. For example, 19.0.0.0.ru-2025-01.spb-1.r1 is released after 19.0.0.0.ru-2025-01.rur-2025-01.r1.

RDS for Oracle has separate paths for automatic minor version upgrades of RUs and SPBs. If your DB instance uses an RU, then RDS automatically upgrades your instance to an RU. If your DB instance uses an SPB, then RDS upgrades your instance to an SPB.

For more information about RUs and SPBs, see Oracle minor version upgrades. For a list of supported RUs and SPBs for Oracle Database 19c, see Amazon RDS for Oracle Database 19c (19.0.0.0) in *Amazon RDS for Oracle Release Notes*.

Prerequisites for Oracle Spatial

The following are prerequisites for using Oracle Spatial:

- Make sure that your DB instance is of a sufficient instance class. Oracle Spatial isn't supported for the db.t3.small DB instance classes. For more information, see RDS for Oracle DB instance classes.

- Make sure that your DB instance has **Auto Minor Version Upgrade** enabled. This option enables your DB instance to receive minor DB engine version upgrades automatically when they become available and is required for any options that install the Oracle Java Virtual Machine (JVM). Amazon RDS uses this option to update your DB instance to the latest Oracle Patch Set Update (PSU) or Release Update (RU). For more information, see Modifying an Amazon RDS DB instance.

Best practices for Oracle Spatial

The following are best practices for using Oracle Spatial:

- For maximum security, use the `SPATIAL` option with Secure Sockets Layer (SSL). For more information, see Oracle Secure Sockets Layer.
- Configure your DB instance to restrict access to your DB instance. For more information, see Scenarios for accessing a DB instance in a VPC and Working with a DB instance in a VPC.

Adding the Oracle Spatial option

The following is the general process for adding the `SPATIAL` option to a DB instance:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with the DB instance.

If Oracle Java Virtual Machine (JVM) is *not* installed on the DB instance, there is a brief outage while the `SPATIAL` option is added. There is no outage if Oracle Java Virtual Machine (JVM) is already installed on the DB instance. After you add the option, you don't need to restart your DB instance. As soon as the option group is active, Oracle Spatial is available.

> **Note**
>
> During this outage, password verification functions are disabled briefly. You can also expect to see events related to password verification functions during the outage. Password verification functions are enabled again before the Oracle DB instance is available.

To add the SPATIAL option to a DB instance

1. Determine the option group that you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   a. For **Engine**, choose the Oracle edition for your DB instance.

   b. For **Major engine version**, choose the version of your DB instance.

   For more information, see Creating an option group.
2. Add the **SPATIAL** option to the option group. For more information about adding options, see Adding an option to an option group.
3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

Removing the Oracle Spatial option

After you drop all objects that use data types provided by the SPATIAL option, you can drop the option from a DB instance. If Oracle Java Virtual Machine (JVM) is *not* installed on the DB instance, there is a brief outage while the SPATIAL option is removed. There is no outage if Oracle Java Virtual Machine (JVM) is already installed on the DB instance. After you remove the SPATIAL option, you don't need to restart your DB instance.

To drop the SPATIAL option

1. Back up your data.

   Warning

   If the instance uses data types that were enabled as part of the option, and if you remove the SPATIAL option, you can lose data. For more information, see Backing up, restoring, and exporting data.

2. Check whether any existing objects reference data types or features of the SPATIAL option.

   If SPATIAL options exist, the instance can get stuck when applying the new option group that doesn't have the SPATIAL option. You can identify the objects by using the following queries:

```
SELECT OWNER, SEGMENT_NAME, TABLESPACE_NAME, BYTES/1024/1024 mbytes
FROM   DBA_SEGMENTS
WHERE  SEGMENT_TYPE LIKE '%TABLE%'
AND    (OWNER, SEGMENT_NAME) IN
       (SELECT DISTINCT OWNER, TABLE_NAME
        FROM   DBA_TAB_COLUMNS
        WHERE  DATA_TYPE='SDO_GEOMETRY'
        AND    OWNER <> 'MDSYS')
ORDER BY 1,2,3,4;

SELECT OWNER, TABLE_NAME, COLUMN_NAME
FROM   DBA_TAB_COLUMNS
WHERE  DATA_TYPE = 'SDO_GEOMETRY'
AND    OWNER <> 'MDSYS'
ORDER BY 1,2,3;
```

3. Drop any objects that reference data types or features of the SPATIAL option.
4. Do one of the following:

   - Remove the SPATIAL option from the option group it belongs to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
   - Modify the DB instance and specify a different option group that doesn't include the SPATIAL option. This change affects a single DB instance. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Oracle SQLT

Amazon RDS supports Oracle SQLTXPLAIN (SQLT) through the use of the SQLT option. You can use SQLT with any edition of Oracle Database 19c and higher.

The Oracle `EXPLAIN PLAN` statement can determine the execution plan of a SQL statement. It can verify whether the Oracle optimizer chooses a certain execution plan, such as a nested loops join. It also helps you understand the optimizer's decisions, such as why it chose a nested loops join over a hash join. So `EXPLAIN PLAN` helps you understand the statement's performance.

SQLT is an Oracle utility that produces a report. The report includes object statistics, object metadata, optimizer-related initialization parameters, and other information that a database administrator can use to tune a SQL statement for optimal performance. SQLT produces an HTML report with hyperlinks to all of the sections in the report.

Unlike Automatic Workload Repository or Statspack reports, SQLT works on individual SQL statements. SQLT is a collection of SQL, PL/SQL, and SQL*Plus files that collect, store, and display performance data.

Following are the supported Oracle versions for each SQLT version.

SQLT version	Oracle Database 21c	Oracle Database 19c
2018-07-25.v1	Supported	Supported
2018-03-31.v1	Not supported	Not supported
2016-04-29.v1	Not supported	Not supported

To download SQLT and access instructions for using it:

- Log in to your My Oracle Support account, and open the following documents:
- To download SQLT: Document 215187.1
- For SQLT usage instructions: Document 1614107.1
- For frequently asked questions about SQLT: Document 1454160.1
- For information about reading SQLT output: Document 1456176.1
- For interpreting the Main report: Document 1922234.1

Amazon RDS doesn't support the following SQLT methods:

- XPLORE
- XHUME

Prerequisites for SQLT

The following are prerequisites for using SQLT:

- You must remove users and roles that are required by SQLT, if they exist.

  The SQLT option creates the following users and roles on a DB instance:

  - SQLTXPLAIN user
  - SQLTXADMIN user
  - SQLT_USER_ROLE role

  If your DB instance has any of these users or roles, log in to the DB instance using a SQL client, and drop them using the following statements:

```
DROP USER SQLTXPLAIN CASCADE;
DROP USER SQLTXADMIN CASCADE;
DROP ROLE SQLT_USER_ROLE CASCADE;
```

- You must remove tablespaces that are required by SQLT, if they exist.

  The SQLT option creates the following tablespaces on a DB instance:

  - RDS_SQLT_TS
  - RDS_TEMP_SQLT_TS

  If your DB instance has these tablespaces, log in to the DB instance using a SQL client, and drop them.

SQLT option settings

SQLT can work with licensed features that are provided by the Oracle Tuning Pack and the Oracle Diagnostics Pack. The Oracle Tuning Pack includes the SQL Tuning Advisor, and the Oracle

Diagnostics Pack includes the Automatic Workload Repository. The SQLT settings enable or disable access to these features from SQLT.

Amazon RDS supports the following settings for the SQLT option.

Option setting	Valid values	Default value	Description
LICENSE_PACK	T, D, N	N	The Oracle Management Packs that you want to access with SQLT. Enter one of the following values: • T indicates that you have a license for the Oracle Tuning Pack and the Oracle Diagnostics Pack, and you want to access the SQL Tuning Advisor and Automatic Workload Repository from SQLT. • D indicates that you have a license for the Oracle Diagnostics Pack, and you want to access the Automatic Workload Repository from SQLT. • N indicates that you don't have a license for the Oracle Tuning Pack and the Oracle Diagnostics Pack, or that you have a license for one or both of them, but you don't want SQLT to access them. **Note** Amazon RDS does not provide licenses for these Oracle Management Packs. If you indicate that you want to use a pack that is not included in

Option setting	Valid values	Default value	Description
			your DB instance, you can use SQLT with the DB instance. However, SQLT can't access the pack, and the SQLT report doesn't include the data for the pack. For example, if you specify T, but the DB instance doesn't include the Oracle Tuning Pack, SQLT works on the DB instance, but the report it generates doesn't contain data related to the Oracle Tuning Pack.
VERSION	2016-04-29.v1 2018-03-31.v1 2018-07-25.v1	2016-04-29.v1	The version of SQLT that you want to install. **Note** For Oracle Database 19c and 21c, the only supported version is 2018-07-25.v1 . This version is the default for these releases.

Adding the SQLT option

The following is the general process for adding the SQLT option to a DB instance:

1. Create a new option group, or copy or modify an existing option group.
2. Add the SQLT option to the option group.
3. Associate the option group with the DB instance.

After you add the SQLT option, as soon as the option group is active, SQLT is active.

To add the SQLT option to a DB instance

1. Determine the option group that you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   a. For **Engine**, choose the Oracle edition that you want to use. The SQLT option is supported on all editions.

   b. For **Major engine version**, choose the version of your DB instance.

   For more information, see Creating an option group.
2. Add the **SQLT** option to the option group. For more information about adding options, see Adding an option to an option group.
3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.
4. (Optional) Verify the SQLT installation on each DB instance with the SQLT option.

   a. Use a SQL client to connect to the DB instance as the master user.

   For information about connecting to an Oracle DB instance using a SQL client, see Connecting to your Oracle DB instance.

   b. Run the following query:

```
SELECT sqltxplain.sqlt$a.get_param('tool_version') sqlt_version FROM DUAL;
```

   The query returns the current version of the SQLT option on Amazon RDS. `12.1.160429` is an example of a version of SQLT that is available on Amazon RDS.

5. Change the passwords of the users that are created by the SQLT option.

   a. Use a SQL client to connect to the DB instance as the master user.

   b. Run the following SQL statement to change the password for the SQLTXADMIN user:

```
ALTER USER SQLTXADMIN IDENTIFIED BY new_password ACCOUNT UNLOCK;
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

   c. Run the following SQL statement to change the password for the SQLTXPLAIN user:

```
ALTER USER SQLTXPLAIN IDENTIFIED BY new_password ACCOUNT UNLOCK;
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

> **Note**
>
> Upgrading SQLT requires uninstalling an older version of SQLT and then installing the new version. So, all SQLT metadata can be lost when you upgrade SQLT. A major version upgrade of a database also uninstalls and re-installs SQLT. An example of a major version upgrade is an upgrade from Oracle Database 19c to Oracle Database 21c.

Using SQLT

SQLT works with the Oracle SQL*Plus utility.

To use SQLT

1. Download the SQLT .zip file from Document 215187.1 on the My Oracle Support site.

   > **Note**
   >
   > You can't download SQLT 12.1.160429 from the My Oracle Support site. Oracle has deprecated this older version.

2. Unzip the SQLT .zip file.
3. From a command prompt, change to the `sqlt/run` directory on your file system.
4. From the command prompt, open SQL*Plus, and connect to the DB instance as the master user.

   For information about connecting to a DB instance using SQL*Plus, see Connecting to your Oracle DB instance.
5. Get the SQL ID of a SQL statement:

   ```
   SELECT SQL_ID FROM V$SQL WHERE SQL_TEXT='sql_statement';
   ```

   Your output is similar to the following:

   ```
   SQL_ID
   -------------
   chvsmttqjzjkn
   ```
6. Analyze a SQL statement with SQLT:

   ```
   START sqltxtract.sql sql_id sqltxplain_user_password
   ```

   For example, for the SQL ID `chvsmttqjzjkn`, enter the following:

```
START sqltxtract.sql chvsmttqjzjkn sqltxplain_user_password
```

SQLT generates the HTML report and related resources as a .zip file in the directory from which the SQLT command was run.

7. (Optional) To enable application users to diagnose SQL statements with SQLT, grant SQLT_USER_ROLE to each application user with the following statement:

```
GRANT SQLT_USER_ROLE TO application_user_name;
```

> **Note**
>
> Oracle does not recommend running SQLT with the SYS user or with users that have the DBA role. It is a best practice to run SQLT diagnostics using the application user's account, by granting SQLT_USER_ROLE to the application user.

Upgrading the SQLT option

With Amazon RDS for Oracle, you can upgrade the SQLT option from your existing version to a higher version. To upgrade the SQLT option, complete steps 1–3 in Using SQLT for the new version of SQLT. Also, if you granted privileges for the previous version of SQLT in step 7 of that section, grant the privileges again for the new SQLT version.

Upgrading the SQLT option results in the loss of the older SQLT version's metadata. The older SQLT version's schema and related objects are dropped, and the newer version of SQLT is installed. For more information about the changes in the latest SQLT version, see Document 1614201.1 on the My Oracle Support site.

> **Note**
>
> Version downgrades are not supported.

Modifying SQLT settings

After you enable SQLT, you can modify the `LICENSE_PACK` and `VERSION` settings for the option.

For more information about how to modify option settings, see Modifying an option setting. For more information about each setting, see SQLT option settings.

Removing the SQLT option

You can remove SQLT from a DB instance.

To remove SQLT from a DB instance, do one of the following:

- To remove SQLT from multiple DB instances, remove the SQLT option from the option group to which the DB instances belong. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
- To remove SQLT from a single DB instance, modify the DB instance and specify a different option group that doesn't include the SQLT option. You can specify the default (empty) option group or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Oracle Statspack

The Oracle Statspack option installs and enables the Oracle Statspack performance statistics feature. Oracle Statspack is a collection of SQL, PL/SQL, and SQL*Plus scripts that collect, store, and display performance data. For information about using Oracle Statspack, see Oracle Statspack in the Oracle documentation.

> **Note**
>
> Oracle Statspack is no longer supported by Oracle and has been replaced by the more advanced Automatic Workload Repository (AWR). AWR is available only for Oracle Enterprise Edition customers who have purchased the Diagnostics Pack. You can use Oracle Statspack with any Oracle DB engine on Amazon RDS. You can't run Oracle Statspack on Amazon RDS read replicas.

Setting up Oracle Statspack

To run Statspack scripts, you must add the Statspack option.

To set up Oracle Statspack

1. In a SQL client, log in to the Oracle DB with an administrative account.
2. Do either of the following actions, depending on whether Statspack is installed:
   - If Statspack is installed, and the PERFSTAT account is associated with Statspack, skip to Step 4.
   - If Statspack is not installed, and the PERFSTAT account exists, drop the account as follows:

     ```
     DROP USER PERFSTAT CASCADE;
     ```

     Otherwise, attempting to add the Statspack option generates an error and RDS-Event-0058.
3. Add the Statspack option to an option group. See Adding an option to an option group.

   Amazon RDS automatically installs the Statspack scripts on the DB instance and then sets up the PERFSTAT account.

4. Reset the password using the following SQL statement, replacing *pwd* with your new password:

```
ALTER USER PERFSTAT IDENTIFIED BY pwd ACCOUNT UNLOCK;
```

   You can log in using the PERFSTAT user account and run the Statspack scripts.

5. Grant the CREATE JOB privilege to the PERFSTAT account using the following statement:

```
GRANT CREATE JOB TO PERFSTAT;
```

6. Ensure that idle wait events in the PERFSTAT.STATS$IDLE_EVENT table are populated.

   Because of Oracle Bug 28523746, the idle wait events in PERFSTAT.STATS$IDLE_EVENT may not be populated. To ensure all idle events are available, run the following statement:

```
INSERT INTO PERFSTAT.STATS$IDLE_EVENT (EVENT)
SELECT NAME FROM V$EVENT_NAME WHERE WAIT_CLASS='Idle'
MINUS
SELECT EVENT FROM PERFSTAT.STATS$IDLE_EVENT;
COMMIT;
```

Generating Statspack reports

A Statspack report compares two snapshots.

To generate Statspack reports

1. In a SQL client, log in to the Oracle DB with the PERFSTAT account.
2. Create a snapshot using either of the following techniques:

   - Create a Statspack snapshot manually.
   - Create a job that takes a Statspack snapshot after a given time interval. For example, the following job creates a Statspack snapshot every hour:

```
VARIABLE jn NUMBER;
exec dbms_job.submit(:jn, 'statspack.snap;',SYSDATE,'TRUNC(SYSDATE
+1/24,''HH24'')');
COMMIT;
```

3. View the snapshots using the following query:

```
SELECT SNAP_ID, SNAP_TIME FROM STATS$SNAPSHOT ORDER BY 1;
```

4. Run the Amazon RDS procedure `rdsadmin.rds_run_spreport`, replacing *begin_snap* and *end_snap* with the snapshot IDs:

```
exec rdsadmin.rds_run_spreport(begin_snap,end_snap);
```

For example, the following command creates a report based on the interval between Statspack snapshots 1 and 2:

```
exec rdsadmin.rds_run_spreport(1,2);
```

The file name of the Statspack report includes the number of the two snapshots. For example, a report file created using Statspack snapshots 1 and 2 would be named `ORCL_spreport_1_2.lst`.

5. Monitor the output for errors.

Oracle Statspack performs checks before running the report. Therefore, you could also see error messages in the command output. For example, you might try to generate a report based on an invalid range, where the beginning Statspack snapshot value is larger than the ending value. In this case, the output shows the error message, but the DB engine does not generate an error file.

```
exec rdsadmin.rds_run_spreport(2,1);
*
ERROR at line 1:
ORA-20000: Invalid snapshot IDs. Find valid ones in perfstat.stats$snapshot.
```

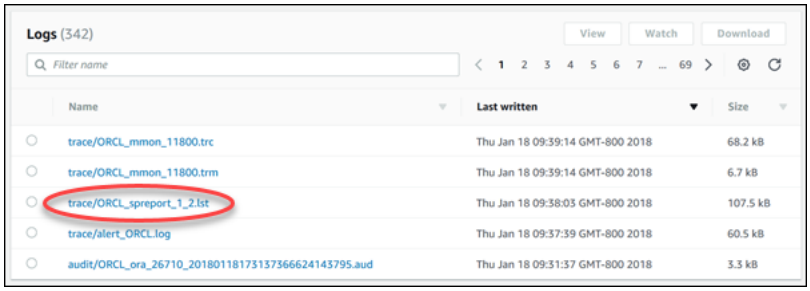
If you use an invalid number a Statspack snapshot, the output shows an error. For example, if you try to generate a report for snapshots 1 and 50, but snapshot 50 doesn't exist, the output shows an error.

```
exec rdsadmin.rds_run_spreport(1,50);
*
ERROR at line 1:
ORA-20000: Could not find both snapshot IDs
```

6. (Optional)

   To retrieve the report, call the trace file procedures, as explained in Working with Oracle trace files.

   Alternatively, download the Statspack report from the RDS console. Go to the **Log** section of the DB instance details and choose **Download**. The following example shows `trace/ORCL_spreport_1_2.lst`



   If an error occurs while generating a report, the DB engine uses the same naming conventions as for a report but with an extension of `.err`. For example, if an error occurred while creating a report using Statspack snapshots 1 and 7, the report file would be named `ORCL_spreport_1_7.err`. You can download the error report using the same techniques as for a standard Snapshot report.

Removing Statspack snapshots

To remove a range of Oracle Statspack snapshots, use the following command:

```
exec statspack.purge(begin snap, end snap);
```

Oracle time zone

To change the system time zone used by your Oracle DB instance, use the time zone option. For example, you might change the time zone of a DB instance to be compatible with an on-premises environment, or a legacy application. The time zone option changes the time zone at the host level. Changing the time zone impacts all date columns and values, including `SYSDATE` and `SYSTIMESTAMP`.

The time zone option differs from the `rdsadmin_util.alter_db_time_zone` command. The `alter_db_time_zone` command changes the time zone only for certain data types. The time zone option changes the time zone for all date columns and values. For more information about `alter_db_time_zone`, see Setting the database time zone. For more information about upgrade considerations, see Time zone considerations.

Restrictions for setting the time zone

The time zone option is a permanent and persistent option. Therefore, you can't do the following:

- Remove the option from an option group after you add the time zone option.
- Remove the option group from a DB instance after you add the group.
- Modify the time zone setting of the option to a different time zone.

Recommendations for setting the time zone

Before you add the time zone option to your production database, we strongly recommend that you do the following:

- Take a snapshot of your DB instance. If you accidentally set the time zone incorrectly, you must recover your DB instance to its previous time zone setting. For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.
- Add the time zone option to a test DB instance. Adding the time zone option can cause problems with tables that use the system date to add dates or times. We recommend that you analyze your data and applications on the test instance. This way you can assess the impact of changing the time zone on your production instance.

If your DB instance uses the default option group, then follow these steps:

1. Take a snapshot of your DB instance.

2. Add the time zone option to your DB instance.

If your DB instance currently uses a nondefault option group, then follow these steps:

1. Take a snapshot of your DB instance.
2. Create a new option group.
3. Add the time zone option to it, along with all other options that are currently associated with the existing option group.

   This prevents the existing options from being uninstalled while enabling the time zone option.
4. Add the option group to your DB instance.

Time zone option settings

Amazon RDS supports the following settings for the time zone option.

Option setting	Valid values	Description
`TIME_ZONE`	One of the available time zones. For the full list, see Available time zones.	The new time zone for your DB instance.

Adding the time zone option

Complete the following steps to add the time zone option to your DB instance:

1. (Recommended) Take a snapshot of your DB instance.
2. Do one of the following tasks:
   - Create a new option group from scratch. For more information, see Creating an option group.
   - Copy an existing option group using the AWS CLI or API. For more information, see Copying an option group.
   - Reuse an existing non-default option group. A best practice is to use an option group that isn't currently associated with any DB instances or snapshots.
3. Add the new option to the option group from the preceding step.

4. If the option group that is currently associated with your DB instance has options enabled, add these options to your new option group. This strategy prevents the existing options from being uninstalled while enabling the new option.
5. Add the new option group to your DB instance.

When you add the time zone option, a brief outage occurs while your DB instance is automatically restarted.

Console

To add the time zone option to an option group and associate it with a DB instance

1. In the RDS console, choose **Option groups.**
2. Choose the name of the option group to which you want to add the option.
3. Choose **Add option**.
4. For **Option name**, choose **Timezone**, and then configure the option settings.
5. Associate the option group with a new or existing DB instance:
   - For a new DB instance, apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, apply the option group by modifying the instance and attaching the new option group. When you add the new option to an existing DB instance, a brief outage occurs while your DB instance is automatically restarted. For more information, see Modifying an Amazon RDS DB instance.

AWS CLI

The following example uses the AWS CLI add-option-to-option-group command to add the `Timezone` option and the `TIME_ZONE` option setting to an option group called `myoptiongroup`. The time zone is set to `Africa/Cairo`.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name "myoptiongroup" \
    --options "OptionName=Timezone,OptionSettings=[{Name=TIME_ZONE,Value=Africa/
Cairo}]" \
```

```
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name "myoptiongroup" ^
    --options "OptionName=Timezone,OptionSettings=[{Name=TIME_ZONE,Value=Africa/
Cairo}]" ^
    --apply-immediately
```

Modifying time zone settings

The time zone option is a permanent and persistent option. You can't remove the option from an option group after you add it. You can't remove the option group from a DB instance after you add it. You can't modify the time zone setting of the option to a different time zone. If you set the time zone incorrectly, restore a snapshot of your DB instance from before you added the time zone option.

Removing the time zone option

The time zone option is a permanent and persistent option. You can't remove the option from an option group after you add it. You can't remove the option group from a DB instance after you add it. To remove the time zone option, restore a snapshot of your DB instance from before you added the time zone option.

Available time zones

You can use the following values for the time zone option.

Zone	Time zone
Africa	Africa/Cairo, Africa/Casablanca, Africa/Harare, Africa/Lagos, Africa/Luanda, Africa/Monrovia, Africa/Nairobi, Africa/Tripoli, Africa/Windhoek
America	America/Araguaina, America/Argentina/Buenos_Aires, America/Asuncion, America/Bogota, America/Caracas, America/Chicago, America/Chihuahua, America/Cuiaba, America/Denver, America/Detroit, America/Fortaleza, America/Godthab, America/Guatemala, America/Halifax, America/Lima, America/Los_Angeles, America/Manaus, America/Matamoros, America/M

Zone	Time zone
	exico_City, America/Monterrey, America/Montevideo, America/New_York, America/Phoenix, America/Santiago, America/Sao_Paulo, America/Tijuana, America/Toronto
Asia	Asia/Amman, Asia/Ashgabat, Asia/Baghdad, Asia/Baku, Asia/Bangkok, Asia/Beirut, Asia/Calcutta, Asia/Damascus, Asia/Dhaka, Asia/Hong_Kong, Asia/Irkutsk, Asia/Jakarta, Asia/Jerusalem, Asia/Kabul, Asia/Karachi, Asia/Kathmandu, Asia/Kolkata, Asia/Krasnoyarsk, Asia/Magadan, Asia/Manila, Asia/Muscat, Asia/Novosibirsk, Asia/Rangoon, Asia/Riyadh, Asia/Seoul, Asia/Shanghai, Asia/Singapore, Asia/Taipei, Asia/Tehran, Asia/Tokyo, Asia/Ulaanbaatar, Asia/Vladivostok, Asia/Yakutsk, Asia/Yerevan
Atlantic	Atlantic/Azores, Atlantic/Cape_Verde
Australia	Australia/Adelaide, Australia/Brisbane, Australia/Darwin, Australia/Eucla, Australia/Hobart, Australia/Lord_Howe, Australia/Perth, Australia/Sydney
Brazil	Brazil/DeNoronha, Brazil/East
Canada	Canada/Newfoundland, Canada/Saskatchewan
Etc	Etc/GMT-3
Europe	Europe/Amsterdam, Europe/Athens, Europe/Berlin, Europe/Dublin, Europe/Helsinki, Europe/Kaliningrad, Europe/London, Europe/Madrid, Europe/Moscow, Europe/Paris, Europe/Prague, Europe/Rome, Europe/Sarajevo
Pacific	Pacific/Apia, Pacific/Auckland, Pacific/Chatham, Pacific/Fiji, Pacific/Guam, Pacific/Honolulu, Pacific/Kiritimati, Pacific/Marquesas, Pacific/Samoa, Pacific/Tongatapu, Pacific/Wake
US	US/Alaska, US/Central, US/East-Indiana, US/Eastern, US/Pacific
UTC	UTC

Oracle time zone file autoupgrade

With the `TIMEZONE_FILE_AUTOUPGRADE` option, you can upgrade the current time zone file to the latest version on your RDS for Oracle DB instance.

Topics

- Overview of Oracle time zone files
- Strategies for updating your time zone file
- Downtime during the time zone file update
- Preparing to update the time zone file
- Adding the time zone file autoupgrade option
- Checking your data after the update of the time zone file

Overview of Oracle time zone files

An Oracle Database *time zone file* stores the following information:

- Offset from Coordinated Universal Time (UTC)
- Transition times for Daylight Saving Time (DST)
- Abbreviations for standard time and DST

Oracle Database supplies multiple versions of time zone files. When you create an Oracle database in an on-premises environment, you choose the time zone file version. For more information , see Choosing a Time Zone File in the *Oracle Database Globalization Support Guide*.

If the rules change for DST, Oracle publishes new time zone files. Oracle releases these new time zone files independently of the schedule for quarterly Release Updates (RUs) and Release Update Revisions (RURs). The time zone files reside on the database host in the directory `$ORACLE_HOME/oracore/zoneinfo/`. The time zone file names use the format DSTv*`version`*, as in DSTv35.

How the time zone file affects data transfer

In Oracle Database, the `TIMESTAMP WITH TIME ZONE` data type stores time stamp and time zone data. Data with the `TIMESTAMP WITH TIME ZONE` data type uses the rules in the associated time zone file version. Thus, existing `TIMESTAMP WITH TIME ZONE` data is affected when you update the time zone file.

Problems can occur when you transfer data between databases that use different versions of the time zone file. For example, if you import data from a source database with a higher time zone file version than the target database, the database issues the `ORA-39405` error. Previously, you had to work around this error by using either of the following techniques:

- Create an RDS for Oracle DB instance with the desired time zone file, export data from your source database, and then import it into the new database.
- Use AWS DMS or logical replication to migrate your data.

Automatic updates using the TIMEZONE_FILE_AUTOUPGRADE option

When the option group attached to your RDS for Oracle DB instance includes the `TIMEZONE_FILE_AUTOUPGRADE` option, RDS updates your time zone files automatically. By ensuring that your Oracle databases use the same time zone file version, you avoid time-consuming manual techniques when you move data between different environments. The `TIMEZONE_FILE_AUTOUPGRADE` option is supported for both container databases (CDBs) and non-CDBs.

When you add the `TIMEZONE_FILE_AUTOUPGRADE` option to your option group, you can choose whether to add the option immediately or during the maintenance window. After your DB instance applies the new option, RDS checks whether it can install a newer DSTv*`version`* file. The target DSTv*`version`* depends on the following:

- The minor engine version that your DB instance is currently running
- The minor engine version to which you want to upgrade your DB instance

For example, your current time zone file version might be DSTv33. When RDS applies the update to your option group, it might determine that DSTv34 is currently available on your DB instance file system. RDS will then update your time zone file to DSTv34 automatically.

To find the available DST versions in the supported RDS release updates, look at the patches in Release notes for Amazon Relational Database Service (Amazon RDS) for Oracle. For example, version 19.0.0.0.ru-2022-10.rur-2022-10.r1 lists patch 34533061: RDBMS - DSTV39 UPDATE - TZDATA2022C.

Strategies for updating your time zone file

Upgrading your DB engine and adding the TIMEZONE_FILE_AUTOUPGRADE option to an option group are separate operations. Adding the TIMEZONE_FILE_AUTOUPGRADE option initiates the update of your time zone file if a more current one is available. You run the following commands (only relevant options are shown) either immediately or at the next maintenance window:

- Upgrade your DB engine only using the following RDS CLI command:

```
modify-db-instance --engine-version name ...
```

- Add the TIMEZONE_FILE_AUTOUPGRADE option only using the following CLI command:

```
add-option-to-option-group --option-group-name name --options
 OptionName=TIMEZONE_FILE_AUTOUPGRADE ...
```

- Upgrade your DB engine and add a new option group to your instance using the following CLI command:

```
modify-db-instance --engine-version name --option-group-name name ...
```

Your update strategy depends on whether you want to upgrade your database and time zone file together or perform just one of these operations. Keep in mind that if you update your option group and then upgrade your DB engine in separate API operations, it's possible for a time zone file update to be currently in progress when you upgrade your DB engine.

The examples in this section assume the following:

- You have not yet added TIMEZONE_FILE_AUTOUPGRADE to the option group currently associated with your DB instance.
- Your DB instance uses database version 19.0.0.0.ru-2019-07.rur-2019-07.r1 and time zone file DSTv33.
- Your DB instance file system includes file DSTv34.
- Release update 19.0.0.0.ru-2022-10.rur-2022-10.r1 includes DSTv35.

To update your time zone file, you can use the following strategies.

Topics

- Update the time zone file without upgrading the engine
- Upgrade the time zone file and DB engine version
- Upgrade your DB engine version without updating the time zone file

Update the time zone file without upgrading the engine

In this scenario, your database is using DSTv33, but DSTv34 is available on your DB instance file system. You want to update the time zone file used by your DB instance from DSTv33 to DSTv34, but you don't want to upgrade your engine to a new minor version, which includes DSTv35.

In an `add-option-to-option-group` command, add `TIMEZONE_FILE_AUTOUPGRADE` to the option group used by your DB instance. Specify whether to add the option immediately or defer it to the maintenance window. After applying the `TIMEZONE_FILE_AUTOUPGRADE` option, RDS does the following:

1. Checks for a new DST version.
2. Determines that DSTv34 is available on the file system.
3. Updates the time zone file immediately.

Upgrade the time zone file and DB engine version

In this scenario, your database is using DSTv33, but DSTv34 is available on your DB instance file system. You want to upgrade your DB engine to minor version 19.0.0.0.ru-2022-10.rur-2022-10.r1, which includes DSTv35, and update your time zone file to DSTv35 during the engine upgrade. Thus, your goal is to skip DSTv34 and update your time zone files directly to DSTv35.

To upgrade the engine and time zone file together, run `modify-db-instance` with the `--option-group-name` and `--engine-version` options. You can run the command immediately or defer it to maintenance window. `In --option-group-name`, specify an option group that includes the `TIMEZONE_FILE_AUTOUPGRADE` option. For example:

```
aws rds modify-db-instance
    --db-instance-identifier my-instance \
    --engine-version new-version \
    ----option-group-name og-with-timezone-file-autoupgrade \
    --apply-immediately
```

RDS begins upgrading your engine to 19.0.0.0.ru-2022-10.rur-2022-10.r1. After applying the TIMEZONE_FILE_AUTOUPGRADE option, RDS checks for a new DST version, sees that DSTv35 is available in 19.0.0.0.ru-2022-10.rur-2022-10.r1, and immediately starts the update to DSTv35.

To upgrade your engine immediately and then upgrade your a timezone file, perform the operations in sequence:

1. Upgrade your DB engine only using the following CLI command:

```
aws rds modify-db-instance \
    --db-instance-identifier my-instance \
    --engine-version new-version \
    --apply-immediately
```

2. Add the TIMEZONE_FILE_AUTOUPGRADE option to the option group attached to your instance using the following CLI command:

```
aws rds add-option-to-option-group \
    --option-group-name og-in-use-by-your-instance \
    --options OptionName=TIMEZONE_FILE_AUTOUPGRADE \
    --apply-immediately
```

Upgrade your DB engine version without updating the time zone file

In this scenario, your database is using DSTv33, but DSTv34 is available on your DB instance file system. You want to upgrade your DB engine to version 19.0.0.0.ru-2022-10.rur-2022-10.r1, which includes DSTv35, but retain time zone file DSTv33. You might choose this strategy for the following reasons:

- Your data doesn't use the TIMESTAMP WITH TIME ZONE data type.
- Your data uses the TIMESTAMP WITH TIME ZONE data type, but your data is not affected by the time zone changes.
- You want to postpone updating the time zone file because you can't tolerate the extra downtime.

Your strategy depends on which of the following possibilities are true:

- Your DB instance isn't associated with an option group that includes `TIMEZONE_FILE_AUTOUPGRADE`. In your `modify-db-instance` command, don't specify a new option group so that RDS doesn't update your time zone file.
- Your DB instance is currently associated with an option group that includes `TIMEZONE_FILE_AUTOUPGRADE`. Within a single `modify-db-instance` command, associate your DB instance with an option group that doesn't include `TIMEZONE_FILE_AUTOUPGRADE` and upgrade your DB engine to 19.0.0.0.ru-2022-10.rur-2022-10.r1.

Downtime during the time zone file update

When RDS updates your time zone file, existing data that uses `TIMESTAMP WITH TIME ZONE` might change. In this case, your primary consideration is downtime.

> **Warning**
>
> If you add the `TIMEZONE_FILE_AUTOUPGRADE` option, your engine upgrade might have prolonged downtime. Updating time zone data for a large database might take hours or even days.

The length of the time zone file update depends on factors such as the following:

- The amount of `TIMESTAMP WITH TIME ZONE` data in your database
- The DB instance configuration
- The DB instance class
- The storage configuration
- The database configuration
- The database parameter settings

Additional downtime can occur when you do the following:

- Add the option to the option group when the DB instance uses an outdated time zone file
- Upgrade the Oracle database engine when the new engine version contains a new version of the time zone file

> **Note**
>
> During the time zone file update, RDS for Oracle calls `PURGE DBA_RECYCLEBIN`.

Preparing to update the time zone file

A time zone file upgrade has two separate phases: prepare and upgrade. While not required, we strongly recommend that you perform the prepare step. In this step, you find out which data will be affected by running the PL/SQL procedure `DBMS_DST.FIND_AFFECTED_TABLES`. For more information about the prepare window, see Upgrading the Time Zone File and Timestamp with Time Zone Data in the Oracle Database documentation.

To prepare to update the time zone file

1. Connect to your Oracle database using a SQL client.
2. Determine the current timezone file version used.

```
SELECT * FROM V$TIMEZONE_FILE;
```

3. Determine the latest timezone file version available on your DB instance.

```
SELECT DBMS_DST.GET_LATEST_TIMEZONE_VERSION FROM DUAL;
```

4. Determine the total size of tables that have columns of type `TIMESTAMP WITH LOCAL TIME ZONE` or `TIMESTAMP WITH TIME ZONE`.

```
SELECT SUM(BYTES)/1024/1024/1024 "Total_size_w_TSTZ_columns_GB"
FROM   DBA_SEGMENTS
WHERE  SEGMENT_TYPE LIKE 'TABLE%'
AND    (OWNER, SEGMENT_NAME) IN
         (SELECT OWNER, TABLE_NAME
          FROM   DBA_TAB_COLUMNS
          WHERE  DATA_TYPE LIKE 'TIMESTAMP%TIME ZONE');
```

5. Determine the names and sizes of segments that have columns of type `TIMESTAMP WITH LOCAL TIME ZONE` or `TIMESTAMP WITH TIME ZONE`.

```
SELECT OWNER, SEGMENT_NAME, SUM(BYTES)/1024/1024/1024
 "SEGMENT_SIZE_W_TSTZ_COLUMNS_GB"
```

```
FROM   DBA_SEGMENTS
WHERE  SEGMENT_TYPE LIKE 'TABLE%'
AND    (OWNER, SEGMENT_NAME) IN
         (SELECT OWNER, TABLE_NAME
          FROM   DBA_TAB_COLUMNS
          WHERE  DATA_TYPE LIKE 'TIMESTAMP%TIME ZONE')
GROUP BY OWNER, SEGMENT_NAME;
```

6. Run the prepare step.

   - The procedure DBMS_DST.CREATE_AFFECTED_TABLE creates a table to store any affected data. You pass the name of this table to the DBMS_DST.FIND_AFFECTED_TABLES procedure. For more information, see CREATE_AFFECTED_TABLE Procedure in the Oracle Database documentation.
   - This procedure CREATE_ERROR_TABLE creates a table to log errors. For more information, see CREATE_ERROR_TABLE Procedure in the Oracle Database documentation.

   The following example creates the affected data and error tables, and finds all affected tables.

```
EXEC DBMS_DST.CREATE_ERROR_TABLE('my_error_table')
EXEC DBMS_DST.CREATE_AFFECTED_TABLE('my_affected_table')

EXEC DBMS_DST.BEGIN_PREPARE(new_version);
EXEC DBMS_DST.FIND_AFFECTED_TABLES('my_affected_table', TRUE, 'my_error_table');
EXEC DBMS_DST.END_PREPARE;

SELECT * FROM my_affected_table;
SELECT * FROM my_error_table;
```

7. Query the affected and error tables.

```
SELECT * FROM my_affected_table;
SELECT * FROM my_error_table;
```

Adding the time zone file autoupgrade option

When you add the option to an option group, the option group is in one of the following states:

- An existing option group is currently attached to at least one DB instance. When you add the option, all DB instances that use this option group automatically restart. This causes a brief outage.
- An existing option group is not attached to any DB instance. You plan to add the option and then associate the existing option group with existing DB instances or with a new DB instance.
- You create a new option group and add the option. You plan to associate the new option group with existing DB instances or with a new DB instance.

Console

To add the time zone file autoupgrade option to a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Option groups**.
3. Determine the option group you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   a. For **Engine** choose the Oracle Database edition for your DB instance.

   b. For **Major engine version** choose the version of your DB instance.

   For more information, see Creating an option group.
4. Choose the option group that you want to modify, and then choose **Add option**.
5. In the **Add option** window, do the following:

   a. Choose **TIMEZONE_FILE_AUTOUPGRADE**.

   b. To enable the option on all associated DB instances as soon as you add it, for **Apply Immediately**, choose **Yes**. If you choose **No** (the default), the option is enabled for each associated DB instance during its next maintenance window.
6. When the settings are as you want them, choose **Add option**.

AWS CLI

The following example uses the AWS CLI add-option-to-option-group command to add the `TIMEZONE_FILE_AUTOUPGRADE` option to an option group called `myoptiongroup`.

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name "myoptiongroup" \
    --options "OptionName=TIMEZONE_FILE_AUTOUPGRADE" \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name "myoptiongroup" ^
    --options "OptionName=TIMEZONE_FILE_AUTOUPGRADE" ^
    --apply-immediately
```

Checking your data after the update of the time zone file

We recommend that you check your data after you update the time zone file. During the prepare step, RDS for Oracle automatically creates the following tables:

- `rdsadmin.rds_dst_affected_tables` – Lists the tables that contain data affected by the update
- `rdsadmin.rds_dst_error_table` – Lists the errors generated during the update

These tables are independent of any tables that you create in the prepare window. To see the results of the update, query the tables as follows.

```
SELECT * FROM rdsadmin.rds_dst_affected_tables;
SELECT * FROM rdsadmin.rds_dst_error_table;
```

For more information about the schema for the affected data and error tables, see FIND_AFFECTED_TABLES Procedure in the Oracle documentation.

Oracle Transparent Data Encryption

Amazon RDS supports Oracle Transparent Data Encryption (TDE), a feature of the Oracle Advanced Security option available in Oracle Enterprise Edition. This feature automatically encrypts data before it is written to storage and automatically decrypts data when the data is read from storage. This option is only supported for the Bring Your Own License (BYOL) model.

TDE is useful in scenarios where you need to encrypt sensitive data in case data files and backups are obtained by a third party. TDE is also useful when you need to comply with security-related regulations.

A detailed explanation about TDE in Oracle Database is beyond the scope of this guide. For information, see the following Oracle Database resources:

- Introduction to Transparent Data Encryption in the Oracle Database documentation
- Oracle advanced security in the Oracle Database documentation
- Oracle advanced security Transparent Data Encryption best practices, which is an Oracle whitepaper

For more information about using TDE with RDS for Oracle, see the following blogs:

- Oracle Database Encryption Options on Amazon RDS
- Migrate a cross-account TDE-enabled Amazon RDS for Oracle DB instance with reduced downtime using AWS DMS

TDE encryption modes

Oracle Transparent Data Encryption supports two encryption modes: TDE tablespace encryption and TDE column encryption. TDE tablespace encryption is used to encrypt entire application tables. TDE column encryption is used to encrypt individual data elements that contain sensitive data. You can also apply a hybrid encryption solution that uses both TDE tablespace and column encryption.

> **Note**
>
> Amazon RDS manages the Oracle Wallet and TDE master key for the DB instance. You do not need to set the encryption key using the command `ALTER SYSTEM set encryption key`.

After you enable the TDE option, you can check the status of the Oracle Wallet by using the following command:

```
SELECT * FROM v$encryption_wallet;
```

To create an encrypted tablespace, use the following command:

```
CREATE TABLESPACE encrypt_ts ENCRYPTION DEFAULT STORAGE (ENCRYPT);
```

To specify the encryption algorithm, use the following command:

```
CREATE TABLESPACE encrypt_ts ENCRYPTION USING 'AES256' DEFAULT STORAGE (ENCRYPT);
```

The previous statements for encrypting a tablespace are the same as you would use on an on-premises Oracle database.

Restrictions for the TDE option

The TDE option is permanent and persistent. After you associate your DB instance with an option group that has the TDE option enabled, you can't do the following actions:

- Disable the TDE option in the currently associated option group.
- Associate your DB instance with a different option group that doesn't include the TDE option.
- Share a DB snapshot that uses the TDE option. For more information about sharing DB snapshots, see Sharing a DB snapshot for Amazon RDS.

For more information about persistent and permanent options, see Persistent and permanent options.

Determining whether your DB instance is using TDE

You might want to determine whether your DB instance is associated with an option group that has the TDE option enabled. To view the option group that a DB instance is associated with, use the RDS console, the describe-db-instance AWS CLI command, or the API operation DescribeDBInstances.

Adding the TDE option

To add the TDE option to your DB instance, complete the following steps:

1. (Recommended) Take a snapshot of your DB instance.
2. Do one of the following tasks:
   - Create a new option group from scratch. For more information, see Creating an option group.
   - Copy an existing option group using the AWS CLI or API. For more information, see Copying an option group.
   - Reuse an existing non-default option group. A best practice is to use an option group that isn't currently associated with any DB instances or snapshots.
3. Add the new option to the option group from the preceding step.
4. If the option group that is currently associated with your DB instance has options enabled, add these options to your new option group. This strategy prevents the existing options from being uninstalled while enabling the new option.
5. Add the new option group to your DB instance.

Console

To add the TDE option to an option group and associate it with your DB instance

1. In the RDS console, choose **Option groups.**
2. Choose the name of the option group to which you want to add the option.
3. Choose **Add option**.
4. For **Option name**, choose **TDE**, and then configure the option settings.
5. Choose **Add option**.

> **Important**
>
> If you add the **TDE** option to an option group that is currently attached to one or more DB instances, a brief outage occurs while all the DB instances are automatically restarted.

   For more information about adding options, see Adding an option to an option group.
6. Associate the option group with a new or existing DB instance:
   - For a new DB instance, apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.

- For an existing DB instance, apply the option group by modifying the instance and attaching the new option group. The DB instance doesn't restart as part of this operation. For more information, see Modifying an Amazon RDS DB instance.

AWS CLI

In the following example, you use the AWS CLI add-option-to-option-group command to add the TDE option to an option group called `myoptiongroup`. For more information, see Getting started: Flink 1.13.2 .

For Linux, macOS, or Unix:

```
aws rds add-option-to-option-group \
    --option-group-name "myoptiongroup" \
    --options "OptionName=TDE" \
    --apply-immediately
```

For Windows:

```
aws rds add-option-to-option-group ^
    --option-group-name "myoptiongroup" ^
    --options "OptionName=TDE" ^
    --apply-immediately
```

Copying your data to a DB instance that doesn't include the TDE option

You can't remove the TDE option from a DB instance or associate it with an option group that doesn't include the TDE option. To migrate your data to an instance that doesn't include the TDE option, do the following:

1. Decrypt the data on your DB instance.
2. Copy the data to a new DB instance that is not associated with an option group that has TDE enabled.
3. Delete your original DB instance.

You can use the same name for the new instance as the previous DB instance.

Considerations when using TDE with Oracle Data Pump

You can use Oracle Data Pump to import or export encrypted dump files. Amazon RDS supports the password encryption mode (`ENCRYPTION_MODE=PASSWORD`) for Oracle Data Pump. Amazon RDS does not support transparent encryption mode (`ENCRYPTION_MODE=TRANSPARENT`) for Oracle Data Pump. For more information, see Importing using Oracle Data Pump.

Oracle UTL_MAIL

Amazon RDS supports Oracle UTL_MAIL through the use of the UTL_MAIL option and SMTP servers. You can send email directly from your database by using the UTL_MAIL package. Amazon RDS supports UTL_MAIL for the following versions of Oracle:

- Oracle Database 21c (21.0.0.0), all versions
- Oracle Database 19c (19.0.0.0), all versions

The following are some limitations to using UTL_MAIL:

- UTL_MAIL does not support Transport Layer Security (TLS) and therefore emails are not encrypted.

  To connect securely to remote SSL/TLS resources by creating and uploading custom Oracle wallets, follow the instructions in Configuring UTL_HTTP access using certificates and an Oracle wallet.

  The specific certificates that are required for your wallet vary by service. For AWS services, these can typically be found in the Amazon trust services repository.
- UTL_MAIL does not support authentication with SMTP servers.
- You can only send a single attachment in an email.
- You can't send attachments larger than 32 K.
- You can only use ASCII and Extended Binary Coded Decimal Interchange Code (EBCDIC) character encodings.
- SMTP port (25) is throttled based on the elastic network interface owner's policies.

When you enable UTL_MAIL, only the master user for your DB instance is granted the execute privilege. If necessary, the master user can grant the execute privilege to other users so that they can use UTL_MAIL.

Important

We recommend that you enable Oracle's built-in auditing feature to track the use of UTL_MAIL procedures.

Prerequisites for Oracle UTL_MAIL

The following are prerequisites for using Oracle UTL_MAIL:

- One or more SMTP servers, and the corresponding IP addresses or public or private Domain Name Server (DNS) names. For more information about private DNS names resolved through a custom DNS server, see Setting up a custom DNS server.

Adding the Oracle UTL_MAIL option

The general process for adding the Oracle UTL_MAIL option to a DB instance is the following:

1. Create a new option group, or copy or modify an existing option group.
2. Add the option to the option group.
3. Associate the option group with the DB instance.

After you add the UTL_MAIL option, as soon as the option group is active, UTL_MAIL is active.

To add the UTL_MAIL option to a DB instance

1. Determine the option group you want to use. You can create a new option group or use an existing option group. If you want to use an existing option group, skip to the next step. Otherwise, create a custom DB option group with the following settings:

   a. For **Engine**, choose the edition of Oracle you want to use.

   b. For **Major engine version**, choose the version of your DB instance.

   For more information, see Creating an option group.
2. Add the **UTL_MAIL** option to the option group. For more information about adding options, see Adding an option to an option group.
3. Apply the option group to a new or existing DB instance:

   - For a new DB instance, you apply the option group when you launch the instance. For more information, see Creating an Amazon RDS DB instance.
   - For an existing DB instance, you apply the option group by modifying the instance and attaching the new option group. For more information, see Modifying an Amazon RDS DB instance.

Using Oracle UTL_MAIL

After you enable the UTL_MAIL option, you must configure the SMTP server before you can begin using it.

You configure the SMTP server by setting the SMTP_OUT_SERVER parameter to a valid IP address or public DNS name. For the SMTP_OUT_SERVER parameter, you can specify a comma-separated list of the addresses of multiple servers. If the first server is unavailable, UTL_MAIL tries the next server, and so on.

You can set the default SMTP_OUT_SERVER for a DB instance by using a DB parameter group. You can set the SMTP_OUT_SERVER parameter for a session by running the following code on your database on your DB instance.

```
ALTER SESSION SET smtp_out_server = mailserver.domain.com:25;
```

After the UTL_MAIL option is enabled, and your SMTP_OUT_SERVER is configured, you can send mail by using the SEND procedure. For more information, see UTL_MAIL in the Oracle documentation.

Removing the Oracle UTL_MAIL option

You can remove Oracle UTL_MAIL from a DB instance.

To remove UTL_MAIL from a DB instance, do one of the following:

- To remove UTL_MAIL from multiple DB instances, remove the UTL_MAIL option from the option group they belong to. This change affects all DB instances that use the option group. For more information, see Removing an option from an option group.
- To remove UTL_MAIL from a single DB instance, modify the DB instance and specify a different option group that doesn't include the UTL_MAIL option. You can specify the default (empty) option group, or a different custom option group. For more information, see Modifying an Amazon RDS DB instance.

Troubleshooting

The following are issues you might encounter when you use UTL_MAIL with Amazon RDS.

- Throttling. SMTP port (25) is throttled based on the elastic network interface owner's policies. If you can successfully send email by using UTL_MAIL, and you see the error `ORA-29278: SMTP transient error: 421 Service not available`, you are possibly being throttled. If you experience throttling with email delivery, we recommend that you implement a backoff algorithm. For more information about backoff algorithms, see Error retries and exponential backoff in AWS and How to handle a "throttling – Maximum sending rate exceeded" error.

  You can request that this throttle be removed. For more information, see How do I remove the throttle on port 25 from my EC2 instance?.

Oracle XML DB

Oracle XML DB adds native XML support to your DB instance. With XML DB, you can store and retrieve structured or unstructured XML and relational data. The XML DB protocol server isn't supported on RDS for Oracle.

XML DB is preinstalled on Oracle Database 12c and higher. Thus, you don't need to use an option group to explicitly install XML DB as an additional feature.

To learn how to configure and use XML DB, see Oracle XML DB Developer's Guide in the Oracle Database documentation.

Upgrading the RDS for Oracle DB engine

When Amazon RDS supports a new version of Oracle Database, you can upgrade your DB instances to the new version. For information about which Oracle versions are available on Amazon RDS, see *Amazon RDS for Oracle Release Notes*.

> **Important**
>
> RDS for Oracle Databases 11g, 12c, and 18c are no longer supported. If you maintain Oracle Database 11g, 12c, or 18c snapshots, you can upgrade them to a later release. For more information, see Upgrading an Oracle DB snapshot.

Topics

- Overview of RDS for Oracle engine upgrades
- Oracle major version upgrades
- Oracle minor version upgrades
- Considerations for Oracle database upgrades
- Testing an Oracle DB upgrade
- Upgrading the version of an RDS for Oracle DB instance
- Upgrading an Oracle DB snapshot

Overview of RDS for Oracle engine upgrades

Before upgrading your RDS for Oracle DB instance, familiarize yourself with the following concepts.

Topics

- Major and minor version upgrades
- Support dates and mandatory upgrades for RDS for Oracle
- Oracle engine version management
- Automatic snapshots during engine upgrades
- Oracle upgrades in a Multi-AZ deployment
- Oracle upgrades of read replicas

Major and minor version upgrades

Major versions are major releases of Oracle Database that occur every 1-2 years. Oracle Database 19c and Oracle Database 21c are major releases.

Every quarter, RDS for Oracle releases new minor engine versions for every supported major engine. A Release Update (RU) engine version incorporates bug fixes from Oracle by including the RU patches for the specified quarter. For example, 21.0.0.0.ru-2024-10.rur-2024-10.r1 is a minor version of Oracle Database 21c that incorporates the October 2024 RU.

A Spatial Patch Bundle (SPB) engine version contains RU patches and patches specific to Oracle Spatial. For example, 19.0.0.0.ru-2025-01.spb-1.r1 is a minor engine version that contains the RU patches in engine version 19.0.0.0.ru-2025-01.rur-2025-01.r1 plus Spatial patches. Typically, RDS for Oracle releases SPBs 2–3 weeks after the corresponding RU. For an explanation of the differences between RUs and SPBs, see Release Updates (RUs) and Spatial Patch Bundles (SPBs). For information about supported RUs and SPBs, see Release notes for Amazon Relational Database Service (Amazon RDS) for Oracle.

RDS for Oracle supports the following upgrades to a DB instance.

Upgrade type	Application compatibility	Upgrade methods	Sample upgrade path
Major version	A major version upgrade can introduce changes that aren't compatible with existing applications.	Manual only	From Oracle Database 19c to Oracle Database 21c
Minor version	A minor version upgrade includes only changes that are backward-compatible with existing applications.	Automatic or manual	From 21.0.0.0.ru-2023-07.rur-2022-07.r1 to 21.0.0.0.ru-2023-10.rur-2022-10.r1

Important

When you upgrade your DB engine, an outage occurs. The duration of the outage depends on your engine version and DB instance size.

Make sure that you thoroughly test any upgrade to verify that your applications work correctly before applying the upgrade to your production databases. For more information, see Testing an Oracle DB upgrade.

Support dates and mandatory upgrades for RDS for Oracle

Database versions of RDS for Oracle have expected support dates. When a major or minor version of an RDS for Oracle DB engine nears its end-of-support date, RDS begins mandatory upgrades, also known as *forced upgrades*. RDS publishes the following information:

- A recommendation for you to begin manually upgrading instances on deprecated versions to supported versions
- A date after which you can no longer create instances on the unsupported versions
- A date on which RDS begins to upgrade your instances to supported versions automatically during maintenance windows
- A date on which RDS begins to upgrade your instances to supported versions automatically outside of maintenance windows

Important

Forced upgrades can have unexpected consequences for CloudFormation stacks. If you rely on RDS to upgrade your DB instances automatically, you might encounter issues with CloudFormation.

This section contains the following topics:

Topics

- Support dates for major releases of RDS for Oracle
- Support dates for minor versions of RDS for Oracle

Support dates for major releases of RDS for Oracle

RDS for Oracle major versions remain available at least until the end of support date for the corresponding Oracle Database release version. You can use the following dates to plan your

testing and upgrade cycles. These dates represent the earliest date that an upgrade to a newer version might be required. If Amazon extends support for an RDS for Oracle version for longer than originally stated, we plan to update this table to reflect the later date.

> **Note**
>
> You can view the major versions of your Oracle databases by running the describe-db-major-engine-versions AWS CLI command or by using the DescribeDBMajorEngineVersions RDS API operation.

Oracle Database major release version	Expected date for upgrading to a newer version
Oracle Database 19c	December 31, 2029 with BYOL Premier Support (fees waived for Extended Support) December 31, 2032 with BYOL Extended Support (extra cost) or an Unlimited License Agreement December 31, 2029 with License Included (LI)
Oracle Database 21c	July 31, 2027 (not available for Extended Support)

RDS notifies you at least 12 months before you need to upgrade to a newer major version. The notification describes the upgrade process, including the timing of important milestones, the effect on your DB instances, and recommended actions. We recommend that you thoroughly test your applications with new RDS for Oracle versions before you upgrade your database to a major version.

After this advance notification period, an automatic upgrade to the subsequent major version might be applied to any RDS for Oracle DB instance still running the older version. If so, the upgrade is started during scheduled maintenance windows.

For more information, see Release Schedule of Current Database Releases in My Oracle Support.

Support dates for minor versions of RDS for Oracle

In some cases, we end support for minor versions of major releases in RDS for Oracle. RDS notifies you at least 6 months before you need to upgrade to a newer minor version. The notification describes the upgrade process, including the timing of important milestones, the effect on the DB instances running the deprecated minor version, and recommended actions. We recommend that you thoroughly test your applications with new RDS for Oracle versions before you upgrade your database to a new minor version.

For more information about deprecated and desupported minor versions, see Release notes for Amazon Relational Database Service (Amazon RDS) for Oracle.

Oracle engine version management

With DB engine version management, you control when and how the database engine is patched and upgraded. You get the flexibility to maintain compatibility with database engine patch versions. You can also test new patch versions of RDS for Oracle to ensure they work with your application before deploying them in production. In addition, you upgrade the versions on your own terms and timelines.

> **Note**
>
> Amazon RDS periodically aggregates official Oracle database patches using an Amazon RDS-specific DB engine version. To see a list of which Oracle patches are contained in an Amazon RDS Oracle-specific engine version, go to *Amazon RDS for Oracle Release Notes*.

Automatic snapshots during engine upgrades

During upgrades of an Oracle DB instance, snapshots offer protection against upgrade issues. If the backup retention period for your DB instance is greater than 0, Amazon RDS takes the following DB snapshots during the upgrade:

1. A snapshot of the DB instance before any upgrade changes have been made. If the upgrade fails, you can restore this snapshot to create a DB instance running the old version.
2. A snapshot of the DB instance after the upgrade completes.

> **Note**
>
> To change your backup retention period, see Modifying an Amazon RDS DB instance.

After an upgrade, you can't revert to the previous engine version. However, you can create a new Oracle DB instance by restoring the pre-upgrade snapshot.

Oracle upgrades in a Multi-AZ deployment

If your DB instance is in a Multi-AZ deployment, Amazon RDS upgrades both the primary and standby replicas. If no operating system updates are required, the primary and standby upgrades occur simultaneously. The instances are not available until the upgrade completes.

If operating system updates are required in a Multi-AZ deployment, Amazon RDS applies the updates when you request the database upgrade. Amazon RDS performs the following steps:

1. Updates the operating system on the current standby DB instance.
2. Fails over the primary DB instance to the standby DB instance.
3. Upgrades the database version on the new primary DB instance, which was formerly the standby instance. The primary database is unavailable during the upgrade.
4. Updates the operating system on the new standby DB instance, which was formerly the primary DB instance.
5. Upgrades the database version on the new standby DB instance.
6. Fails over the new primary DB instance back to the original primary DB instance, and the new standby DB instance back to the original standby DB instance. Thus, Amazon RDS returns the replication configuration to its original state.

Oracle upgrades of read replicas

The Oracle DB engine version of the source DB instance and all of its read replicas must be the same. Amazon RDS performs the upgrade in the following stages:

1. Upgrades the source DB instance. The read replicas are available during this stage.
2. Upgrades the read replicas in parallel, regardless of the replica maintenance windows. The source DB is available during this stage.

For major version upgrades of cross-Region read replicas, Amazon RDS performs additional actions:

- Generates an option group for the target version automatically
- Copies all options and option settings from the original option group to the new option group
- Associates the upgraded cross-Region read replica with the new option group

Oracle major version upgrades

To perform a major version upgrade, modify the DB instance manually. Major version upgrades don't occur automatically.

> **Important**
>
> Make sure that you thoroughly test any upgrade to verify that your applications work correctly before applying the upgrade to your production databases. For more information, see Testing an Oracle DB upgrade.

Topics

- Supported versions for major upgrades
- Supported instance classes for major upgrades
- Gathering statistics before major upgrades
- Allowing major upgrades

Supported versions for major upgrades

Amazon RDS supports the following major version upgrades.

Current version	Upgrade supported
19.0.0.0 using the CDB architecture	21.0.0.0

A major version upgrade of Oracle Database must upgrade to a Release Update (RU) that was released in the same month or later. Major version downgrades aren't supported for any Oracle Database versions.

Supported instance classes for major upgrades

Your current Oracle DB instance might run on a DB instance class that isn't supported for the version to which you are upgrading. In this case, before you upgrade, migrate the DB instance to a supported DB instance class. For more information about the supported DB instance classes for each version and edition of Amazon RDS for Oracle, see DB instance classes.

Gathering statistics before major upgrades

Before you perform a major version upgrade, Oracle recommends that you gather optimizer statistics on the DB instance that you are upgrading. This action can reduce DB instance downtime during the upgrade.

To gather optimizer statistics, connect to the DB instance as the master user, and run the `DBMS_STATS.GATHER_DICTIONARY_STATS` procedure, as in the following example.

```
EXEC DBMS_STATS.GATHER_DICTIONARY_STATS;
```

For more information, see GATHER_DICTIONARY_STATS Procedure in the Oracle documentation.

Allowing major upgrades

A major engine version upgrade might be incompatible with your application. The upgrade is irreversible. If you specify a major version for the EngineVersion parameter that is different from the current major version, you must allow major version upgrades.

If you upgrade a major version using the CLI command modify-db-instance, specify `--allow-major-version-upgrade`. This setting isn't persistent, so you must specify `--allow-major-version-upgrade` whenever you perform a major upgrade. This parameter has no impact on upgrades of minor engine versions. For more information, see Upgrading a DB instance engine version.

If you upgrade a major version using the console, you don't need to choose an option to allow the upgrade. Instead, the console displays a warning that major upgrades are irreversible.

Oracle minor version upgrades

In RDS for Oracle, a minor version upgrade is an update to a major DB engine version. In RDS, a minor engine version is either a Release Update (RU) or Spatial Patch Bundle (SPB). For example, if your DB instance runs major version Oracle Database 19c and minor version

19.0.0.0.ru-2025-10.rur-2025-10.r1, you can upgrade your DB engine to minor version 19.0.0.0.ru-2026-01.rur-2026-01.r1. RDS for Oracle doesn't support minor version downgrades.

You can upgrade your DB engine to a minor version manually or automatically. To learn how to upgrade manually, see Manually upgrading the engine version. To learn how to configure automatic upgrades, see Automatically upgrading the minor engine version. Whether you upgrade manually or automatically, a minor version upgrade entails downtime. Consider this downtime when you plan your upgrades.

Amazon RDS also supports upgrade rollout policy to manage automatic minor version upgrades across multiple database resources and AWS accounts. For more information, see Using AWS Organizations upgrade rollout policy for automatic minor version upgrades.

> **Important**
>
> Make sure that you thoroughly test any upgrade to verify that your applications work correctly before applying the upgrade to your production databases. For more information, see Testing an Oracle DB upgrade.

Topics

- Release Updates (RUs) and Spatial Patch Bundles (SPBs)
- Turning on automatic minor version upgrades for Oracle
- Using AWS Organizations upgrade rollout policy for automatic minor version upgrades
- Notification of automatic minor version upgrades in RDS for Oracle
- How Amazon RDS schedules automatic minor version upgrades
- Managing an automatic minor version upgrade in RDS for Oracle

Release Updates (RUs) and Spatial Patch Bundles (SPBs)

In RDS, a release update (RU) is a quarterly minor engine version that includes security fixes, bug fixes, and new features for Oracle Database. A Spatial Patch Bundle (SPB) is an RU engine version that includes patches designed for the Oracle Spatial option. For example, the SPB named 19.0.0.0.ru-2025-01.spb-1.r1 includes all patches in the corresponding RU 19.0.0.0.ru-2025-01.rur-2025-01.r1 plus patches specific to Spatial. SPBs are supported only for Oracle Database 19c.

When your instance is configured for automatic minor version upgrades, RUs and SPBs are on separate upgrade paths. Typically, an SPB is released 2–3 weeks after its corresponding RU. The following table shows sample minor versions for Oracle Database 19c.

Standard RU upgrade path	SPB upgrade path
19.0.0.0.ru-2025-01.rur-2025-01.r1	19.0.0.0.ru-2025-01.spb-1.r1
19.0.0.0.ru-2025-04.rur-2025-04.r1	19.0.0.0.ru-2025-04.spb-1.r1
19.0.0.0.ru-2025-07.rur-2025-07.r1	19.0.0.0.ru-2025-07.spb-1.r1
19.0.0.0.ru-2025-10.rur-2025-10.r1	19.0.0.0.ru-2025-10.spb-1.r1

If your DB instance is configured for automatic upgrades, your instance is on the upgrade path corresponding to your current version. For example, if your DB instance runs version 19.0.0.0.ru-2025-01.rur-2025-01.r1, then when 19.0.0.0.ru-2025-04.rur-2025-04.r1 is released, your instance automatically upgrades to this RU. Similarly, if your DB instance runs 19.0.0.0.ru-2025-01.spb-1.r1, then when 19.0.0.0.ru-2025-04.spb-1.r1 is released, your instance automatically upgrades to this SPB. An instance running 19.0.0.0.ru-2025-01.rur-2025-01.r1, which is an RU, won't automatically upgrade to 19.0.0.0.ru-2025-04.spb-1.r1, which is an SPB on a separate upgrade path.

You can upgrade your DB instance to SPBs even if your instance doesn't use Spatial, but the Spatial patches apply only to Oracle Spatial. You can upgrade manually from an RU to an SPB at the same engine version or higher. For example, you can upgrade your instance from 19.0.0.0.ru-2025-01.rur-2025-01.r1 to either of the following engine versions:

- 19.0.0.0.ru-2025-01.spb-1.r1
- 19.0.0.0.ru-2025-04.spb-1.r1

You can upgrade your instance from an SPB to an RU only if the RU is a higher engine version. For example, you can upgrade from SPB version 19.0.0.0.ru-2025-04.spb-1.r1 to a higher RU version 19.0.0.0.ru-2025-07.rur-2025-07.r1 but not to the same RU version 19.0.0.0.ru-2025-04.rur-2025-04.r1.

If your DB instance is configured for automatic minor version upgrades, and you manually upgrade from an RU to an SPB or from an SPB to an RU, your automatic upgrade path changes. Suppose

that you manually upgrade from RU version 19.0.0.0.ru-2025-01.rur-2025-01.r1 to SPB version 19.0.0.0.ru-2025-01.spb-1.r1. Your next automatic minor version upgrade will be to SPB version 19.0.0.0.ru-2025-04.spb-1.r1.

Because SPBs function as RUs, the RDS APIs for upgrading your instance to RUs and SPBs are identical. The following commands demonstrate upgrading to an RU and to an SPB.

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --engine-version 19.0.0.0.ru-2025-01.rur-2025-01.r1

aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --engine-version 19.0.0.0.ru-2025-01.spb-1.r1
```

For more information about the Oracle Spatial option, see How Spatial Patch Bundles (SPBs) work. For supported RUs and SPBs for Oracle Database 19c, see Amazon RDS for Oracle Database 19c (19.0.0.0).

Turning on automatic minor version upgrades for Oracle

In an automatic minor version upgrade, RDS applies the latest available minor version to your Oracle database without manual intervention. An Amazon RDS for Oracle DB instance schedules your upgrade during the next maintenance window in the following circumstances:

- Your DB instance has the **Auto minor version upgrade** option turned on.
- Your DB instance isn't already running the latest minor DB engine version.

To learn how to turn on automatic upgrades, see Automatically upgrading the minor engine version.

Using AWS Organizations upgrade rollout policy for automatic minor version upgrades

Amazon RDS for Oracle supports AWS Organizations upgrade rollout policy to manage automatic minor version upgrades across multiple database resources and AWS accounts. This policy eliminates the operational overhead of coordinating automatic minor version upgrades either manually or through custom tools, while ensuring upgrades are first applied in non-production environments before being rolled out to production. When a new minor engine version becomes

available, Amazon RDS upgrades your DB instances based on their configured upgrade rollout order:

Upgrade Rollout Order	Typical Use Case	When Upgrade Begins
First	Development and Test Environments	Earliest - ideal for validating new versions
Second	Staging and Non-critical Production Environments	After "First" phase completes
Last	Critical Production Environments	After "Second" phase completes

> **Important**
>
> If you do not configure an upgrade rollout order for your DB instance, it defaults to second.

For detailed information about phase timing and duration, see How Amazon RDS schedules automatic minor version upgrades. For information about configuring upgrade rollout policies in AWS Organizations, see Using AWS Organizations upgrade rollout policy for automatic minor version upgrades.

Notification of automatic minor version upgrades in RDS for Oracle

If automatic minor version upgrade is enabled on your DB instance, RDS for Oracle creates pending maintenance actions to notify you before applying upgrades. You can view these pending maintenance actions on the **Maintenance & backups** tab of your database details page in the Amazon RDS console.

When a new minor version becomes available, RDS for Oracle publishes an early notification (pending maintenance action). The early notification has the following format:

```
An automatic minor version upgrade to engine-version will be applied during your
 maintenance window on apply-date based on the upgrade rollout order rollout-order. You
```

```
can change the upgrade rollout order or apply this upgrade manually at any time before
the scheduled date through the AWS console or AWS CLI.
```

`apply-date` in the early notification is the date when Amazon RDS will upgrade your DB instance. `rollout-order` is your upgrade rollout order (first, second, or last). If you have not configured an upgrade rollout policy, this value is second by default. For more information, see Using AWS Organizations upgrade rollout policy for automatic minor version upgrades.

When the upgrade rollout phase begins, the pending maintenance action message changes to the following format:

```
Automatic minor version upgrade to engine-version
```

This message indicates that the upgrade has been scheduled and will be applied during your maintenance window on the scheduled apply date. You can check the scheduled apply date on the **Maintenance & backups** tab of your database details page in the Amazon RDS console or in the `CurrentApplyDate` field of the `describe-pending-maintenance-actions` API response.

The following example shows you can get the details about pending maintenance actions by using the `describe-pending-maintenance-actions` command in the AWS CLI:

```
aws rds describe-pending-maintenance-actions

    "PendingMaintenanceActions": [
        {
            "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:orclinst1",
            "PendingMaintenanceActionDetails": [
                {
                    "Action": "db-upgrade",
                    "Description": "Automatic minor version upgrade to
 21.0.0.0.ru-2024-07.rur-2024-07.r1",
                    "CurrentApplyDate": "2024-12-02T08:10:00Z"
                }
            ]
        }, ...
```

For more information about describe-pending-maintenance-actions, see the *AWS CLI Command Reference*.

How Amazon RDS schedules automatic minor version upgrades

When you use AWS Organizations upgrade rollout policy, Amazon RDS upgrades DB instances in phases based on their configured rollout order. This section describes the timing and duration of each phase.

Phase 0: Early Notification

When RDS for Oracle releases a new minor version (typically 3 to 4 weeks after Oracle's quarterly RU release), all DB instances with auto minor version upgrade enabled receive an early notification. This notification appears on the **Maintenance & backups** tab of the database details page in the Amazon RDS console and in the `describe-pending-maintenance-actions` API response. The early notification phase lasts 2 weeks. During this phase, no automatic upgrades occur.

Phase 1: Upgrade Rollout Order First

At the end of the early notification phase, RDS for Oracle begins upgrading DB instances with the upgrade rollout order first. This phase lasts 2 to 3 weeks for the January, April, July quarterly minor versions, and 7 to 8 weeks for the October quarterly minor version. The extended period for the October minor version provides sufficient time to test the new minor version during the year-end holiday season. New DB instances created during this phase with the upgrade rollout order first will be upgraded automatically.

Phase 2: Upgrade Rollout Order Second

At the end of phase 1, RDS for Oracle begins upgrading DB instances with upgrade rollout order second. This phase lasts 2 weeks for all quarterly minor versions. New DB instances created with upgrade rollout order first or second during this phase will be upgraded automatically.

Phase 3: Upgrade Rollout Order Last

At the end of Phase 2, RDS for Oracle begins upgrading DB instances with upgrade rollout order last. This phase lasts until the next quarterly minor version release. New DB instances created with upgrade rollout order first, second, or last during this phase will be upgraded automatically.

Phase	When it starts	Duration	Pending maintenance action message
Phase 0: Early Notification	When RDS for Oracle releases a new minor version	2 weeks	An automatic minor version upgrade to *engine-version* will be applied during your maintenance window on *apply-date* based on the upgrade rollout order *rollout-order*. You can change the upgrade rollout order or apply this upgrade manually at any time before the scheduled date through the AWS console or AWS CLI.
Phase 1: Upgrade Rollout Order First	End of early notification phase	2 to 4 weeks for January/April/July minor versions, 7 to	Automatic minor version upgrade to

Phase	When it starts	Duration	Pending maintenance action message
		9 weeks for October minor version	*engine-version*
Phase 2: Upgrade Rollout Order Second	End of Phase 1	2 weeks	Automatic minor version upgrade to *engine-version*
Phase 3: Upgrade Rollout Order Last	End of Phase 2	Until the next quarterly minor version release	Automatic minor version upgrade to *engine-version*

Managing an automatic minor version upgrade in RDS for Oracle

When auto minor version upgrade is enabled on your DB instance, Amazon RDS automatically upgrades your DB instance to the latest minor version during your maintenance window. However, you can choose to apply the upgrade manually before the scheduled date using the AWS CLI or on the **Maintenance & backups** tab of the database details page.

To upgrade your DB instance immediately instead of waiting for the scheduled maintenance window:

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-east-1:123456789012:db:orclinst1 \
    --apply-action db-upgrade \
    --opt-in-type immediate
```

To apply the upgrade during your next maintenance window instead of the scheduled apply date:

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-east-1:123456789012:db:orclinst1 \
```

```
    --apply-action db-upgrade \
    --opt-in-type next-maintenance
```

To opt out of an automatic minor version upgrade, modify your DB instance and turn off the automatic minor version upgrade option. This unschedules any pending automatic upgrade.

To learn more about how to turn off automatic minor version upgrade, see Automatically upgrading the minor engine version. If you need assistance with turning off automatic minor version upgrade, please reach out to the AWS Support.

Sometimes a new minor version becomes available before RDS applies a previous minor version. For example, your instance is running on `21.0.0.0.ru-2025-07.rur-2025-07.r1` when both `21.0.0.0.ru-2025-10.rur-2025-10.r1` and `21.0.0.0.ru-2026-01.rur-2026-01.r1` are available as upgrade targets. In this situation, to avoid unnecessary downtime for your DB instances, RDS schedules the automatic minor version upgrade to the most recent version, skipping the upgrade to the previous version. In this example, RDS upgrades your instance from `21.0.0.0.ru-2025-07.rur-2025-07.r1` directly to `21.0.0.0.ru-2026-01.rur-2026-01.r1`.

Considerations for Oracle database upgrades

Before you upgrade your Oracle instance, review the following information.

Topics

- Oracle Multitenant considerations
- Option group considerations
- Parameter group considerations
- Time zone considerations
- Spatial Patch Bundle (SPB) considerations

Oracle Multitenant considerations

The following table describes the Oracle Database architectures supported in different releases.

Oracle Database release	RDS support status	Architecture
Oracle Database 21c	Supported	CDB only

Oracle Database release	RDS support status	Architecture
Oracle Database 19c	Supported	CDB or non-CDB

The following table describes supported and unsupported upgrade paths.

Upgrade path	Supported?
CDB to CDB	Yes
Non-CDB to CDB	No, but you can convert a non-CDB to a CDB and then upgrade it
CDB to non-CDB	No

For more information about Oracle Multitenant in RDS for Oracle, see Single-tenant configuration of the CDB architecture.

Option group considerations

If your DB instance uses a custom option group, sometimes Amazon RDS can't automatically assign a new option group. For example, this situation occurs when you upgrade to a new major version. In such cases, specify a new option group when you upgrade. We recommend that you create a new option group, and add the same options to it as in your existing custom option group.

For more information, see Creating an option group or Copying an option group.

If your DB instance uses a custom option group that contains the `APEX` and `APEX-DEV` options, you can sometimes reduce the upgrade time. To do this, upgrade your version of Oracle APEX at the same time as your DB instance. For more information, see Upgrading the Oracle APEX version.

Parameter group considerations

If your DB instance uses a custom parameter group, sometimes Amazon RDS can't automatically assign your DB instance a new parameter group. For example, this situation occurs when you upgrade to a new major version. In such cases, make sure to specify a new parameter group when you upgrade. We recommend that you create a new parameter group, and configure the parameters as in your existing custom parameter group.

For more information, see Creating a DB parameter group in Amazon RDS or Copying a DB parameter group in Amazon RDS.

Time zone considerations

You can use the time zone option to change the *system time zone* used by your Oracle DB instance. For example, you might change the time zone of a DB instance to be compatible with an on-premises environment, or a legacy application. The time zone option changes the time zone at the host level. Amazon RDS for Oracle updates the system time zone automatically throughout the year. For more information about the system time zone, see Oracle time zone.

When you create an Oracle DB instance, the database automatically sets the *database time zone*. The database time zone is also known as the Daylight Saving Time (DST) time zone. The database time zone is distinct from the system time zone.

Between Oracle Database releases, patch sets or individual patches may include new DST versions. These patches reflect the changes in transition rules for various time zone regions. For example, a government might change when DST takes effect. Changes to DST rules may affect existing data of the `TIMESTAMP WITH TIME ZONE` data type.

If you upgrade an RDS for Oracle DB instance, Amazon RDS doesn't upgrade the database time zone file automatically. To upgrade the time zone file automatically, you can include the `TIMEZONE_FILE_AUTOUPGRADE` option in the option group associated with your DB instance during or after the engine version upgrade. For more information, see Oracle time zone file autoupgrade.

Alternatively, to upgrade the database time zone file manually, create a new Oracle DB instance that has the desired DST patch. However, we recommend that you upgrade the database time zone file using the `TIMEZONE_FILE_AUTOUPGRADE` option.

After upgrading the time zone file, migrate the data from your current instance to the new instance. You can migrate data using several techniques, including the following:

- AWS Database Migration Service
- Oracle GoldenGate
- Oracle Data Pump
- Original Export/Import (desupported for general use)

> **Note**
>
> When you migrate data using Oracle Data Pump, the utility raises the error ORA-39405 when the target time zone version is lower than the source time zone version.

For more information, see TIMESTAMP WITH TIMEZONE restrictions in the Oracle documentation.

Spatial Patch Bundle (SPB) considerations

In RDS for Oracle, release update (RU) is a minor engine version that includes security fixes, bug fixes, and new features for Oracle Database. A Spatial Patch Bundle (SPB) is minor engine version that also includes patches designed for the Oracle Spatial option. For example, 19.0.0.0.ru-2025-01.spb-1.r1 is a minor engine version that contains the RU patches in engine version 19.0.0.0.ru-2025-01.rur-2025-01.r1 plus Spatial patches.

When you upgrade your database to SPBs, consider the following:

- SPBs are supported only for Oracle Database 19c.
- Typically, an SPB is released 2–3 weeks after its corresponding quarterly RU.
- You can upgrade your DB instance to an SPB even if the instance doesn't use the Oracle Spatial option, but the Spatial patches in the engine version apply only to Oracle Spatial. You can create a new instance on an SPB and install the Oracle Spatial option later.
- If you enable automatic minor version upgrade for your DB instance, your upgrade path depends on whether your instance currently uses an SPB or RU. If your instance uses an SPB, RDS automatically upgrades your instance to the latest SPB. If your instance uses an RU, RDS automatically upgrades your instance to the latest RU.
- You can manually upgrade your DB instance from an RU to an SPB only if the SPB is the same engine version or higher as your current RU.
- You can manually upgrade your DB instance from an SPB to an RU only if the RU is a higher version.

Testing an Oracle DB upgrade

Before you upgrade your DB instance to a major version, thoroughly test your database and all applications that access the database for compatibility with the new version. We recommend that you use the following procedure.

To test a major version upgrade

1. Review the Oracle upgrade documentation for the new version of the database engine to see if there are compatibility issues that might affect your database or applications. For more information, see Database Upgrade Guide in the Oracle documentation.
2. If your DB instance uses a custom option group, create a new option group compatible with the new version you are upgrading to. For more information, see Option group considerations.
3. If your DB instance uses a custom parameter group, create a new parameter group compatible with the new version you are upgrading to. For more information, see Parameter group considerations.
4. Create a DB snapshot of the DB instance to be upgraded. For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.
5. Restore the DB snapshot to create a new test DB instance. For more information, see Restoring to a DB instance.
6. Modify this new test DB instance to upgrade it to the new version, by using one of the following methods:
   - Console
   - AWS CLI
   - RDS API
7. Perform testing:
   - Run as many of your quality assurance tests against the upgraded DB instance as needed to ensure that your database and application work correctly with the new version.
   - Implement any new tests needed to evaluate the impact of any compatibility issues that you identified in step 1.
   - Test all stored procedures, functions, and triggers.
   - Direct test versions of your applications to the upgraded DB instance. Verify that the applications work correctly with the new version.
   - Evaluate the storage used by the upgraded instance to determine if the upgrade requires additional storage. You might need to choose a larger instance class to support the new version in production. For more information, see DB instance classes.
8. If all tests pass, upgrade your production DB instance. We recommend that you confirm that the DB instance working correctly before allowing write operations to the DB instance.

Upgrading the version of an RDS for Oracle DB instance

To manually upgrade the DB engine version of an RDS for Oracle DB instance,use the AWS Management Console, the AWS CLI, or the RDS API. For general information about database upgrades in RDS, see Upgrading the version of an RDS for Oracle DB instance. To get valid upgrade targets, use the AWS CLI describe-db-engine-versions command.

Console

To upgrade the engine version of an RDS for Oracle DB instance by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to upgrade.
3. Choose **Modify**.
4. For **DB engine version**, choose a higher database version.
5. Choose **Continue** and check the summary of modifications. Make sure that you understand the implications of a database version upgrade. You can't convert an upgraded DB instance back to the previous version. Make sure you have tested both your database and your application with the new version before continuing.
6. Decide when to schedule your DB instance upgrade. To apply the changes immediately, choose **Apply immediately**. Choosing this option can cause an outage in some cases. For more information, see Using the schedule modifications setting.
7. On the confirmation page, review your changes. If they are correct, choose **Modify DB instance** to save your changes.

   Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

AWS CLI

To upgrade the engine version of an RDS for Oracle DB instance, you can use the CLI modify-db-instance command. Specify the following parameters:

- `--db-instance-identifier` – the name of the RDS for Oracle DB instance.
- `--engine-version` – the version number of the database engine to upgrade to.

For information about valid engine versions, use the AWS CLI describe-db-engine-versions command.

- `--allow-major-version-upgrade` – to upgrade the DB engine version.
- `--no-apply-immediately` – to apply changes during the next maintenance window. To apply changes immediately, use `--apply-immediately`.

Example

The following example upgrades a CDB instance named `myorainst` from its current version of `19.0.0.0.ru-2024-01.rur-2024-01.r1` to version `21.0.0.0.ru-2024-04.rur-2024-04.r1`.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier myorainst \
    --engine-version 21.0.0.0.ru-2024-04.rur-2024-04.r1 \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier myorainst ^
    --engine-version 21.0.0.0.ru-2024-04.rur-2024-04.r1 ^
    --allow-major-version-upgrade ^
    --no-apply-immediately
```

RDS API

To upgrade an RDS for Oracle DB instance, use the ModifyDBInstance action. Specify the following parameters:

- `DBInstanceIdentifier` – the name of the DB instance, for example *`myorainst`*.
- `EngineVersion` – the version number of the database engine to upgrade to. For information about valid engine versions, use the DescribeDBEngineVersions operation.
- `AllowMajorVersionUpgrade` – whether to allow a major version upgrade. To do so, set the value to `true`.

- `ApplyImmediately` – whether to apply changes immediately or during the next maintenance window. To apply changes immediately, set the value to `true`. To apply changes during the next maintenance window, set the value to `false`.

Upgrading an Oracle DB snapshot

Upgrading your Oracle DB snapshots in Amazon RDS ensures that your database remains secure, compatible, and fully supported. As older Oracle versions reach the end of patch support, you can upgrade any manual DB snapshots tied to these versions to avoid potential vulnerabilities or service limitations. For more information, see Oracle engine version management.

Amazon RDS supports upgrading snapshots in all AWS Regions.

Console

To upgrade an Oracle DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**, and then select the DB snapshot that you want to upgrade.
3. For **Actions**, choose **Upgrade snapshot**. The **Upgrade snapshot** page appears.
4. Choose the **New engine version** to upgrade the snapshot to.
5. (Optional) For **Option group**, choose the option group for the upgraded DB snapshot. The same option group considerations apply when upgrading a DB snapshot as when upgrading a DB instance. For more information, see Option group considerations.
6. Choose **Save changes** to save your changes.

   During the upgrade process, all snapshot actions are disabled for this DB snapshot. Also, the DB snapshot status changes from **available** to **upgrading**, and then changes to **active** upon completion. If the DB snapshot can't be upgraded because of snapshot corruption issues, the status changes to **unavailable**. You can't recover the snapshot from this state.

   > **Note**
   >
   > If the DB snapshot upgrade fails, the snapshot is rolled back to the original state with the original version.

AWS CLI

To upgrade an Oracle DB snapshot by using the AWS CLI, call the modify-db-snapshot command with the following parameters:

- `--db-snapshot-identifier` – The name of the DB snapshot.
- `--engine-version` – The version to upgrade the snapshot to.

You might also need to include the following parameter. The same option group considerations apply when upgrading a DB snapshot as when upgrading a DB instance. For more information, see Option group considerations.

- `--option-group-name` – The option group for the upgraded DB snapshot.

Example

The following example upgrades a DB snapshot.

For Linux, macOS, or Unix:

```
aws rds modify-db-snapshot \
    --db-snapshot-identifier mydbsnapshot \
    --engine-version 19.0.0.0.ru-2020-10.rur-2020-10.r1 \
    --option-group-name default:oracle-se2-19
```

For Windows:

```
aws rds modify-db-snapshot ^
    --db-snapshot-identifier mydbsnapshot ^
    --engine-version 19.0.0.0.ru-2020-10.rur-2020-10.r1 ^
    --option-group-name default:oracle-se2-19
```

RDS API

To upgrade an Oracle DB snapshot by using the Amazon RDS API, call the ModifyDBSnapshot operation with the following parameters:

- `DBSnapshotIdentifier` – The name of the DB snapshot.
- `EngineVersion` – The version to upgrade the snapshot to.

You might also need to include the `OptionGroupName` parameter. The same option group considerations apply when upgrading a DB snapshot as when upgrading a DB instance. For more information, see Option group considerations.

Using third-party software with your RDS for Oracle DB instance

You can host an RDS for Oracle DB instance that supports tools and third-party software.

Topics

- Using Oracle GoldenGate with Amazon RDS for Oracle
- Using the Oracle Repository Creation Utility on RDS for Oracle
- Configuring Oracle Connection Manager on an Amazon EC2 instance
- Installing a Siebel database on Oracle on Amazon RDS

Using Oracle GoldenGate with Amazon RDS for Oracle

Oracle GoldenGate collects, replicates, and manages transactional data between databases. It is a log-based change data capture (CDC) and replication software package used with databases for online transaction processing (OLTP) systems. Oracle GoldenGate creates trail files that contain the most recent changed data from the source database. It then pushes these files to the server, where a process converts the trail file into standard SQL to be applied to the target database.

Oracle GoldenGate with RDS for Oracle supports the following features:

- Active-Active database replication
- Disaster recovery
- Data protection
- In-Region and cross-Region replication
- Zero-downtime migration and upgrades
- Data replication between an RDS for Oracle DB instance and a non-Oracle database

> **Note**
>
> For a list of supported databases, see Oracle Fusion Middleware Supported System Configurations in the Oracle documentation.

You can use Oracle GoldenGate with RDS for Oracle to upgrade to major versions of Oracle Database. For example, you can use Oracle GoldenGate to upgrade from an Oracle Database 11g on-premises database to Oracle Database 19c on an Amazon RDS DB instance.

Topics

- Supported versions and licensing options for Oracle GoldenGate
- Requirements and limitations for Oracle GoldenGate
- Oracle GoldenGate architecture
- Setting up Oracle GoldenGate
- Working with the EXTRACT and REPLICAT utilities of Oracle GoldenGate
- Monitoring Oracle GoldenGate
- Troubleshooting Oracle GoldenGate

Supported versions and licensing options for Oracle GoldenGate

You can use Standard Edition 2 (SE2) or Enterprise Edition (EE) of RDS for Oracle with Oracle GoldenGate version 12c and higher. You can use the following Oracle GoldenGate features:

- Oracle GoldenGate Remote Capture (extract) is supported.
- Capture (extract) is supported on RDS for Oracle DB instances that use the traditional non-CDB database architecture. Oracle GoldenGate Remote PDB capture is supported on CDBs running Oracle Database 21c or Oracle Database 19c version 19.0.0.0.ru-2024-04.rur-2024-04.r1 or higher.
- Oracle GoldenGate Remote Delivery (replicat) is supported on RDS for Oracle DB instances that use either the non-CDB or CDB architectures. Remote Delivery supports Integrated Replicat, Parallel Replicat, Coordinated Replicat, and classic Replicat.
- RDS for Oracle supports the Classic and Microservices architectures of Oracle GoldenGate.
- Oracle GoldenGate DDL and Sequence value replication is supported when using Integrated capture mode.

You are responsible for managing Oracle GoldenGate licensing (BYOL) for use with Amazon RDS in all AWS Regions. For more information, see RDS for Oracle licensing options.

Requirements and limitations for Oracle GoldenGate

When you're working with Oracle GoldenGate and RDS for Oracle, consider the following requirements and limitations:

- You're responsible for setting up and managing Oracle GoldenGate for use with RDS for Oracle.
- You're responsible for setting up an Oracle GoldenGate version that is certified with the source and the target databases. For more information, see Oracle Fusion Middleware Supported System Configurations in the Oracle documentation.
- You can use Oracle GoldenGate on many different AWS environments for many different use cases. If you have a support-related issue relating to Oracle GoldenGate, contact Oracle Support Services.
- You can use Oracle GoldenGate on RDS for Oracle DB instances that use Oracle Transparent Data Encryption (TDE). To maintain the integrity of replicated data, configure encryption on the Oracle GoldenGate hub using Amazon EBS encrypted volumes or trail file encryption. Also configure encryption for data sent between the Oracle GoldenGate hub and the source and

target database instances. RDS for Oracle DB instances support encryption with Oracle Secure Sockets Layer or Oracle native network encryption.

Oracle GoldenGate architecture

The Oracle GoldenGate architecture for use with Amazon RDS consists of the following decoupled modules:

Source database

Your source database can be either an on-premises Oracle database, an Oracle database on an Amazon EC2 instance, or an Oracle database on an Amazon RDS DB instance.

Oracle GoldenGate hub

An Oracle GoldenGate hub moves transaction information from the source database to the target database. Your hub can be either of the following:

- An Amazon EC2 instance with Oracle Database and Oracle GoldenGate installed
- An on-premises Oracle installation

You can have more than one Amazon EC2 hub. We recommend that you use two hubs if you use Oracle GoldenGate for cross-Region replication.

Target database

Your target database can be on either an Amazon RDS DB instance, an Amazon EC2 instance, or an on-premises location.

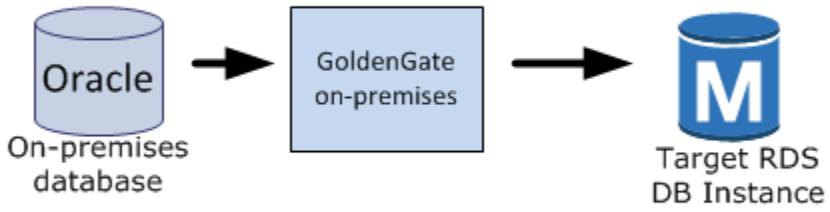
The following sections describe common scenarios for Oracle GoldenGate on Amazon RDS.

Topics

- On-premises source database and Oracle GoldenGate hub
- On-premises source database and Amazon EC2 hub
- Amazon RDS source database and Amazon EC2 hub
- Amazon EC2 source database and Amazon EC2 hub
- Amazon EC2 hubs in different AWS Regions

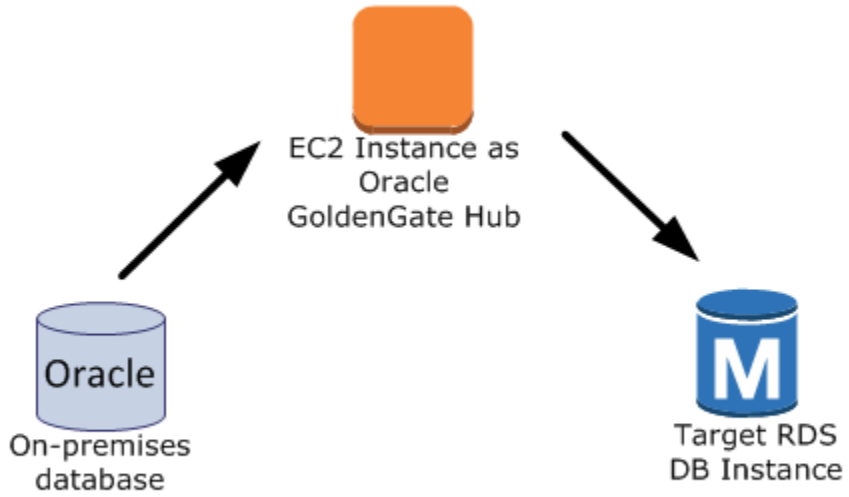
On-premises source database and Oracle GoldenGate hub

In this scenario, an on-premises Oracle source database and on-premises Oracle GoldenGate hub provides data to a target Amazon RDS DB instance.



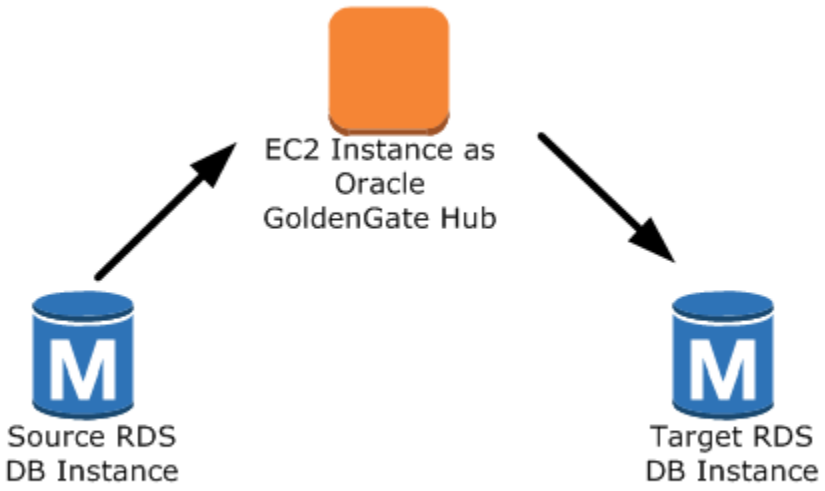
On-premises source database and Amazon EC2 hub

In this scenario, an on-premises Oracle database acts as the source database. It's connected to an Amazon EC2 instance hub. This hub provides data to a target RDS for Oracle DB instance.



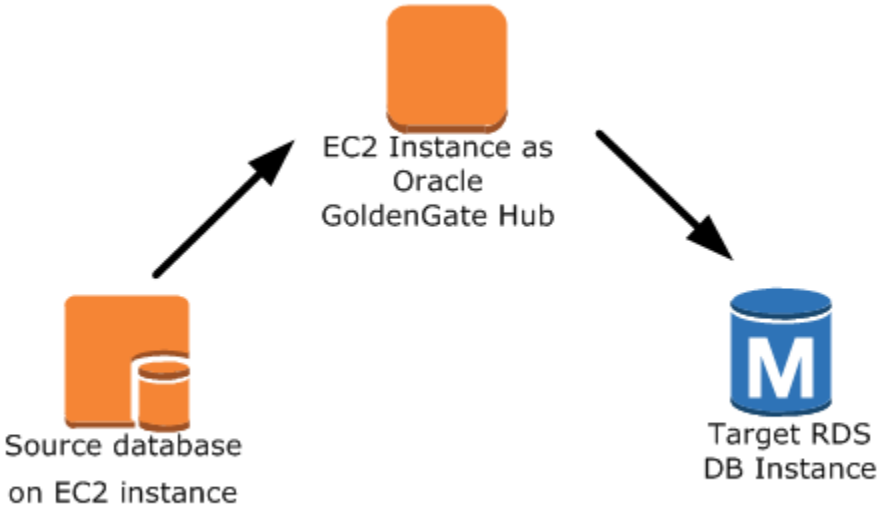
Amazon RDS source database and Amazon EC2 hub

In this scenario, an RDS for Oracle DB instance acts as the source database. It's connected to an Amazon EC2 instance hub. This hub provides data to a target RDS for Oracle DB instance.



Amazon EC2 source database and Amazon EC2 hub

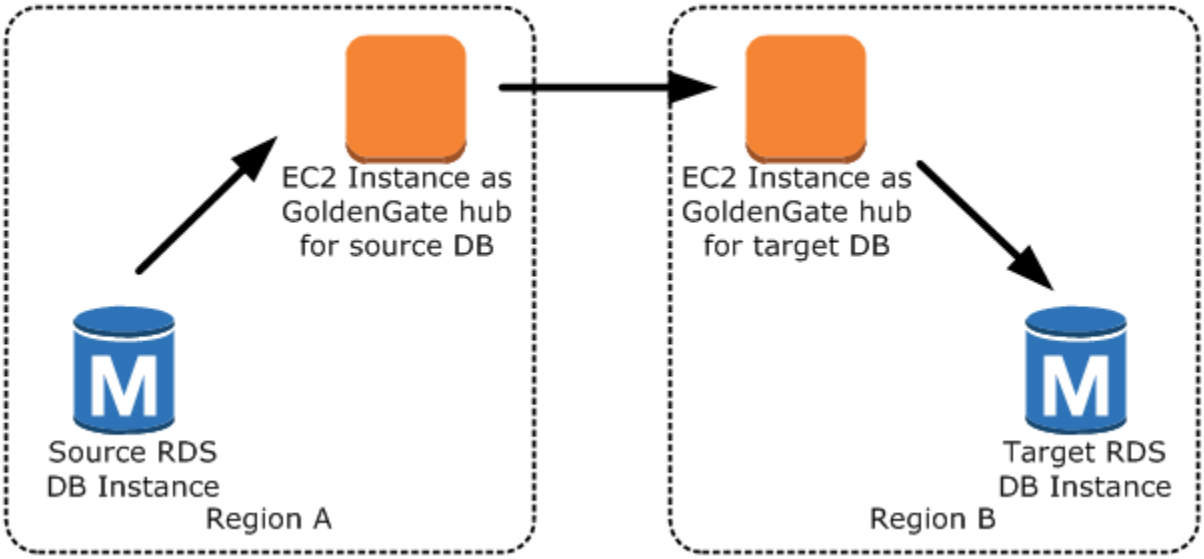
In this scenario, an Oracle database on an Amazon EC2 instance acts as the source database. It's connected to an Amazon EC2 instance hub. This hub provides data to a target RDS for Oracle DB instance.



Amazon EC2 hubs in different AWS Regions

In this scenario, an Oracle database on an Amazon RDS DB instance is connected to an Amazon EC2 instance hub in the same AWS Region. The hub is connected to an Amazon EC2 instance hub in a

different AWS Region. This second hub provides data to the target RDS for Oracle DB instance in the same AWS Region as the second Amazon EC2 instance hub.



> **Note**
>
> Any issues that affect running Oracle GoldenGate on an on-premises environment also affect running Oracle GoldenGate on AWS. We strongly recommend that you monitor the Oracle GoldenGate hub to ensure that `EXTRACT` and `REPLICAT` are resumed if a failover occurs. Because the Oracle GoldenGate hub is run on an Amazon EC2 instance, Amazon RDS does not manage the Oracle GoldenGate hub and cannot ensure that it is running.

Setting up Oracle GoldenGate

To set up Oracle GoldenGate using Amazon RDS, configure the hub on an Amazon EC2 instance, and then configure the source and target databases. The following sections give an example of how to set up Oracle GoldenGate for use with Amazon RDS for Oracle.

Topics

- Setting up an Oracle GoldenGate hub on Amazon EC2
- Setting up a source database for use with Oracle GoldenGate on Amazon RDS
- Setting up a target database for use with Oracle GoldenGate on Amazon RDS

Setting up an Oracle GoldenGate hub on Amazon EC2

To create an Oracle GoldenGate hub on an Amazon EC2 instance, you first create an Amazon EC2 instance with a full client installation of Oracle RDBMS. The Amazon EC2 instance must also have Oracle GoldenGate software installed. The Oracle GoldenGate software versions depend on the source and target database versions. For more information about installing Oracle GoldenGate, see the Oracle GoldenGate documentation.

The Amazon EC2 instance that serves as the Oracle GoldenGate hub stores and processes the transaction information from the source database into trail files. To support this process, make sure that you meet the following requirements:

- You have allocated enough storage for the trail files.
- The Amazon EC2 instance has enough processing power to manage the amount of data.
- The EC2 instance has enough memory to store the transaction information before it's written to the trail file.

To set up an Oracle GoldenGate classic architecture hub on an Amazon EC2 instance

1. Create subdirectories in the Oracle GoldenGate directory.

   In the Amazon EC2 command line shell, start `ggsci`, the Oracle GoldenGate command interpreter. The `CREATE SUBDIRS` command creates subdirectories under the `/gg` directory for parameter, report, and checkpoint files.

   ```
   prompt$ cd /gg
   prompt$ ./ggsci

   GGSCI> CREATE SUBDIRS
   ```

2. Configure the `mgr.prm` file.

   The following example adds lines to the `$GGHOME/dirprm/mgr.prm` file.

   ```
   PORT 8199
   PurgeOldExtracts ./dirdat/*, UseCheckpoints, MINKEEPDAYS 5
   ```

3. Start the manager.

   The following example starts `ggsci` and runs the `start mgr` command.

```
GGSCI> start mgr
```

The Oracle GoldenGate hub is now ready for use.

Setting up a source database for use with Oracle GoldenGate on Amazon RDS

Complete the following tasks to set up a source database for use with Oracle GoldenGate.

Setup steps

- Step 1: Turn on supplemental logging on the source database
- Step 2: Set the ENABLE_GOLDENGATE_REPLICATION initialization parameter to true
- Step 3: Set the log retention period on the source database
- Step 4: Create an Oracle GoldenGate user account on the source database
- Step 5: Grant user account privileges on the source database
- Step 6: Add a TNS alias for the source database

Step 1: Turn on supplemental logging on the source database

To turn on the minimum database-level supplemental logging, run the following PL/SQL procedure:

```
EXEC rdsadmin.rdsadmin_util.alter_supplemental_logging(p_action => 'ADD')
```

Step 2: Set the ENABLE_GOLDENGATE_REPLICATION initialization parameter to true

When you set the `ENABLE_GOLDENGATE_REPLICATION` initialization parameter to `true`, it allows database services to support logical replication. If your source database is on an Amazon RDS DB instance, make sure that you have a parameter group assigned to the DB instance with the `ENABLE_GOLDENGATE_REPLICATION` initialization parameter set to `true`. For more information about the `ENABLE_GOLDENGATE_REPLICATION` initialization parameter, see the Oracle Database documentation.

Step 3: Set the log retention period on the source database

Make sure that you configure the source database to retain archived redo logs. Consider the following guidelines:

- Specify the duration for log retention in hours. The minimum value is one hour.
- Set the duration to exceed any potential downtime of the source DB instance, any potential period of communication, and any potential period of networking issues for the source instance. Such a duration lets Oracle GoldenGate recover logs from the source instance as needed.
- Ensure that you have sufficient storage on your instance for the files.

For example, set the retention period for archived redo logs to 24 hours.

```
EXEC rdsadmin.rdsadmin_util.set_configuration('archivelog retention hours',24)
```

If you don't have log retention enabled, or if the retention value is too small, you receive an error message similar to the following.

```
2022-03-06 06:17:27  ERROR   OGG-00446  error 2 (No such file or directory)
opening redo log /rdsdbdata/db/GGTEST3_A/onlinelog/o1_mf_2_9k4bp1n6_.log for sequence
 1306
Not able to establish initial position for begin time 2022-03-06 06:16:55.
```

Because your DB instance retains your archived redo logs, make sure that you have sufficient space for the files. To see how much space you have used in the last *num_hours* hours, run the following query, replacing *num_hours* with the number of hours.

```
SELECT SUM(BLOCKS * BLOCK_SIZE) BYTES FROM V$ARCHIVED_LOG
   WHERE NEXT_TIME>=SYSDATE-num_hours/24 AND DEST_ID=1;
```

Step 4: Create an Oracle GoldenGate user account on the source database

Oracle GoldenGate runs as a database user and requires the appropriate database privileges to access the redo and archived redo logs for the source database. To provide these, create a user account on the source database. For more information about the permissions for an Oracle GoldenGate user account, see the Oracle documentation.

The following statements create a user account named oggadm1.

```
CREATE TABLESPACE administrator;
CREATE USER oggadm1  IDENTIFIED BY "password"
   DEFAULT TABLESPACE ADMINISTRATOR TEMPORARY TABLESPACE TEMP;
```

```
ALTER USER oggadm1 QUOTA UNLIMITED ON administrator;
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Step 5: Grant user account privileges on the source database

In this task, you grant necessary account privileges for database users on your source database.

To grant account privileges on the source database

1. Grant the necessary privileges to the Oracle GoldenGate user account using the SQL command `grant` and the `rdsadmin.rdsadmin_util` procedure `grant_sys_object`. The following statements grant privileges to a user named oggadm1.

   ```
   GRANT CREATE SESSION, ALTER SESSION TO oggadm1;
   GRANT RESOURCE TO oggadm1;
   GRANT SELECT ANY DICTIONARY TO oggadm1;
   GRANT FLASHBACK ANY TABLE TO oggadm1;
   GRANT SELECT ANY TABLE TO oggadm1;
   GRANT SELECT_CATALOG_ROLE TO rds_master_user_name WITH ADMIN OPTION;
   EXEC rdsadmin.rdsadmin_util.grant_sys_object ('DBA_CLUSTERS', 'OGGADM1');
   GRANT EXECUTE ON DBMS_FLASHBACK TO oggadm1;
   GRANT SELECT ON SYS.V_$DATABASE TO oggadm1;
   GRANT ALTER ANY TABLE TO oggadm1;
   ```

2. Grant the privileges needed by a user account to be an Oracle GoldenGate administrator. Run the following PL/SQL program.

   ```
   EXEC rdsadmin.rdsadmin_dbms_goldengate_auth.grant_admin_privilege (
       grantee                 => 'OGGADM1',
       privilege_type          => 'capture',
       grant_select_privileges => true,
       do_grants               => TRUE);
   ```

   To revoke privileges, use the procedure `revoke_admin_privilege` in the same package.

Step 6: Add a TNS alias for the source database

Add the following entry to `$ORACLE_HOME/network/admin/tnsnames.ora` in the Oracle home to be used by the EXTRACT process. For more information on the `tnsnames.ora` file, see the Oracle documentation.

```
OGGSOURCE=
   (DESCRIPTION=
        (ENABLE=BROKEN)
        (ADDRESS_LIST=
            (ADDRESS=(PROTOCOL=TCP)(HOST=goldengate-source.abcdef12345.us-
west-2.rds.amazonaws.com)(PORT=8200)))
        (CONNECT_DATA=(SERVICE_NAME=ORCL))
    )
```

Setting up a target database for use with Oracle GoldenGate on Amazon RDS

In this task, you set up a target DB instance for use with Oracle GoldenGate.

Setup steps

- Step 1: Set the ENABLE_GOLDENGATE_REPLICATION initialization parameter to true
- Step 2: Create an Oracle GoldenGate user account on the target database
- Step 3: Grant account privileges on the target database
- Step 4: Add a TNS alias for the target database

Step 1: Set the ENABLE_GOLDENGATE_REPLICATION initialization parameter to true

When you set the `ENABLE_GOLDENGATE_REPLICATION` initialization parameter is to `true`, it allows database services to support logical replication. If your source database is on an Amazon RDS DB instance, make sure that you have a parameter group assigned to the DB instance with the `ENABLE_GOLDENGATE_REPLICATION` initialization parameter set to `true`. For more information about the `ENABLE_GOLDENGATE_REPLICATION` initialization parameter, see the Oracle Database documentation.

Step 2: Create an Oracle GoldenGate user account on the target database

Oracle GoldenGate runs as a database user and requires the appropriate database privileges. To make sure it has these privileges, create a user account on the target database.

The following statement creates a user named `oggadm1`.

```
CREATE TABLESPSACE administrator;
CREATE USER oggadm1  IDENTIFIED BY "password"
   DEFAULT TABLESPACE administrator
   TEMPORARY TABLESPACE temp;
ALTER USER oggadm1 QUOTA UNLIMITED ON administrator;
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Step 3: Grant account privileges on the target database

In this task, you grant necessary account privileges for database users on your target database.

To grant account privileges on the target database

1. Grant necessary privileges to the Oracle GoldenGate user account on the target database. In the following example, you grant privileges to `oggadm1`.

```
GRANT CREATE SESSION        TO oggadm1;
GRANT ALTER SESSION         TO oggadm1;
GRANT CREATE CLUSTER        TO oggadm1;
GRANT CREATE INDEXTYPE      TO oggadm1;
GRANT CREATE OPERATOR       TO oggadm1;
GRANT CREATE PROCEDURE      TO oggadm1;
GRANT CREATE SEQUENCE       TO oggadm1;
GRANT CREATE TABLE          TO oggadm1;
GRANT CREATE TRIGGER        TO oggadm1;
GRANT CREATE TYPE           TO oggadm1;
GRANT SELECT ANY DICTIONARY TO oggadm1;
GRANT CREATE ANY TABLE      TO oggadm1;
GRANT ALTER ANY TABLE       TO oggadm1;
GRANT LOCK ANY TABLE        TO oggadm1;
GRANT SELECT ANY TABLE      TO oggadm1;
GRANT INSERT ANY TABLE      TO oggadm1;
GRANT UPDATE ANY TABLE      TO oggadm1;
GRANT DELETE ANY TABLE      TO oggadm1;
```

2. Grant the privileges needed by a user account to be an Oracle GoldenGate administrator. Run the following PL/SQL program.

```
EXEC rdsadmin.rdsadmin_dbms_goldengate_auth.grant_admin_privilege (
    grantee                 => 'OGGADM1',
    privilege_type          => 'apply',
    grant_select_privileges => true,
    do_grants               => TRUE);
```

To revoke privileges, use the procedure `revoke_admin_privilege` in the same package.

Step 4: Add a TNS alias for the target database

Add the following entry to `$ORACLE_HOME/network/admin/tnsnames.ora` in the Oracle home to be used by the `REPLICAT` process. For Oracle Multitenant databases, make sure that the TNS alias points to the service name of the PDB. For more information on the `tnsnames.ora` file, see the Oracle documentation.

```
OGGTARGET=
    (DESCRIPTION=
        (ENABLE=BROKEN)
        (ADDRESS_LIST=
            (ADDRESS=(PROTOCOL=TCP)(HOST=goldengate-target.abcdef12345.us-
west-2.rds.amazonaws.com)(PORT=8200)))
        (CONNECT_DATA=(SERVICE_NAME=ORCL))
    )
```

Working with the EXTRACT and REPLICAT utilities of Oracle GoldenGate

The Oracle GoldenGate utilities `EXTRACT` and `REPLICAT` work together to keep the source and target databases in sync via incremental transaction replication using trail files. All changes that occur on the source database are automatically detected by `EXTRACT`, then formatted and transferred to trail files on the Oracle GoldenGate on-premises or Amazon EC2 instance hub. After initial load is completed, the data is read from these files and replicated to the target database by the `REPLICAT` utility.

Running the Oracle GoldenGate EXTRACT utility

The `EXTRACT` utility retrieves, converts, and outputs data from the source database to trail files. The basic process is as follows:

1. `EXTRACT` queues transaction details to memory or to temporary disk storage.

2. The source database commits the transaction.
3. EXTRACT writes the transaction details to a trail file.
4. The trail file routes these details to the Oracle GoldenGate on-premises or the Amazon EC2 instance hub and then to the target database.

The following steps start the EXTRACT utility, capture the data from EXAMPLE.TABLE in source database OGGSOURCE, and create the trail files.

To run the EXTRACT utility

1. Configure the EXTRACT parameter file on the Oracle GoldenGate hub (on-premises or Amazon EC2 instance). The following listing shows an example EXTRACT parameter file named $GGHOME/dirprm/eabc.prm.

```
EXTRACT EABC

USERID oggadm1@OGGSOURCE, PASSWORD "my-password"
EXTTRAIL /path/to/goldengate/dirdat/ab

IGNOREREPLICATES
GETAPPLOPS
TRANLOGOPTIONS EXCLUDEUSER OGGADM1

TABLE EXAMPLE.TABLE;
```

2. On the Oracle GoldenGate hub, log in to the source database and launch the Oracle GoldenGate command line interface ggsci. The following example shows the format for logging in.

```
dblogin oggadm1@OGGSOURCE
```

3. Add transaction data to turn on supplemental logging for the database table.

```
add trandata EXAMPLE.TABLE
```

4. Using the ggsci command line, enable the EXTRACT utility using the following commands.

```
add extract EABC tranlog, INTEGRATED tranlog, begin now
add exttrail /path/to/goldengate/dirdat/ab
   extract EABC,
```

```
   MEGABYTES 100
```

5. Register the EXTRACT utility with the database so that the archive logs are not deleted. This task allows you to recover old, uncommitted transactions if necessary. To register the EXTRACT utility with the database, use the following command.

```
register EXTRACT EABC, DATABASE
```

6. Start the EXTRACT utility with the following command.

```
start EABC
```

Running the Oracle GoldenGate REPLICAT utility

The REPLICAT utility "pushes" transaction information in the trail files to the target database.

The following steps enable and start the REPLICAT utility so that it can replicate the captured data to the table EXAMPLE.TABLE in target database OGGTARGET.

To run the REPLICATE utility

1. Configure the REPLICAT parameter file on the Oracle GoldenGate hub (on-premises or EC2 instance). The following listing shows an example REPLICAT parameter file named $GGHOME/dirprm/rabc.prm.

```
REPLICAT RABC

USERID oggadm1@OGGTARGET, password "my-password"

ASSUMETARGETDEFS
MAP EXAMPLE.TABLE, TARGET EXAMPLE.TABLE;
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

2. Log in to the target database and launch the Oracle GoldenGate command line interface (ggsci). The following example shows the format for logging in.

```
dblogin userid oggadm1@OGGTARGET
```

3. Using the `ggsci` command line, add a checkpoint table. The user indicated should be the Oracle GoldenGate user account, not the target table schema owner. The following example creates a checkpoint table named `gg_checkpoint`.

```
add checkpointtable oggadm1.oggchkpt
```

4. To enable the `REPLICAT` utility, use the following command.

```
add replicat RABC EXTTRAIL /path/to/goldengate/dirdat/ab CHECKPOINTTABLE
 oggadm1.oggchkpt
```

5. Start the `REPLICAT` utility by using the following command.

```
start RABC
```

Monitoring Oracle GoldenGate

When you use Oracle GoldenGate for replication, make sure that the Oracle GoldenGate process is up and running and the source and target databases are synchronized. You can use the following monitoring tools:

- Amazon CloudWatch is a monitoring service that is used in this pattern to monitor GoldenGate error logs.
- Amazon SNS is a message notification service that is used in this pattern to send email notifications.

For detailed instructions, see Monitor Oracle GoldenGate logs by using Amazon CloudWatch.

Troubleshooting Oracle GoldenGate

This section explains the most common issues when using Oracle GoldenGate with Amazon RDS for Oracle.

Topics

- Error opening an online redo log

- Oracle GoldenGate appears to be properly configured but replication is not working
- Integrated REPLICAT slow due to query on SYS."_DBA_APPLY_CDR_INFO"

Error opening an online redo log

Make sure that you configure your databases to retain archived redo logs. Consider the following guidelines:

- Specify the duration for log retention in hours. The minimum value is one hour.
- Set the duration to exceed any potential downtime of the source DB instance, any potential period of communication, and any potential period of networking issues for the source DB instance. Such a duration lets Oracle GoldenGate recover logs from the source DB instance as needed.
- Ensure that you have sufficient storage on your instance for the files.

If you don't have log retention enabled, or if the retention value is too small, you receive an error message similar to the following.

```
2022-03-06 06:17:27  ERROR   OGG-00446  error 2 (No such file or directory)
opening redo log /rdsdbdata/db/GGTEST3_A/onlinelog/o1_mf_2_9k4bp1n6_.log for sequence
 1306
Not able to establish initial position for begin time 2022-03-06 06:16:55.
```

Oracle GoldenGate appears to be properly configured but replication is not working

For pre-existing tables, you must specify the SCN that Oracle GoldenGate works from.

To fix this issue

1. Log in to the source database and launch the Oracle GoldenGate command line interface (`ggsci`). The following example shows the format for logging in.

   ```
   dblogin userid oggadm1@OGGSOURCE
   ```

2. Using the `ggsci` command line, set up the start SCN for the EXTRACT process. The following example sets the SCN to 223274 for the EXTRACT.

   ```
   ALTER EXTRACT EABC SCN 223274
   ```

```
start EABC
```

3. Log in to the target database. The following example shows the format for logging in.

```
dblogin userid oggadm1@OGGTARGET
```

4. Using the ggsci command line, set up the start SCN for the REPLICAT process. The following example sets the SCN to 223274 for the REPLICAT.

```
start RABC atcsn 223274
```

Integrated REPLICAT slow due to query on SYS."_DBA_APPLY_CDR_INFO"

Oracle GoldenGate Conflict Detection and Resolution (CDR) provides basic conflict resolution routines. For example, CDR can resolve a unique conflict for an INSERT statement.

When CDR resolves a collision, it can insert records into the exception table _DBA_APPLY_CDR_INFO temporarily. Integrated REPLICAT deletes these records later. In a rare scenario, the integrated REPLICAT can process a large number of collisions, but a new integrated REPLICAT does not replace it. Instead of being removed, the existing rows in _DBA_APPLY_CDR_INFO are orphaned. Any new integrated REPLICAT processes slow down because they are querying orphaned rows in _DBA_APPLY_CDR_INFO.

To remove all rows from _DBA_APPLY_CDR_INFO, use the Amazon RDS procedure rdsadmin.rdsadmin_util.truncate_apply$_cdr_info. This procedure is released as part of the October 2020 release and patch update. The procedure is available in the following database versions:

- Version 21.0.0.0.ru-2022-01.rur-2022-01.r1 and higher
- Version 19.0.0.0.ru-2020-10.rur-2020-10.r1 and higher

The following example truncates the table _DBA_APPLY_CDR_INFO.

```
SET SERVEROUTPUT ON SIZE 2000
EXEC rdsadmin.rdsadmin_util.truncate_apply$_cdr_info;
```

Using the Oracle Repository Creation Utility on RDS for Oracle

You can use Amazon RDS to host an RDS for Oracle DB instance that holds the schemas to support your Oracle Fusion Middleware components. Before you can use Fusion Middleware components, create and populate schemas for them in your database. You create and populate the schemas by using the Oracle Repository Creation Utility (RCU).

Supported versions and licensing options for RCU

Amazon RDS supports Oracle Repository Creation Utility (RCU) version 12c only. You can use the RCU in the following configurations:

- RCU 12c with Oracle Database 21c
- RCU 12c with Oracle Database 19c

Before you can use RCU, make sure that you do the following:

- Obtain a license for Oracle Fusion Middleware.
- Follow the Oracle licensing guidelines for the Oracle database that hosts the repository. For more information, see Oracle Fusion Middleware Licensing Information User Manual in the Oracle documentation.

Fusion MiddleWare supports repositories on Oracle Database Enterprise Edition and Standard Edition 2. Oracle recommends Enterprise Edition for production installations that require partitioning and installations that require online index rebuild.

Before you create your RDS for Oracle instance, confirm the Oracle database version that you need to support the components that you want to deploy. To find the requirements for the Fusion Middleware components and versions you want to deploy, use the Certification Matrix. For more information, see Oracle Fusion Middleware Supported System Configurations in the Oracle documentation.

Amazon RDS supports Oracle database version upgrades as needed. For more information, see Upgrading a DB instance engine version.

Requirements and limitations for RCU

To use RCU, you need an Amazon VPC. Your Amazon RDS DB instance must be available only to your Fusion Middleware components, and not to the public Internet. Thus, host your Amazon RDS

DB instance in a private subnet, which provides greater security. You also need an RDS for Oracle DB instance. For more information, see Creating and connecting to an Oracle DB instance.

You can store the schemas for any Fusion Middleware components in your Amazon RDS DB instance. The following schemas have been verified to install correctly:

- Analytics (ACTIVITIES)
- Audit Services (IAU)
- Audit Services Append (IAU_APPEND)
- Audit Services Viewer (IAU_VIEWER)
- Discussions (DISCUSSIONS)
- Metadata Services (MDS)
- Oracle Business Intelligence (BIPLATFORM)
- Oracle Platform Security Services (OPSS)
- Portal and Services (WEBCENTER)
- Portlet Producers (PORTLET)
- Service Table (STB)
- SOA Infrastructure (SOAINFRA)
- User Messaging Service (UCSUMS)
- WebLogic Services (WLS)

Guidelines for using RCU

The following are some recommendations for working with your DB instance in this scenario:

- We recommend that you use Multi-AZ for production workloads. For more information about working with multiple Availability Zones, see Regions, Availability Zones, and Local Zones.
- For additional security, Oracle recommends that you use Transparent Data Encryption (TDE) to encrypt your data at rest. If you have an Enterprise Edition license that includes the Advanced Security Option, you can enable encryption at rest by using the TDE option. For more information, see Oracle Transparent Data Encryption.

  Amazon RDS also provides an encryption at rest option for all database editions. For more information, see Encrypting Amazon RDS resources.

- Configure your VPC Security Groups to allow communication between your application servers and your Amazon RDS DB instance. The application servers that host the Fusion Middleware components can be on Amazon EC2 or on-premises.

Running RCU

To create and populate the schemas to support your Fusion Middleware components, use the Oracle Repository Creation Utility (RCU). You can run RCU in different ways.

Topics

- Running RCU using the command line in one step
- Running RCU using the command line in multiple steps
- Running RCU in interactive mode

Running RCU using the command line in one step

If you don't need to edit any of your schemas before populating them, you can run RCU in a single step. Otherwise, see the following section for running RCU in multiple steps.

You can run the RCU in silent mode by using the command-line parameter `-silent`. When you run RCU in silent mode, you can avoid entering passwords on the command line by creating a text file containing the passwords. Create a text file with the password for dbUser on the first line, and the password for each component on subsequent lines. You specify the name of the password file as the last parameter to the RCU command.

Example

The following example creates and populates schemas for the SOA Infrastructure component (and its dependencies) in a single step.

For Linux, macOS, or Unix:

```
export ORACLE_HOME=/u01/app/oracle/product/12.2.1.0/fmw
export JAVA_HOME=/usr/java/jdk1.8.0_65
${ORACLE_HOME}/oracle_common/bin/rcu \
-silent \
-createRepository \
-connectString ${dbhost}:${dbport}:${dbname} \
-dbUser ${dbuser} \
```

```
-dbRole Normal \
-honorOMF \
-schemaPrefix ${SCHEMA_PREFIX} \
-component MDS \
-component STB \
-component OPSS \
-component IAU \
-component IAU_APPEND \
-component IAU_VIEWER \
-component UCSUMS \
-component WLS \
-component SOAINFRA \
-f < /tmp/passwordfile.txt
```

For more information, see Running Repository Creation Utility from the command line in the Oracle documentation.

Running RCU using the command line in multiple steps

To manually edit your schema scripts, run RCU in multiple steps:

1. Run RCU in **Prepare Scripts for System Load** mode by using the `-generateScript` command-line parameter to create the scripts for your schemas.
2. Manually edit and run the generated script `script_systemLoad.sql`.
3. Run RCU again in **Perform Product Load** mode by using the `-dataLoad` command-line parameter to populate the schemas.
4. Run the generated cleanup script `script_postDataLoad.sql`.

To run RCU in silent mode, specify the command-line parameter `-silent`. When you run RCU in silent mode, you can avoid typing passwords on the command line by creating a text file containing the passwords. Create a text file with the password for `dbUser` on the first line, and the password for each component on subsequent lines. Specify the name of the password file as the last parameter to the RCU command.

Example

The following example creates schema scripts for the SOA Infrastructure component and its dependencies.

For Linux, macOS, or Unix:

```
export ORACLE_HOME=/u01/app/oracle/product/12.2.1.0/fmw
export JAVA_HOME=/usr/java/jdk1.8.0_65
${ORACLE_HOME}/oracle_common/bin/rcu \
-silent \
-generateScript \
-connectString ${dbhost}:${dbport}:${dbname} \
-dbUser ${dbuser} \
-dbRole Normal \
-honorOMF \
[-encryptTablespace true] \
-schemaPrefix ${SCHEMA_PREFIX} \
-component MDS \
-component STB \
-component OPSS \
-component IAU \
-component IAU_APPEND \
-component IAU_VIEWER \
-component UCSUMS \
-component WLS \
-component SOAINFRA \
-scriptLocation /tmp/rcuscripts \
-f < /tmp/passwordfile.txt
```

Now you can edit the generated script, connect to your Oracle DB instance, and run the script. The generated script is named `script_systemLoad.sql`. For information about connecting to your Oracle DB instance, see Step 3: Connect your SQL client to an Oracle DB instance.

The following example populates the schemas for the SOA Infrastructure component (and its dependencies).

For Linux, macOS, or Unix:

```
export JAVA_HOME=/usr/java/jdk1.8.0_65
${ORACLE_HOME}/oracle_common/bin/rcu \
-silent \
-dataLoad \
-connectString ${dbhost}:${dbport}:${dbname} \
-dbUser ${dbuser} \
-dbRole Normal \
-honorOMF \
-schemaPrefix ${SCHEMA_PREFIX} \
-component MDS \
```

```
-component STB \
-component OPSS \
-component IAU \
-component IAU_APPEND \
-component IAU_VIEWER \
-component UCSUMS \
-component WLS \
-component SOAINFRA \
-f < /tmp/passwordfile.txt
```

To finish, you connect to your Oracle DB instance, and run the clean-up script. The script is named `script_postDataLoad.sql`.

For more information, see Running Repository Creation Utility from the command line in the Oracle documentation.

Running RCU in interactive mode

To use the RCU graphical user interface, run RCU in interactive mode. Include the `-interactive` parameter and omit the `-silent` parameter. For more information, see Understanding Repository Creation Utility screens in the Oracle documentation.

Example

The following example starts RCU in interactive mode and pre-populates the connection information.

For Linux, macOS, or Unix:

```
export ORACLE_HOME=/u01/app/oracle/product/12.2.1.0/fmw
export JAVA_HOME=/usr/java/jdk1.8.0_65
${ORACLE_HOME}/oracle_common/bin/rcu \
-interactive \
-createRepository \
-connectString ${dbhost}:${dbport}:${dbname} \
-dbUser ${dbuser} \
-dbRole Normal
```

Troubleshooting RCU

Be mindful of the following issues.

Topics

- Oracle Managed Files (OMF)
- Object privileges
- Enterprise Scheduler Service

Oracle Managed Files (OMF)

Amazon RDS uses OMF data files to simplify storage management. You can customize tablespace attributes, such as size and extent management. However, if you specify a data file name when you run RCU, the tablespace code fails with `ORA-20900`. You can use RCU with OMF in the following ways:

- In RCU 12.2.1.0 and later, use the `-honorOMF` command-line parameter.
- In RCU 12.1.0.3 and later, use multiple steps and edit the generated script. For more information, see Running RCU using the command line in multiple steps.

Object privileges

Because Amazon RDS is a managed service, you don't have full `SYSDBA` access to your RDS for Oracle DB instance. However, RCU 12c supports users with lower privileges. In most cases, the master user privilege is sufficient to create repositories.

The master account can directly grant privileges that it has already been granted `WITH GRANT OPTION`. In some cases, when you attempt to grant SYS object privileges, the RCU might fail with `ORA-01031`. You can retry and run the `rdsadmin_util.grant_sys_object` stored procedure, as shown in the following example:

```
BEGIN
  rdsadmin.rdsadmin_util.grant_sys_object('GV_$SESSION','MY_DBA','SELECT');
END;
/
```

If you attempt to grant SYS privileges on the object `SCHEMA_VERSION_REGISTRY`, the operation might fail with `ORA-20199: Error in rdsadmin_util.grant_sys_object`. You can qualify the table `SCHEMA_VERSION_REGISTRY$` and the view `SCHEMA_VERSION_REGISTRY` with the schema owner name, which is `SYSTEM`, and retry the operation. Or, you can create a synonym. Log in as the master user and run the following statements:

```
CREATE OR REPLACE VIEW SYSTEM.SCHEMA_VERSION_REGISTRY
  AS SELECT * FROM SYSTEM.SCHEMA_VERSION_REGISTRY$;
CREATE OR REPLACE PUBLIC SYNONYM SCHEMA_VERSION_REGISTRY FOR
 SYSTEM.SCHEMA_VERSION_REGISTRY;
CREATE OR REPLACE PUBLIC SYNONYM SCHEMA_VERSION_REGISTRY$ FOR SCHEMA_VERSION_REGISTRY;
```

Enterprise Scheduler Service

When you use the RCU to drop an Enterprise Scheduler Service repository, the RCU might fail with `Error: Component drop check failed`.

Configuring Oracle Connection Manager on an Amazon EC2 instance

Oracle Connection Manager (CMAN) is a proxy server that forwards connection requests to database servers or other proxy servers. You can use CMAN to configure the following:

Access control

You can create rules that filter out user-specified client requests and accept others.

Session multiplexing

You can funnel multiple client sessions through a network connection to a shared server destination.

Typically, CMAN resides on a host separate from the database server and client hosts. For more information, see Configuring Oracle Connection Manager in the Oracle Database documentation.

Topics

- Supported versions and licensing options for CMAN
- Requirements and limitations for CMAN
- Configuring CMAN

Supported versions and licensing options for CMAN

CMAN supports the Enterprise Edition of all versions of Oracle Database that Amazon RDS supports. For more information, see RDS for Oracle releases.

You can install Oracle Connection Manager on a separate host from the host where Oracle Database is installed. You don't need a separate license for the host that runs CMAN.

Requirements and limitations for CMAN

To provide a fully managed experience, Amazon RDS restricts access to the operating system. You can't modify database parameters that require operating system access. Thus, Amazon RDS doesn't support features of CMAN that require you to log in to the operating system.

Configuring CMAN

When you configure CMAN, you perform most of the work outside of your RDS for Oracle database.

Topics

- Step 1: Configure CMAN on an Amazon EC2 instance in the same VPC as the RDS for Oracle instance
- Step 2: Configure database parameters for CMAN
- Step 3: Associate your DB instance with the parameter group

Step 1: Configure CMAN on an Amazon EC2 instance in the same VPC as the RDS for Oracle instance

To learn how to set up CMAN, follow the detailed instructions in the blog post Configuring and using Oracle Connection Manager on Amazon EC2 for Amazon RDS for Oracle.

Step 2: Configure database parameters for CMAN

For CMAN features such as Traffic Director Mode and session multiplexing, set `REMOTE_LISTENER` parameter to the address of CMAN instance in a DB parameter group. Consider the following scenario:

- The CMAN instance resides on a host with IP address `10.0.159.100` and uses port `1521`.
- The databases `orcla`, `orclb`, and `orclc` reside on separate RDS for Oracle DB instances.

The following table shows how to set the `REMOTE_LISTENER` value. The `LOCAL_LISTENER` value is set automatically by Amazon RDS.

DB instance name	DB instance IP	Local listener value (set automatically)	Remote listener value (set by user)
`orcla`	`10.0.159.200`	`( address= (protocol=tcp) (host=10.0.159.200) (port=1521) )`	`10.0.159.100:1521`
`orclb`	`10.0.159.300`	`( address= (protocol=tcp) (host=10.0.159.300)`	`10.0.159.100:1521`

DB instance name	DB instance IP	Local listener value (set automatically)	Remote listener value (set by user)
		(port=1521))	
orclc	10.0.159.400	(address= (protocol=tcp) (host=10.0.159.400) (port=1521))	10.0.159.100:1521

Step 3: Associate your DB instance with the parameter group

Create or modify your DB instance to use the parameter group that you configured in Step 2: Configure database parameters for CMAN. For more information, see Associating a DB parameter group with a DB instance in Amazon RDS.

Installing a Siebel database on Oracle on Amazon RDS

You can use Amazon RDS to host a Siebel Database on an Oracle DB instance. The Siebel Database is part of the Siebel Customer Relationship Management (CRM) application architecture. For an illustration, see Generic architecture of Siebel business application.

Use the following topic to help set up a Siebel Database on an Oracle DB instance on Amazon RDS. You can also find out how to use Amazon Web Services to support the other components required by the Siebel CRM application architecture.

> **Note**
>
> To install a Siebel Database on Oracle on Amazon RDS, you need to use the master user account. You don't need SYSDBA privilege; master user privilege is sufficient. For more information, see Master user account privileges.

Licensing and versions

To install a Siebel Database on Amazon RDS, you must use your own Oracle Database license, and your own Siebel license. You must have the appropriate Oracle Database license (with Software Update License and Support) for the DB instance class and Oracle Database edition. For more information, see RDS for Oracle licensing options.

Oracle Database Enterprise Edition is the only edition certified by Siebel for this scenario. Amazon RDS supports Siebel CRM version 15.0 or 16.0.

Amazon RDS supports database version upgrades. For more information, see Upgrading a DB instance engine version.

Before you begin

Before you begin, you need an Amazon VPC. Because your Amazon RDS DB instance needs to be available only to your Siebel Enterprise Server, and not to the public Internet, your Amazon RDS DB instance is hosted in a private subnet, providing greater security. For information about how to create an Amazon VPC for use with Siebel CRM, see Creating and connecting to an Oracle DB instance.

Before you begin, you also need an Oracle DB instance. For information about how to create an Oracle DB instance for use with Siebel CRM, see Creating an Amazon RDS DB instance.

Installing and configuring a Siebel database

After you create your Oracle DB instance, you can install your Siebel Database. You install the database by creating table owner and administrator accounts, installing stored procedures and functions, and then running the Siebel Database Configuration Wizard. For more information, see Installing the Siebel database on the RDBMS.

To run the Siebel Database Configuration Wizard, you need to use the master user account. You don't need SYSDBA privilege; master user privilege is sufficient. For more information, see Master user account privileges.

Using other Amazon RDS features with a Siebel database

After you create your Oracle DB instance, you can use additional Amazon RDS features to help you customize your Siebel Database.

Collecting statistics with the Oracle Statspack option

You can add features to your DB instance through the use of options in DB option groups. When you created your Oracle DB instance, you used the default DB option group. If you want to add features to your database, you can create a new option group for your DB instance.

If you want to collect performance statistics on your Siebel Database, you can add the Oracle Statspack feature. For more information, see Oracle Statspack.

Some option changes are applied immediately, and some option changes are applied during the next maintenance window for the DB instance. For more information, see Working with option groups. After you create a customized option group, modify your DB instance to attach it. For more information, see Modifying an Amazon RDS DB instance.

Performance tuning with parameters

You manage your DB engine configuration through the use of parameters in a DB parameter group. When you created your Oracle DB instance, you used the default DB parameter group. If you want to customize your database configuration, you can create a new parameter group for your DB instance.

When you change a parameter, depending on the type of the parameter, the changes are applied either immediately or after you manually reboot the DB instance. For more information, see Parameter groups for Amazon RDS. After you create a customized parameter group, modify your DB instance to attach it. For more information, see Modifying an Amazon RDS DB instance.

To optimize your Oracle DB instance for Siebel CRM, you can customize certain parameters. The following table shows some recommended parameter settings. For more information about performance tuning Siebel CRM, see Siebel CRM Performance Tuning Guide.

Parameter name	Default value	Guidance for optimal Siebel CRM performance
_always_semi_join	CHOOSE	OFF
_b_tree_bitmap_plans	TRUE	FALSE
_like_with_bind_as_equality	FALSE	TRUE
_no_or_expansion	FALSE	FALSE
_optimizer_join_sel_sanity_check	TRUE	TRUE
_optimizer_max_permutations	2000	100
_optimizer_sortmerge_join_enabled	TRUE	FALSE
_partition_view_enabled	TRUE	FALSE

Parameter name	Default value	Guidance for optimal Siebel CRM performance
`open_cursors`	`300`	At least **`2000`**.

Creating snapshots

After you create your Siebel Database, you can copy the database by using the snapshot features of Amazon RDS. For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS and Restoring to a DB instance.

Support for other Siebel CRM components

In addition to your Siebel Database, you can also use Amazon Web Services to support the other components of your Siebel CRM application architecture. You can find more information about the support provided by Amazon AWS for additional Siebel CRM components in the following table.

Siebel CRM component	Amazon AWS Support
Siebel Enterprise (with one or more Siebel Servers)	You can host your Siebel Servers on Amazon Elastic Compute Cloud (Amazon EC2) instances. You can use Amazon EC2 to launch as many or as few virtual servers as you need. Using Amazon EC2, you can scale up or down easily to handle changes in requirements. For more information, see What is Amazon EC2? You can put your servers in the same VPC with your DB instance and use the VPC security group to access the database. For more information, see Working with a DB instance in a VPC.
Web Servers (with Siebel Web Server Extensions)	You can install multiple Web Servers on multiple EC2 instances. You can then use Elastic Load Balancing to distribute incoming traffic among the instances. For more information, see What is Elastic Load Balancing?

Siebel CRM component	Amazon AWS Support
Siebel Gateway Name Server	You can host your Siebel Gateway Name Server on an EC2 instance. You can then put your server in the same VPC with the DB instance and use the VPC security group to access the database. For more information, see Working with a DB instance in a VPC.

Oracle Database engine release notes

Updates to your Amazon RDS for Oracle DB instances keep them current. If you apply updates, you can be confident that your DB instance is running a version of the database software that has been tested by both Oracle and Amazon. We don't support applying one-off patches to individual RDS for Oracle DB instances.

You can specify any currently supported Oracle Database version when you create a new DB instance. You can specify the major version, such as Oracle Database 19c, and any supported minor version for the specified major version. If no version is specified, Amazon RDS defaults to a supported version, typically the most recent version. If a major version is specified but a minor version is not, Amazon RDS defaults to a recent release of the major version that you have specified. To see a list of supported versions and defaults for newly created DB instances, use the `describe-db-engine-versions` AWS CLI command.

For details about the Oracle Database versions that Amazon RDS supports, see the *Amazon RDS for Oracle Release Notes*.

Amazon RDS for PostgreSQL

Amazon RDS supports DB instances running several versions of PostgreSQL. For a list of available versions, see Available PostgreSQL database versions.

You can create DB instances and DB snapshots, point-in-time restores and backups. DB instances running PostgreSQL support Multi-AZ deployments, read replicas, Provisioned IOPS, and can be created inside a virtual private cloud (VPC). You can also use Secure Socket Layer (SSL) to connect to a DB instance running PostgreSQL.

Before creating a DB instance, make sure to complete the steps in Setting up your Amazon RDS environment.

You can use any standard SQL client application to run commands for the instance from your client computer. Such applications include pgAdmin, a popular Open Source administration and development tool for PostgreSQL, or psql, a command line utility that is part of a PostgreSQL installation. To deliver a managed service experience, Amazon RDS doesn't provide host access to DB instances. Also, it restricts access to certain system procedures and tables that require advanced privileges. Amazon RDS supports access to databases on a DB instance using any standard SQL client application. Amazon RDS doesn't allow direct host access to a DB instance by using Telnet or Secure Shell (SSH).

Amazon RDS for PostgreSQL is compliant with many industry standards. For example, you can use Amazon RDS for PostgreSQL databases to build HIPAA-compliant applications and to store healthcare-related information. This includes storage for protected health information (PHI) under a completed Business Associate Agreement (BAA) with AWS. Amazon RDS for PostgreSQL also meets Federal Risk and Authorization Management Program (FedRAMP) security requirements. Amazon RDS for PostgreSQL has received a FedRAMP Joint Authorization Board (JAB) Provisional Authority to Operate (P-ATO) at the FedRAMP HIGH Baseline within the AWS GovCloud (US) Regions. For more information on supported compliance standards, see AWS cloud compliance.

To import PostgreSQL data into a DB instance, follow the information in the Importing data into PostgreSQL on Amazon RDS section.

Important

If you encounter an issue with your RDS for PostgreSQL DB instance, your AWS support agent might need more information about the health of your databases. The goal is to ensure that AWS Support gets the required information as soon as possible.

You can use PG Collector to help gather valuable database information in a consolidated HTML file. For more information on PG Collector, how to run it, and how to download the HTML report, see PG Collector.
Upon successful completion, and unless otherwise noted, the script returns output in a readable HTML format. The script is designed to exclude any data or security details from the HTML that might compromise your business. It also makes no modifications to your database or its environment. However, if you find any information in the HTML that you are uncomfortable sharing, feel free to remove the problematic information before uploading the HTML. When the HTML is acceptable, upload it using the attachments section in the case details of your support case.

Topics

- Common management tasks for Amazon RDS for PostgreSQL
- Working with the Database Preview environment
- Available PostgreSQL database versions
- Understanding the RDS for PostgreSQL incremental release process
- Supported PostgreSQL extension versions
- Working with PostgreSQL features supported by Amazon RDS for PostgreSQL
- Connecting to a DB instance running the PostgreSQL database engine
- Securing connections to RDS for PostgreSQL with SSL/TLS
- Using Kerberos authentication with Amazon RDS for PostgreSQL
- Using a custom DNS server for outbound network access
- Upgrades of the RDS for PostgreSQL DB engine
- Upgrading a PostgreSQL DB snapshot engine version
- Working with read replicas for Amazon RDS for PostgreSQL
- Improving query performance for RDS for PostgreSQL with Amazon RDS Optimized Reads
- Importing data into PostgreSQL on Amazon RDS
- Exporting data from an RDS for PostgreSQL DB instance to Amazon S3
- Invoking an AWS Lambda function from an RDS for PostgreSQL DB instance
- Common DBA tasks for Amazon RDS for PostgreSQL
- Tuning with wait events for RDS for PostgreSQL

- Tuning RDS for PostgreSQL with Amazon DevOps Guru proactive insights
- Using PostgreSQL extensions with Amazon RDS for PostgreSQL
- Working with the supported foreign data wrappers for Amazon RDS for PostgreSQL
- Working with Trusted Language Extensions for PostgreSQL

Common management tasks for Amazon RDS for PostgreSQL

The following are the common management tasks you perform with an Amazon RDS for PostgreSQL DB instance, with links to relevant documentation for each task.

Task area	Relevant documentation
Setting up Amazon RDS for first-time use Before you can create your DB instance, make sure to complete a few prerequisites. For example, DB instances are created by default with a firewall that prevents access to it. So you need to create a security group with the correct IP addresses and network configuration to access the DB instance.	Setting up your Amazon RDS environment
Understanding Amazon RDS DB instances If you are creating a DB instance for production purposes, you should understand how instance classes, storage types, and Provisioned IOPS work in Amazon RDS.	DB instance classes Amazon RDS storage types Provisioned IOPS SSD storage
Finding available PostgreSQL versions Amazon RDS supports several versions of PostgreSQL.	Available PostgreSQL database versions
Setting up high availability and failover support A production DB instance should use Multi-AZ deployments. Multi-AZ deployments provide increased availability, data durability, and fault tolerance for DB instances.	Configuring and managing a Multi-AZ deployment for Amazon RDS
Understanding the Amazon Virtual Private Cloud (VPC) network	Working with a DB instance in a VPC

Task area	Relevant documentation
If your AWS account has a default VPC, then your DB instance is automatically created inside the default VPC. In some cases, your account might not have a default VPC, and you might want the DB instance in a VPC. In these cases, create the VPC and subnet groups before you create the DB instance.	
Importing data into Amazon RDS PostgreSQL You can use several different tools to import data into your PostgreSQL DB instance on Amazon RDS.	Importing data into PostgreSQL on Amazon RDS
Setting up read-only read replicas (primary and standbys) RDS for PostgreSQL supports read replicas in both the same AWS Region and in a different AWS Region from the primary instance.	Working with DB instance read replicas Working with read replicas for Amazon RDS for PostgreSQL Creating a read replica in a different AWS Region
Understanding security groups By default, DB instances are created with a firewall that prevents access to them. To provide access through that firewall, you edit the inbound rules for the VPC security group associated with the VPC hosting the DB instance.	Controlling access with security groups
Setting up parameter groups and features To change the default parameters for your DB instance, create a custom DB parameter group and change settings to that. If you do this before creating your DB instance, you can choose your custom DB parameter group when you create the instance.	Parameter groups for Amazon RDS

Task area	Relevant documentation
Connecting to your PostgreSQL DB instance After creating a security group and associating it to a DB instance, you can connect to the DB instance using any standard SQL client application such as `psql` or `pgAdmin`.	Connecting to a DB instance running the PostgreSQL database engine Using SSL with a PostgreSQL DB instance
Backing up and restoring your DB instance You can configure your DB instance to take automated backups, or take manual snapshots, and then restore instances from the backups or snapshots.	Backing up, restoring, and exporting data
Monitoring the activity and performance of your DB instance You can monitor a PostgreSQL DB instance by using CloudWatch Amazon RDS metrics, events, and enhanced monitoring.	Viewing metrics in the Amazon RDS console Viewing Amazon RDS events
Upgrading the PostgreSQL database version You can do both major and minor version upgrades for your PostgreSQL DB instance.	Upgrades of the RDS for PostgreSQL DB engine Choosing a major version for an RDS for PostgreSQL upgrade
Working with log files You can access the log files for your PostgreSQL DB instance.	RDS for PostgreSQL database log files
Understanding the best practices for PostgreSQL DB instances Find some of the best practices for working with PostgreSQL on Amazon RDS.	Best practices for working with PostgreSQL

Following is a list of other sections in this guide that can help you understand and use important features of RDS for PostgreSQL:

- Understanding PostgreSQL roles and permissions
- Controlling user access to the PostgreSQL database
- Working with parameters on your RDS for PostgreSQL DB instance
- Understanding logging mechanisms supported by RDS for PostgreSQL
- Working with PostgreSQL autovacuum on Amazon RDS for PostgreSQL
- Using a custom DNS server for outbound network access

Working with the Database Preview environment

The PostgreSQL community continuously releases new PostgreSQL version and extensions, including beta versions. This gives PostgreSQL users the opportunity to try out a new PostgreSQL version early. To learn more about the PostgreSQL community beta release process, see Beta Information in the PostgreSQL documentation. Similarly, Amazon RDS makes certain PostgreSQL beta versions available as Preview releases. This allows you to create DB instances using the Preview version and test out its features in the Database Preview Environment.

RDS for PostgreSQL DB instances in the Database Preview Environment are functionally similar to other RDS for PostgreSQL instances. However, you can't use a Preview version for production.

Keep in mind the following important limitations:

- All DB instances are deleted 60 days after you create them, along with any backups and snapshots.
- You can only create a DB instance in a virtual private cloud (VPC) based on the Amazon VPC service.
- You can only use General Purpose SSD and Provisioned IOPS SSD storage.
- You can't get help from AWS Support with DB instances. Instead, you can post your questions to the AWS-managed Q&A community, AWS re:Post.
- You can't copy a snapshot of a DB instance to a production environment.

The following options are supported by the Preview.

- You can create DB instances using M6i, R6i, M6g, M5, T3, R6g, and R5 instance types only. For more information about RDS instance classes, see DB instance classes.
- You can use both single-AZ and multi-AZ deployments.
- You can use standard PostgreSQL dump and load functions to export databases from or import databases to the Database Preview Environment.

Topics

- Features not supported in the Database Preview environment
- PostgreSQL version 17 in the Database Preview environment
- Creating a new DB instance in the Database Preview environment

Features not supported in the Database Preview environment

The following features aren't available in the Database Preview environment:

- Cross-Region snapshot copy
- Cross-Region read replicas

PostgreSQL version 17 in the Database Preview environment

> **Note**
>
> This is preview documentation for Amazon RDS PostgreSQL version 17. It is subject to change.

PostgreSQL version 17.0 is now available in the Amazon RDS Database Preview environment. PostgreSQL version 17.0 contains several improvements that are described in the following PostgreSQL documentation, PostgreSQL 17 Released!

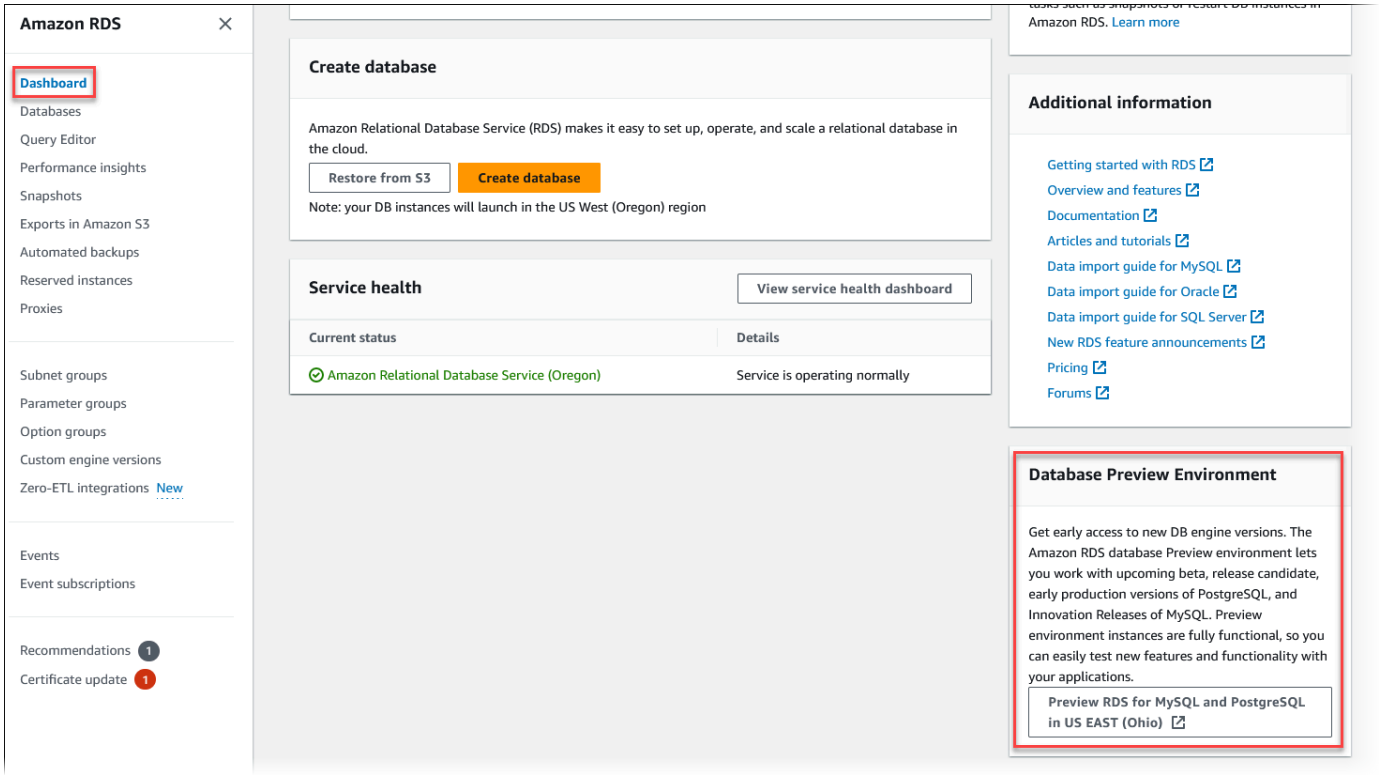
For information on the Database Preview Environment, see the section called “Working with the Database Preview environment”. To access the Preview Environment from the console, select https://console.aws.amazon.com/rds-preview/.

Creating a new DB instance in the Database Preview environment

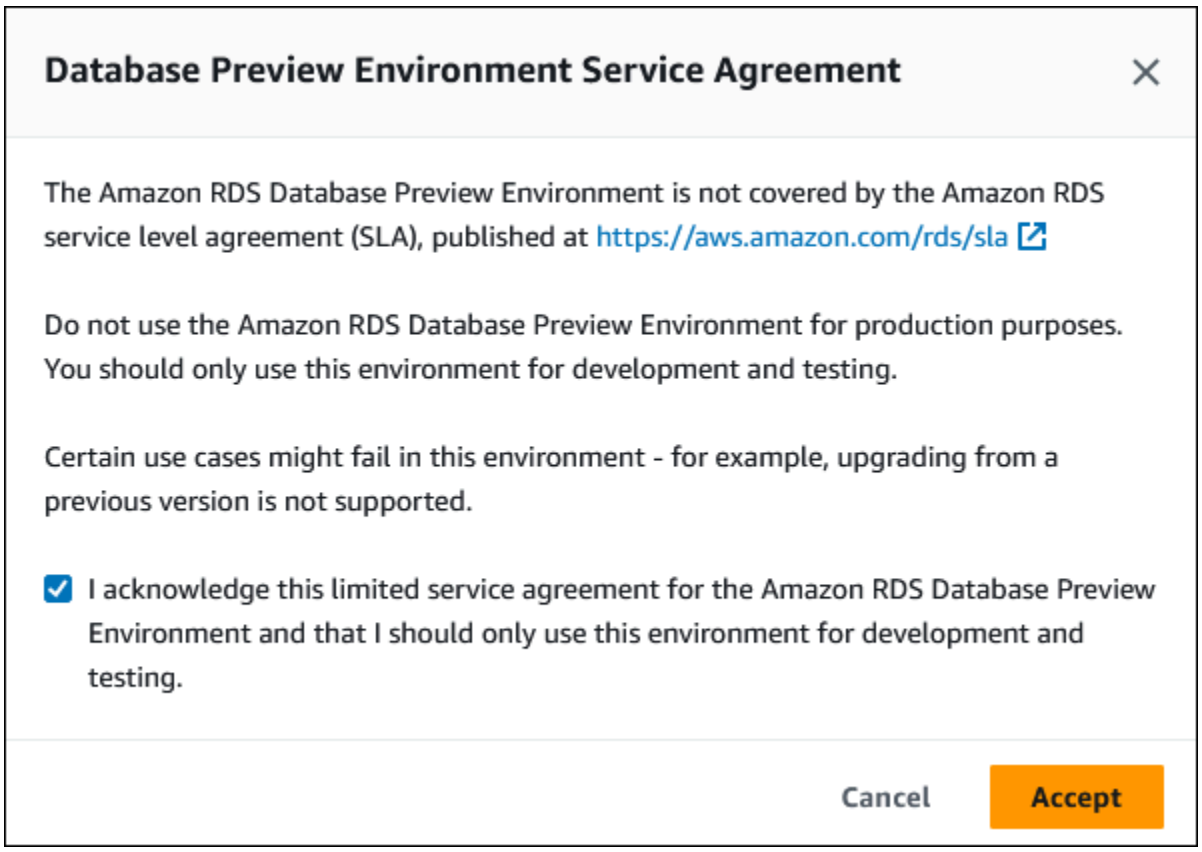
Use the following procedure to create a DB instance in the preview environment.

To create a DB instance in the Database Preview environment

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose **Dashboard** from the navigation pane.
3. In the Dashboard page, locate the **Database Preview Environment** section on the Dashboard page, as shown in the following image.



You can navigate directly to the Database Preview environment. Before you can proceed, you must acknowledge and accept the limitations.



4. To create the RDS for PostgreSQL DB instance, follow the same process as that for creating any Amazon RDS DB instance. For more information, see the Console procedure in Creating a DB instance.

To create an instance in the Database Preview Environment using the RDS API or the AWS CLI, use the following endpoint.

```
rds-preview.us-east-2.amazonaws.com
```

Available PostgreSQL database versions

Amazon RDS supports DB instances running several editions of PostgreSQL. You can specify any currently available PostgreSQL version when creating a new DB instance. You can specify the major version (such as PostgreSQL 14), and any available minor version for the specified major version. If no version is specified, Amazon RDS defaults to an available version, typically the most recent version. If a major version is specified but a minor version is not, Amazon RDS defaults to a recent release of the major version you have specified.

To see a list of available versions, as well as defaults for newly created DB instances, use the `describe-db-engine-versions` AWS CLI command. For example, to display the default PostgreSQL engine version, use the following command:

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

For details about the PostgreSQL versions that are supported on Amazon RDS, see the *Amazon RDS for PostgreSQL Release Notes*. You can also view information about support dates for major engine versions by running the describe-db-major-engine-versions AWS CLI command or by using the DescribeDBMajorEngineVersions RDS API operation.

If you aren't ready to manually upgrade to a new major engine version before the RDS end of standard support date, Amazon RDS will automatically enroll your databases in Amazon RDS Extended Support after the RDS end of standard support date. Then, you can continue to run RDS for PostgreSQL version 11 and higher. For more information, see Amazon RDS Extended Support with Amazon RDS and Amazon RDS pricing.

Deprecated versions for Amazon RDS for PostgreSQL

Note the following deprecated versions:

- RDS for PostgreSQL 10 was deprecated in February 2023.
- RDS for PostgreSQL 9.6 was deprecated in March 2022.
- RDS for PostgreSQL 9.5 was deprecated in March 2021.

To learn more about deprecation policy for RDS for PostgreSQL, see Amazon RDS FAQs. For more information about PostgreSQL versions, see Versioning Policy in the PostgreSQL documentation.

Understanding the RDS for PostgreSQL incremental release process

RDS for PostgreSQL delivers security fixes, performance improvements, and new features through incremental releases while maintaining minor version compatibility. These releases are labeled as R1, R2, R3, and so on.

Release version naming convention

- R1 is the initial release of a minor version. It occasionally includes new features, extensions, or upgrades to existing extensions.
- Subsequent release versions (R2, R3, and later) include:
  - Security updates
  - Performance improvements
  - Bug fixes
  - Extension updates

Advantages of RDS for PostgreSQL incremental release process

The incremental release process provides the following advantages:

- Quick adoption of new PostgreSQL community releases while separately managing RDS-specific enhancements through subsequent releases. This streamlines the release process and ensures faster delivery of critical updates.
- Access to bug fixes, new features, security updates, and extension updates while maintaining compatibility with the PostgreSQL minor version.

Managing release updates

Amazon RDS notifies you about new incremental releases through pending maintenance actions in the AWS Management Console. You can update your database using one of these methods:

- Enable automatic updates during scheduled maintenance windows.
- Apply updates manually through pending maintenance actions.

- Use Blue/Green deployments with physical replication to minimize downtime. For more information, see Blue/Green Deployments support minor version upgrade for RDS for PostgreSQL.

Before updating your database, consider the following key points:

- Database reboots are required for updates unless you use Blue/Green deployments with physical replication.
- Some incremental releases are mandatory, particularly those that include security fixes.

For more information about updating your Amazon RDS DB instance instance, see PostgreSQL trusted extensions and apply-pending-maintenance-action.

Supported PostgreSQL extension versions

RDS for PostgreSQL supports many PostgreSQL extensions. The PostgreSQL community sometimes refers to these as modules. Extensions expand on the functionality provided by the PostgreSQL engine. You can find a list of extensions supported by Amazon RDS in the default DB parameter group for that PostgreSQL version. You can also see the current extensions list using `psql` by showing the `rds.extensions` parameter as in the following example.

```
SHOW rds.extensions;
```

> **Note**
>
> Parameters added in a minor version release might display inaccurately when using the `rds.extensions` parameter in `psql`.

As of RDS for PostgreSQL 13, certain extensions can be installed by database users other than the `rds_superuser`. These are known as *trusted extensions*. To learn more, see PostgreSQL trusted extensions.

Certain versions of RDS for PostgreSQL support the `rds.allowed_extensions` parameter. This parameter lets an `rds_superuser` limit the extensions that can be installed in the RDS for PostgreSQL DB instance. For more information, see Restricting installation of PostgreSQL extensions.

For lists of PostgreSQL extensions and versions that are supported by each available RDS for PostgreSQL version, see PostgreSQL extensions supported on Amazon RDS in *Amazon RDS for PostgreSQL Release Notes*.

Restricting installation of PostgreSQL extensions

You can restrict which extensions can be installed on a PostgreSQL DB instance. By default, this parameter isn't set, so any supported extension can be added if the user has permissions to do so. To do so, set the `rds.allowed_extensions` parameter to a string of comma-separated extension names. By adding a list of extensions to this parameter, you explicitly identify the extensions that your RDS for PostgreSQL DB instance can use. Only these extensions can then be installed in the PostgreSQL DB instance.

The default string for the `rds.allowed_extensions` parameter is '*', which means that any extension available for the engine version can be installed. Changing the `rds.allowed_extensions` parameter does not require a database restart because it's a dynamic parameter.

The PostgreSQL DB instance engine must be one of the following versions for you to use the `rds.allowed_extensions` parameter:

- All PostgreSQL 16 versions
- PostgreSQL 15 and all higher versions
- PostgreSQL 14 and all higher versions
- PostgreSQL 13.3 and higher minor versions
- PostgreSQL 12.7 and higher minor versions

To see which extension installations are allowed, use the following psql command.

```
postgres=> SHOW rds.allowed_extensions;
 rds.allowed_extensions
------------------------
 *
```

If an extension was installed prior to it being left out of the list in the `rds.allowed_extensions` parameter, the extension can still be used normally, and commands such as `ALTER EXTENSION` and `DROP EXTENSION` will continue to work. However, after an extension is restricted, `CREATE EXTENSION` commands for the restricted extension will fail.

Installation of extension dependencies with `CREATE EXTENSION CASCADE` are also restricted. The extension and its dependencies must be specified in `rds.allowed_extensions`. If an extension dependency installation fails, the entire `CREATE EXTENSION CASCADE` statement will fail.

If an extension is not included with the `rds.allowed_extensions` parameter, you will see an error such as the following if you try to install it.

```
ERROR: permission denied to create extension "extension-name"
HINT: This extension is not specified in "rds.allowed_extensions".
```

PostgreSQL trusted extensions

To install most PostgreSQL extensions requires `rds_superuser` privileges. PostgreSQL 13 introduced trusted extensions, which reduce the need to grant `rds_superuser` privileges to regular users. With this feature, users can install many extensions if they have the CREATE privilege on the current database instead of requiring the `rds_superuser` role. For more information, see the SQL CREATE EXTENSION command in the PostgreSQL documentation.

The following lists the extensions that can be installed by a user who has the CREATE privilege on the current database and do not require the `rds_superuser` role:

- bool_plperl
- btree_gin
- btree_gist
- citext
- cube
- dict_int
- fuzzystrmatch
- hstore
- intarray
- isn
- jsonb_plperl
- ltree
- pg_trgm
- pgcrypto
- plperl
- plpgsql
- pltcl
- tablefunc
- tsm_system_rows
- tsm_system_time
- unaccent
- uuid-ossp

For lists of PostgreSQL extensions and versions that are supported by each available RDS for PostgreSQL version, see PostgreSQL extensions supported on Amazon RDS in *Amazon RDS for PostgreSQL Release Notes*.

Working with PostgreSQL features supported by Amazon RDS for PostgreSQL

Amazon RDS for PostgreSQL supports many of the most common PostgreSQL features. For example, PostgreSQL has an autovacuum feature that performs routine maintenance on the database. The autovacuum feature is active by default. Although you can turn off this feature, we highly recommend that you keep it on. Understanding this feature and what you can do to make sure it works as it should is a basic task of any DBA. For more information about the autovacuum, see Working with PostgreSQL autovacuum on Amazon RDS for PostgreSQL. To learn more about other common DBA tasks, Common DBA tasks for Amazon RDS for PostgreSQL.

RDS for PostgreSQL also supports extensions that add important functionality to the DB instance. For example, you can use the PostGIS extension to work with spatial data, or use the pg_cron extension to schedule maintenance from within the instance. For more information about PostgreSQL extensions, see Using PostgreSQL extensions with Amazon RDS for PostgreSQL.

Foreign data wrappers are a specific type of extension designed to let your RDS for PostgreSQL DB instance work with other commercial databases or data types. For more information about foreign data wrappers supported by RDS for PostgreSQL, see Working with the supported foreign data wrappers for Amazon RDS for PostgreSQL.

Following, you can find information about some other features supported by RDS for PostgreSQL.

Topics

- Custom data types and enumerations with RDS for PostgreSQL
- Event triggers for RDS for PostgreSQL
- Huge pages for RDS for PostgreSQL
- Performing logical replication for Amazon RDS for PostgreSQL
- Configuring IAM authentication for logical replication connections
- RAM disk for the stats_temp_directory
- Tablespaces for RDS for PostgreSQL
- RDS for PostgreSQL collations for EBCDIC and other mainframe migrations
- Managing logical slot synchronization for RDS for PostgreSQL

Custom data types and enumerations with RDS for PostgreSQL

PostgreSQL supports creating custom data types and working with enumerations. For more information about creating and working with enumerations and other data types, see Enumerated types in the PostgreSQL documentation.

The following is an example of creating a type as an enumeration and then inserting values into a table.

```
CREATE TYPE rainbow AS ENUM ('red', 'orange', 'yellow', 'green', 'blue', 'purple');
CREATE TYPE
CREATE TABLE t1 (colors rainbow);
CREATE TABLE
INSERT INTO t1 VALUES ('red'), ( 'orange');
INSERT 0 2
SELECT * from t1;
colors
--------
red
orange
(2 rows)
postgres=> ALTER TYPE rainbow RENAME VALUE 'red' TO 'crimson';
ALTER TYPE
postgres=> SELECT * from t1;
colors
---------
crimson
orange
(2 rows)
```

Event triggers for RDS for PostgreSQL

All current PostgreSQL versions support event triggers, and so do all available versions of RDS for PostgreSQL. You can use the main user account (default, `postgres`) to create, modify, rename, and delete event triggers. Event triggers are at the DB instance level, so they can apply to all databases on an instance.

For example, the following code creates an event trigger that prints the current user at the end of every data definition language (DDL) command.

```
CREATE OR REPLACE FUNCTION raise_notice_func()
```

```
    RETURNS event_trigger
    LANGUAGE plpgsql AS
$$
BEGIN
    RAISE NOTICE 'In trigger function: %', current_user;
END;
$$;

CREATE EVENT TRIGGER event_trigger_1
    ON ddl_command_end
EXECUTE PROCEDURE raise_notice_func();
```

For more information about PostgreSQL event triggers, see Event triggers in the PostgreSQL documentation.

There are several limitations to using PostgreSQL event triggers on Amazon RDS. These include the following:

- You can't create event triggers on read replicas. You can, however, create event triggers on a read replica source. The event triggers are then copied to the read replica. The event triggers on the read replica don't fire on the read replica when changes are pushed from the source. However, if the read replica is promoted, the existing event triggers fire when database operations occur.
- To perform a major version upgrade to a PostgreSQL DB instance that uses event triggers, make sure to delete the event triggers before you upgrade the instance.

Huge pages for RDS for PostgreSQL

Huge pages are a memory management feature that reduces overhead when a DB instance is working with large contiguous chunks of memory, such as that used by shared buffers. This PostgreSQL feature is supported by all currently available RDS for PostgreSQL versions. You allocate huge pages for your application by using calls to `mmap` or SYSV shared memory. RDS for PostgreSQL supports both 4-KB and 2-MB page sizes.

You can turn huge pages on or off by changing the value of the `huge_pages` parameter. The feature is turned on by default for all the DB instance classes other than micro, small, and medium DB instance classes.

RDS for PostgreSQL uses huge pages based on the available shared memory. If the DB instance can't use huge pages due to shared memory constraints, Amazon RDS prevents the DB instance from starting. In this case, Amazon RDS sets the status of the DB instance to an incompatible

parameters state. If this occurs, you can set the `huge_pages` parameter to `off` to allow Amazon RDS to start the DB instance.

The `shared_buffers` parameter is key to setting the shared memory pool that is required for using huge pages. The default value for the `shared_buffers` parameter uses a database parameters macro. This macro sets a percentage of the total 8 KB pages available for the DB instance's memory. When you use huge pages, those pages are located with the huge pages. Amazon RDS puts a DB instance into an incompatible parameters state if the shared memory parameters are set to require more than 90 percent of the DB instance memory.

To learn more about PostgreSQL memory management, see Resource Consumption in the PostgreSQL documentation.

Performing logical replication for Amazon RDS for PostgreSQL

Starting with version 10.4, RDS for PostgreSQL supports the publication and subscription SQL syntax that was introduced in PostgreSQL 10. To learn more, see Logical replication in the PostgreSQL documentation.

> **Note**
>
> In addition to the native PostgreSQL logical replication feature introduced in PostgreSQL 10, RDS for PostgreSQL also supports the `pglogical` extension. For more information, see Using pglogical to synchronize data across instances.

Following, you can find information about setting up logical replication for an RDS for PostgreSQL DB instance.

Topics

- Understanding logical replication and logical decoding
- Working with logical replication slots
- Replicating table level data using logical replication

Understanding logical replication and logical decoding

RDS for PostgreSQL supports the streaming of write-ahead log (WAL) changes using PostgreSQL's logical replication slots. It also supports using logical decoding. You can set up logical replication

slots on your instance and stream database changes through these slots to a client such as `pg_recvlogical`. You create logical replication slots at the database level, and they support replication connections to a single database.

The most common clients for PostgreSQL logical replication are AWS Database Migration Service or a custom-managed host on an Amazon EC2 instance. The logical replication slot has no information about the receiver of the stream. Also, there's no requirement that the target be a replica database. If you set up a logical replication slot and don't read from the slot, data can be written and quickly fill up your DB instance's storage.

You turn on PostgreSQL logical replication and logical decoding for Amazon RDS with a parameter, a replication connection type, and a security role. The client for logical decoding can be any client that can establish a replication connection to a database on a PostgreSQL DB instance.

To turn on logical decoding for an RDS for PostgreSQL DB instance

1. Make sure that the user account that you're using has these roles:
   - The `rds_superuser` role so you can turn on logical replication
   - The `rds_replication` role to grant permissions to manage logical slots and to stream data using logical slots
2. Set the `rds.logical_replication` static parameter to 1. As part of applying this parameter, also set the parameters `wal_level`, `max_wal_senders`, `max_replication_slots`, and `max_connections`. These parameter changes can increase WAL generation, so set the `rds.logical_replication` parameter only when you are using logical slots.
3. Reboot the DB instance for the static `rds.logical_replication` parameter to take effect.
4. Create a logical replication slot as explained in the next section. This process requires that you specify a decoding plugin. Currently, RDS for PostgreSQL supports the test_decoding and wal2json output plugins that ship with PostgreSQL.

For more information on PostgreSQL logical decoding, see the PostgreSQL documentation.

Working with logical replication slots

You can use SQL commands to work with logical slots. For example, the following command creates a logical slot named `test_slot` using the default PostgreSQL output plugin `test_decoding`.

```
SELECT * FROM pg_create_logical_replication_slot('test_slot', 'test_decoding');
slot_name    | xlog_position
-----------------+---------------
regression_slot | 0/16B1970
(1 row)
```

To list logical slots, use the following command.

```
SELECT * FROM pg_replication_slots;
```

To drop a logical slot, use the following command.

```
SELECT pg_drop_replication_slot('test_slot');
pg_drop_replication_slot
-----------------------
(1 row)
```

For more examples on working with logical replication slots, see Logical decoding examples in the PostgreSQL documentation.

After you create the logical replication slot, you can start streaming. The following example shows how logical decoding is controlled over the streaming replication protocol. This example uses the program pg_recvlogical, which is included in the PostgreSQL distribution. Doing this requires that client authentication is set up to allow replication connections.

```
pg_recvlogical -d postgres --slot test_slot -U postgres
    --host -instance-name.111122223333.aws-region.rds.amazonaws.com
    -f -  --start
```

To see the contents of the `pg_replication_origin_status` view, query the `pg_show_replication_origin_status` function.

```
SELECT * FROM pg_show_replication_origin_status();
local_id | external_id | remote_lsn | local_lsn
----------+-------------+------------+-----------
(0 rows)
```

Replicating table level data using logical replication

You can use logical replication to replicate data from source tables to target tables in RDS for PostgreSQL. Logical replication first performs an initial load of existing data from the source tables and then continues to replicate ongoing changes.

1. **Create the source tables**

   Connect to the source database in your RDS for PostgreSQL DB instance:

```
source=> CREATE TABLE testtab (slno int primary key);
CREATE TABLE
```

2. **Insert data into the source tables**

```
source=> INSERT INTO testtab VALUES (generate_series(1,1000));
INSERT 0 1000
```

3. **Create a publication for source tables**

   - Create a publication for the source tables:

```
source=> CREATE PUBLICATION testpub FOR TABLE testtab;
CREATE PUBLICATION
```

   - Use a SELECT query to verify the details of the publication that was created:

```
source=> SELECT * FROM pg_publication;
  oid   | pubname | pubowner | puballtables | pubinsert | pubupdate | pubdelete |
 pubtruncate | pubviaroot
--------+---------+----------+--------------+-----------+-----------+-----------
+-------------+------------
 115069 | testpub |    16395 | f            | t         | t         | t         |
 t           | f
(1 row)
```

- Verify that the source tables are added to the publication:

```
source=> SELECT * FROM pg_publication_tables;
pubname | schemaname | tablename
---------+------------+-----------
 testpub | public     | testtab
(1 rows)
```

- To replicate all tables in a database, use:

```
CREATE PUBLICATION testpub FOR ALL TABLES;
```

- If the publication is already created for individual table and you need to add new table, you can run below query to add any new tables into the existing publication:

```
ALTER PUBLICATION <publication_name> add table <new_table_name>;
```

4. **Connect to target database and create target tables**

- Connect to the target database in the target DB instance. Create the target tables with the same names as the source tables:

```
target=> CREATE TABLE testtab (slno int primary key);
CREATE TABLE
```

- Make sure that there's no data present in the target tables by running a SELECT query on the target tables:

```
target=> SELECT count(*) FROM testtab;
 count
-------
     0
```

```
(1 row)
```

5. **Create and verify subscription in target database**

   - Create the subscription in the target database:

```
target=> CREATE SUBSCRIPTION testsub
CONNECTION 'host=<source RDS/host endpoint> port=5432 dbname=<source_db_name>
 user=<user> password=<password>'
PUBLICATION testpub;
NOTICE:  Created replication slot "testsub" on publisher
CREATE SUBSCRIPTION
```

   - Use a SELECT query to verify that the subscription is enabled:

```
target=> SELECT oid, subname, subenabled, subslotname, subpublications FROM
 pg_subscription;
  oid  | subname | subenabled | subslotname | subpublications
-------+---------+------------+-------------+-----------------
 16434 | testsub | t          | testsub     | {testpub}
(1 row)
```

   - When the subscription is created, it loads all data from the source tables to the target tables. Run a SELECT query on the target tables to verify that the initial data loads:

```
target=> SELECT count(*) FROM testtab;
 count
-------
  1000
(1 row)
```

6. **Verify replication slot in source database**

   The creation of a subscription in the target database creates a replication slot in the source database. Verify the replication slot details by running the following SELECT query on the source database:

```
source=> SELECT * FROM pg_replication_slots;

slot_name |  plugin  | slot_type | datoid | database | temporary | active |
 active_pid | xmin | catalog_xmin | restart_lsn | confirmed_flush_lsn | wal_status
 | safe_wal_size
----------+----------+-----------+--------+----------+-----------+--------
+------------+------+--------------+-------------+---------------------
+------------+---------------
testsub   | pgoutput | logical   | 115048 | source   | f         | t      |
 846 |      |         6945 | 58/B4000568 | 58/B40005A0         | reserved   |
(1 row)
```

7. **Testing replication**

   - Test whether data changes in the source tables are being replicated to the target tables by inserting rows into the source tables:

     ```
     source=> INSERT INTO testtab VALUES(generate_series(1001,2000));
     INSERT 0 1000

     source=> SELECT count(*) FROM testtab;
      count
     -------
       2000
     (1 row)
     ```

   - Verify the number of rows in the target tables to confirm that new inserts are being replicated:

     ```
     target=> SELECT count(*) FROM testtab;
      count
     -------
       2000
     (1 row)
     ```

8. **Refreshing the subscription after adding tables**

   - When you add new tables to an existing publication, it is mandatory to refresh the subscription for the changes to take effect:

     ```
     ALTER SUBSCRIPTION <subscription_name> REFRESH PUBLICATION;
     ```

   - This command fetches missing table information from the publisher and starts replication for tables that were added to the subscribed-to publications since the subscription was created or last refreshed.

Configuring IAM authentication for logical replication connections

Starting with RDS for PostgreSQL versions 11 and higher, you can use AWS Identity and Access Management (IAM) authentication for replication connections. This feature enhances security by allowing you to manage database access using IAM roles instead of passwords. It works both at the cluster and instance granularity and follows the same security model as standard IAM authentication.

IAM authentication for replication connections is an opt-in feature. To enable it, set the `rds.iam_auth_for_replication` parameter to 1 in your DB cluster or DB parameter group. As this is a dynamic parameter, your DB cluster or instance doesn't need to restart, enabling you to leverage IAM authentication with existing workloads without downtime. Before enabling this feature, you must meet the Prerequisites listed below.

Topics

- Prerequisites
- Enabling IAM authentication for replication connections
- Disabling IAM authentication for replication connections
- Limitations and considerations

Prerequisites

To use IAM authentication for replication connections, you need to meet all of the following requirements:

- Your RDS for PostgreSQL DB instance must be version 11 or later.
- On your publisher RDS for PostgreSQL DB instance:
  - Enable IAM database authentication. For more information, see Enabling and disabling IAM database authentication.
  - Enable logical replication by setting the `rds.logical_replication` parameter to 1.

In logical replication, the publisher is the source RDS for PostgreSQL database that sends data to subscriber database. For more information, see Performing logical replication for Amazon RDS for PostgreSQL.

> **Note**
>
> Both IAM authentication and logical replication must be enabled on your publisher RDS for PostgreSQL DB instance. If either one isn't enabled, you can't use IAM authentication for replication connections.

Enabling IAM authentication for replication connections

Complete the following steps to enable IAM authentication for replication connection.

To enable IAM authentication for replication connections

1. Verify that your RDS for PostgreSQL DB cluster or instance meets all prerequisites for IAM authentication with replication connections. For details, see Prerequisites.
2. Configure the `rds.iam_auth_for_replication` parameter based on your RDS for PostgreSQL setup:
   - For RDS for PostgreSQL DB instances: Modify your DB parameter group.
   - For Multi-AZ clusters: Modify your DB cluster parameter group.

   Set `rds.iam_auth_for_replication` to 1. This is a dynamic parameter that takes effect immediately without requiring a reboot.

> **Note**
>
> Multi-AZ clusters use only DB cluster parameter groups. Individual instance parameter groups cannot be modified in Multi-AZ clusters.

3. Connect to your database and grant the necessary roles to your replication user:

   The following SQL commands grant the necessary roles to enable IAM authentication for replication connections:

```
-- Grant IAM authentication role
GRANT rds_iam TO replication_user_name;

-- Grant replication privileges
ALTER USER replication_user_name WITH REPLICATION;
```

   After you complete these steps, the specified user must use IAM authentication for replication connections.

> **Important**
>
> When you enable the feature, users with both `rds_iam` and `rds_replication` roles must use IAM authentication for replication connections. This applies whether the roles are assigned directly to the user or inherited through other roles.

Disabling IAM authentication for replication connections

You can disable IAM authentication for replication connections by using any of the following methods:

- Set the `rds.iam_auth_for_replication` parameter to 0 in your DB parameter group for DB instances or DB cluster parameter group for Multi-AZ clusters.
- Alternatively, you can disable either of these features on your RDS for PostgreSQL DB cluster or instance:
  - Disable logical replication by setting the `rds.logical_replication` parameter to 0

- Disable IAM authentication

When you disable the feature, replication connections can use database passwords for authentication.

> **Note**
>
> Replication connections for users without the `rds_iam` role can use password authentication even when the feature is enabled.

Limitations and considerations

Consider the following limitations and considerations when using IAM authentication for logical replication connections:

- This feature is available only for RDS for PostgreSQL versions 11 and higher.
- The publisher must support IAM authentication for replication connections.
- The IAM authentication token expires after 15 minutes by default. You might need to refresh long-running replication connections before the token expires.

RAM disk for the stats_temp_directory

You can use the RDS for PostgreSQL parameter `rds.pg_stat_ramdisk_size` to specify the system memory allocated to a RAM disk for storing the PostgreSQL `stats_temp_directory`. The RAM disk parameter is only available in RDS for PostgreSQL version 14 and lower versions.

Under certain workloads, setting this parameter can improve performance and decrease I/O requirements. For more information about the `stats_temp_directory`, see the PostgreSQL documentation..

To set up a RAM disk for your `stats_temp_directory`, set the `rds.pg_stat_ramdisk_size` parameter to an integer literal value in the parameter group used by your DB instance. This parameter denotes MB, so you must use an integer value. Expressions, formulas, and functions aren't valid for the `rds.pg_stat_ramdisk_size` parameter. Be sure to reboot the DB instance so that the change takes effect. For information about setting parameters, see Parameter groups for Amazon RDS.

For example, the following AWS CLI command sets the RAM disk parameter to 256 MB.

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name pg-95-ramdisk-testing \
    --parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
 ApplyMethod=pending-reboot"
```

After you reboot, run the following command to see the status of the `stats_temp_directory`.

```
postgres=> SHOW stats_temp_directory;
```

The command should return the following.

```
stats_temp_directory
--------------------------
/rdsdbramdisk/pg_stat_tmp
(1 row)
```

Tablespaces for RDS for PostgreSQL

RDS for PostgreSQL supports tablespaces for compatibility. Because all storage is on a single logical volume, you can't use tablespaces for I/O splitting or isolation. Our benchmarks and experience indicate that a single logical volume is the best setup for most use cases.

To create and use tablespaces with your RDS for PostgreSQL DB instance requires the `rds_superuser` role. Your RDS for PostgreSQL DB instance's main user account (default name, `postgres`) is a member of this role. For more information, see Understanding PostgreSQL roles and permissions.

If you specify a file name when you create a tablespace, the path prefix is `/rdsdbdata/db/base/tablespace`. The following example places tablespace files in `/rdsdbdata/db/base/tablespace/data`. This example assumes that a `dbadmin` user (role) exists and that it's been granted the `rds_superuser` role needed to work with tablespaces.

```
postgres=> CREATE TABLESPACE act_data
  OWNER dbadmin
  LOCATION '/data';
CREATE TABLESPACE
```

To learn more about PostgreSQL tablespaces, see Tablespaces in the PostgreSQL documentation.

RDS for PostgreSQL collations for EBCDIC and other mainframe migrations

RDS for PostgreSQL versions 10 and higher include ICU version 60.2, which is based on Unicode 10.0 and includes collations from the Unicode Common Locale Data Repository, CLDR 32. These software internationalization libraries ensure that character encodings are presented in a consistent way, regardless of operating system or platform. For more information about Unicode CLDR-32, see the CLDR 32 Release Note on the Unicode CLDR website. You can learn more about the internationalization components for Unicode (ICU) at the ICU Technical Committee (ICU-TC) website. For information about ICU-60, see Download ICU 60.

Starting with version 14.3, RDS for PostgreSQL also includes collations that help with data integration and conversion from EBCDIC-based systems. The extended binary coded decimal interchange code or *EBCDIC* encoding is commonly used by mainframe operating systems. These Amazon RDS-provided collations are narrowly defined to sort only those Unicode characters that directly map to EBCDIC code pages. The characters are sorted in EBCDIC code-point order to allow for data validation after conversion. These collations don't include denormalized forms, nor do they include Unicode characters that don't directly map to a character on the source EBCDIC code page.

The character mappings between EBCDIC code pages and Unicode code points are based on tables published by IBM. The complete set is available from IBM as a compressed file for download. RDS for PostgreSQL used these mappings with tools provided by the ICU to create the collations listed in the tables in this section. The collation names include a language and country as required by the ICU. However, EBCDIC code pages don't specify languages, and some EBCDIC code pages cover multiple countries. That means that the language and country portion of the collation names in the table are arbitrary, and they don't need to match the current locale. In other words, the code page number is the most important part of the collation name in this table. You can use any of the collations listed in the following tables in any RDS for PostgreSQL database.

- Unicode to EBCDIC collations table – Some mainframe data migration tools internally use LATIN1 or LATIN9 to encode and process data. Such tools use round-trip schemes to preserve data integrity and support reverse conversion. The collations in this table can be used by tools that process data using LATIN1 encoding, which doesn't require special handling.
- Unicode to LATIN9 collations table – You can use these collations in any RDS for PostgreSQL database.

In the following table, you find collations available in RDS for PostgreSQL that map EBCDIC code pages to Unicode code points. We recommend that you use the collations in this table for application development that requires sorting based on the ordering of IBM code pages.

PostgreSQL collation name	Description of code-page mapping and sort order
da-DK-cp277-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 277 (per conversion tables) are sorted in IBM CP 277 code point order
de-DE-cp273-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 273 (per conversion tables) are sorted in IBM CP 273 code point order
en-GB-cp285-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 285 (per conversion tables) are sorted in IBM CP 285 code point order
en-US-cp037-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 037 (per conversion tables) are sorted in IBM CP 37 code point order
es-ES-cp284-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 284 (per conversion tables) are sorted in IBM CP 284 code point order
fi-FI-cp278-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 278 (per conversion tables) are sorted in IBM CP 278 code point order
fr-FR-cp297-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 297 (per conversion tables) are sorted in IBM CP 297 code point order
it-IT-cp280-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 280 (per conversion tables) are sorted in IBM CP 280 code point order

PostgreSQL collation name	Description of code-page mapping and sort order
nl-BE-cp500-x-icu	Unicode characters that directly map to IBM EBCDIC Code Page 500 (per conversion tables) are sorted in IBM CP 500 code point order

Amazon RDS provides a set of additional collations that sort Unicode code points that map to LATIN9 characters using the tables published by IBM, in the order of the original code points according to the EBCDIC code page of the source data.

PostgreSQL collation name	Description of code-page mapping and sort order
da-DK-cp1142m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1142 (per conversion tables) are sorted in IBM CP 1142 code point order
de-DE-cp1141m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1141 (per conversion tables) are sorted in IBM CP 1141 code point order
en-GB-cp1146m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1146 (per conversion tables) are sorted in IBM CP 1146 code point order
en-US-cp1140m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1140 (per conversion

PostgreSQL collation name	Description of code-page mapping and sort order
	tables) are sorted in IBM CP 1140 code point order
es-ES-cp1145m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1145 (per conversion tables) are sorted in IBM CP 1145 code point order
fi-FI-cp1143m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1143 (per conversion tables) are sorted in IBM CP 1143 code point order
fr-FR-cp1147m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1147 (per conversion tables) are sorted in IBM CP 1147 code point order
it-IT-cp1144m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1144 (per conversion tables) are sorted in IBM CP 1144 code point order
nl-BE-cp1148m-x-icu	Unicode characters that map to LATIN9 characters originally converted from IBM EBCDIC Code Page 1148 (per conversion tables) are sorted in IBM CP 1148 code point order

In the following, you can find an example of using an RDS for PostgreSQL collation.

```
db1=> SELECT pg_import_system_collations('pg_catalog');
 pg_import_system_collations
-----------------------------
                          36
db1=> SELECT '¤' < 'a' col1;
 col1
------
 t
db1=> SELECT '¤' < 'a' COLLATE "da-DK-cp277-x-icu" col1;
 col1
------
 f
```

We recommend that you use the collations in the Unicode to EBCDIC collations table and in the Unicode to LATIN9 collations table for application development that requires sorting based on the ordering of IBM code pages. The following collations (suffixed with the letter “b”) are also visible in `pg_collation`, but are intended for use by mainframe data integration and migration tools at AWS that map code pages with specific code point shifts and require special handling in collation. In other words, the following collations aren't recommended for use.

- da-DK-277b-x-icu
- da-DK-1142b-x-icu
- de-DE-cp273b-x-icu
- de-DE-cp1141b-x-icu
- en-GB-cp1146b-x-icu
- en-GB-cp285b-x-icu
- en-US-cp037b-x-icu
- en-US-cp1140b-x-icu
- es-ES-cp1145b-x-icu
- es-ES-cp284b-x-icu
- fi-FI-cp1143b-x-icu
- fr-FR-cp1147b-x-icu
- fr-FR-cp297b-x-icu
- it-IT-cp1144b-x-icu
- it-IT-cp280b-x-icu

- nl-BE-cp1148b-x-icu
- nl-BE-cp500b-x-icu

To learn more about migrating applications from mainframe environments to AWS, see What is AWS Mainframe Modernization?.

For more information about managing collations in PostgreSQL, see Collation Support in the PostgreSQL documentation.

Managing logical slot synchronization for RDS for PostgreSQL

Starting in community PostgreSQL 17, a new feature to automatically synchronize logical replication slots from primary to standby servers has been introduced through the parameter `sync_replication_slots` or the related function `pg_sync_replication_slots()`, which manually synchronizes slots on execution.

These features are available starting with RDS for PostgreSQL 17. A typical setup will have a primary instance and its read replica, as well as a logical replication subscriber to the primary.

Ensure the subscription is created with the failover option set to true:

```
CREATE SUBSCRIPTION subname CONNECTION 'host=...' PUBLICATION pubname WITH (failover =
 true);
```

This creates a logical slot on the publisher with failover enabled.

```
postgres=> SELECT slot_name, slot_type, failover FROM pg_catalog.pg_replication_slots;
 slot_name | slot_type | failover
-----------+-----------+----------
 subname   | logical   | t
(1 row)
```

By enabling slot synchronization, all of the failover logical replication slots on the primary are automatically created on the physical standbys and are synced periodically. Ensure the following values have been set through parameter groups:

- `rds.logical_replication` must be 1 to enable logical replication
- `hot_standby_feedback` must be 1 on the standby
- `rds.logical_slot_sync_dbname` on the standby must be set to a valid database name

The parameter's default value is `postgres`. If the logical publishing instance has the `postgres` database, the default parameter does not need to be changed.

- `synchronized_standby_slots` on the primary must be set to the physical replication slot of the standby intended to be in-sync
- `sync_replication_slots` must be 1 to enable automatic synchronization

With a failover-enabled subscription slot and the above parameter values, when a standby is promoted, the subscriber can alter its subscription to this newly promoted instance and continue logical replication seamlessly.

Connecting to a DB instance running the PostgreSQL database engine

After Amazon RDS provisions your DB instance, you can use any standard SQL client application to connect to the instance. Before you can connect, the DB instance must be available and accessible. Whether you can connect to the instance from outside the VPC depends on how you created the Amazon RDS DB instance:

- If you created your DB instance as *public*, devices and Amazon EC2 instances outside the VPC can connect to your database.
- If you created your DB instance as *private*, only Amazon EC2 instances and devices inside the Amazon VPC can connect to your database.

To check whether your DB instance is public or private, use the AWS Management Console to view the **Connectivity & security** tab for your instance. Under **Security**, you can find the "Publicly accessible" value, with No for private, Yes for public.

To learn more about different Amazon RDS and Amazon VPC configurations and how they affect accessibility, see Scenarios for accessing a DB instance in a VPC.

Contents

- Installing the psql client
- Finding the connection information for an RDS for PostgreSQL DB instance
- Using pgAdmin to connect to a RDS for PostgreSQL DB instance
- Using psql to connect to your RDS for PostgreSQL DB instance
- Connecting to RDS for PostgreSQL with the Amazon Web Services (AWS) JDBC Driver
- Connecting to RDS for PostgreSQL with the Amazon Web Services (AWS) Python Driver
- Troubleshooting connections to your RDS for PostgreSQL instance
  - Error – FATAL: database name does not exist
  - Error – Could not connect to server: Connection timed out
  - Errors with security group access rules

Installing the psql client

To connect to your DB instance from an EC2 instance, you can install a PostgreSQL client on the EC2 instance. To install the latest version of the psql client on Amazon Linux 2023, run the following command:

```
sudo dnf install postgresql<version number>
```

To install the latest version of the psql client on Amazon Linux 2, run the following command:

```
sudo yum install -y postgresql
```

To install the latest version of the psql client on Ubuntu, run the following command:

```
sudo apt install -y postgresql-client
```

Finding the connection information for an RDS for PostgreSQL DB instance

If the DB instance is available and accessible, you can connect by providing the following information to the SQL client application:

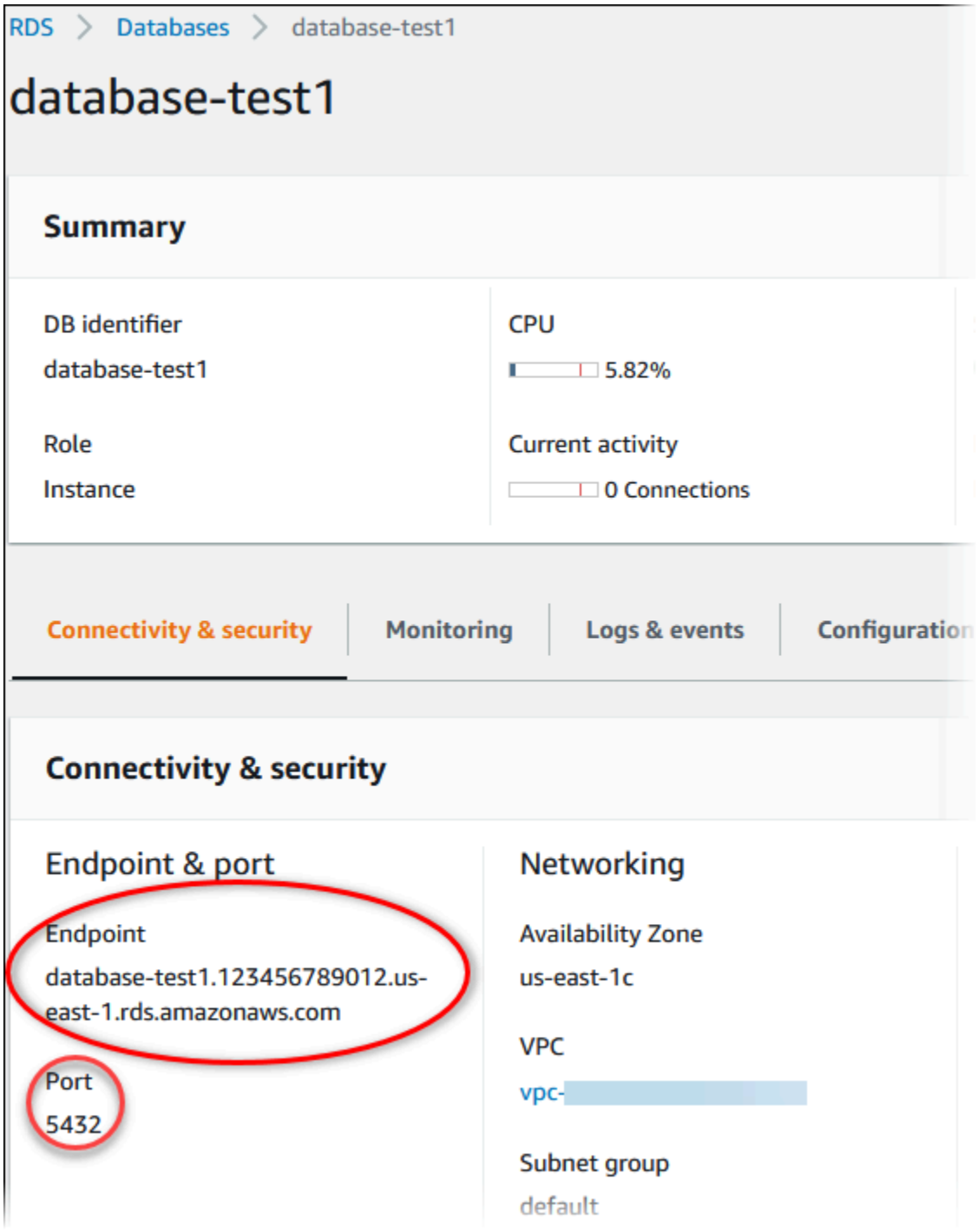
- The DB instance endpoint, which serves as the host name (DNS name) for the instance.
- The port on which the DB instance is listening. For PostgreSQL, the default port is 5432.
- The user name and password for the DB instance. The default 'master username' for PostgreSQL is `postgres`.
- The name and password of the database (DB name).

You can obtain these details by using the AWS Management Console, the AWS CLI describe-db-instances command, or the Amazon RDS API DescribeDBInstances operation.

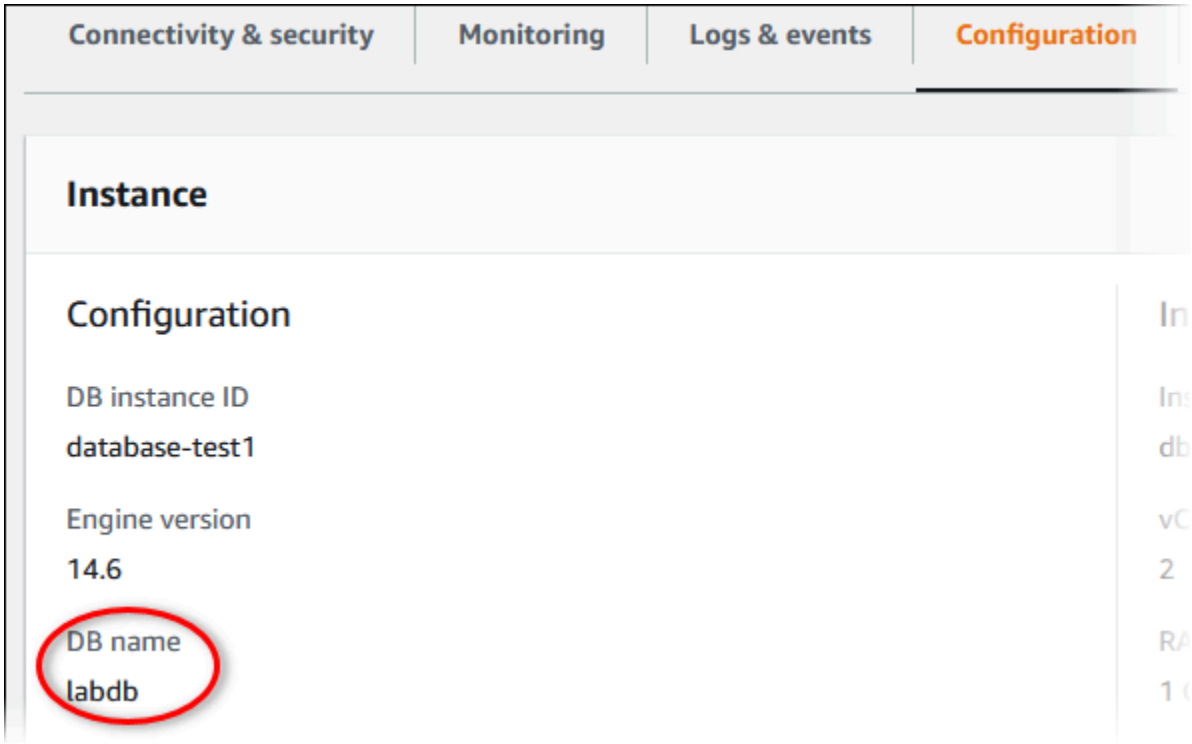
To find the endpoint, port number, and DB name using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Open the RDS console and then choose **Databases** to display a list of your DB instances.

3. Choose the PostgreSQL DB instance name to display its details.
4. On the **Connectivity & security** tab, copy the endpoint. Also, note the port number. You need both the endpoint and the port number to connect to the DB instance.



5. On the **Configuration** tab, note the DB name. If you created a database when you created the RDS for PostgreSQL instance, you see the name listed under DB name. If you didn't create a database, the DB name displays a dash (-).



Following are two ways to connect to a PostgreSQL DB instance. The first example uses pgAdmin, a popular open-source administration and development tool for PostgreSQL. The second example uses psql, a command line utility that is part of a PostgreSQL installation.

Using pgAdmin to connect to a RDS for PostgreSQL DB instance

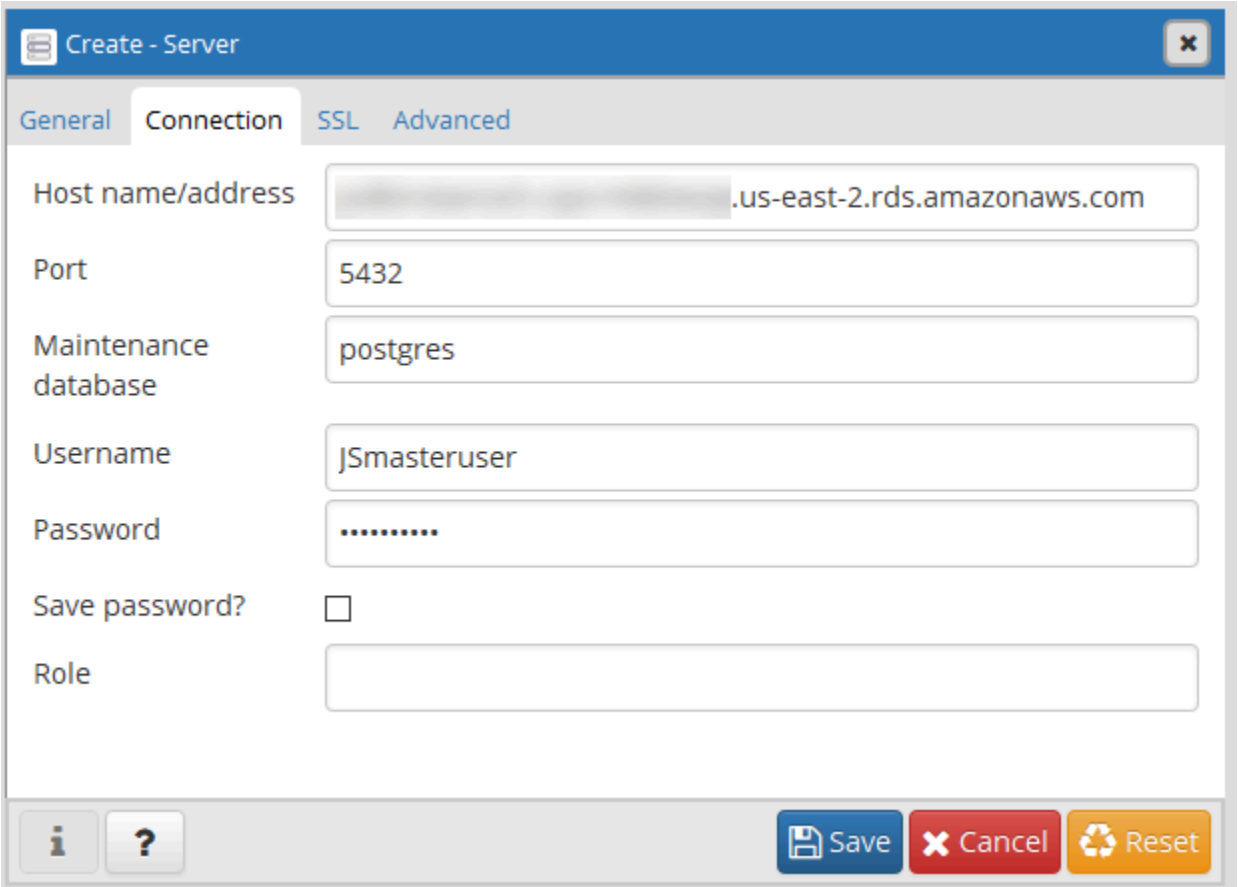
You can use the open-source tool pgAdmin to connect to your RDS for PostgreSQL DB instance. You can download and install pgAdmin from http://www.pgadmin.org/ without having a local instance of PostgreSQL on your client computer.

To connect to your RDS for PostgreSQL DB instance using pgAdmin

1. Launch the pgAdmin application on your client computer.
2. On the **Dashboard** tab, choose **Add New Server**.
3. In the **Create - Server** dialog box, type a name on the **General** tab to identify the server in pgAdmin.
4. On the **Connection** tab, type the following information from your DB instance:

   - For **Host**, type the endpoint, for example `mypostgresql.c6c8dntfzzhgv0.us-east-2.rds.amazonaws.com`.

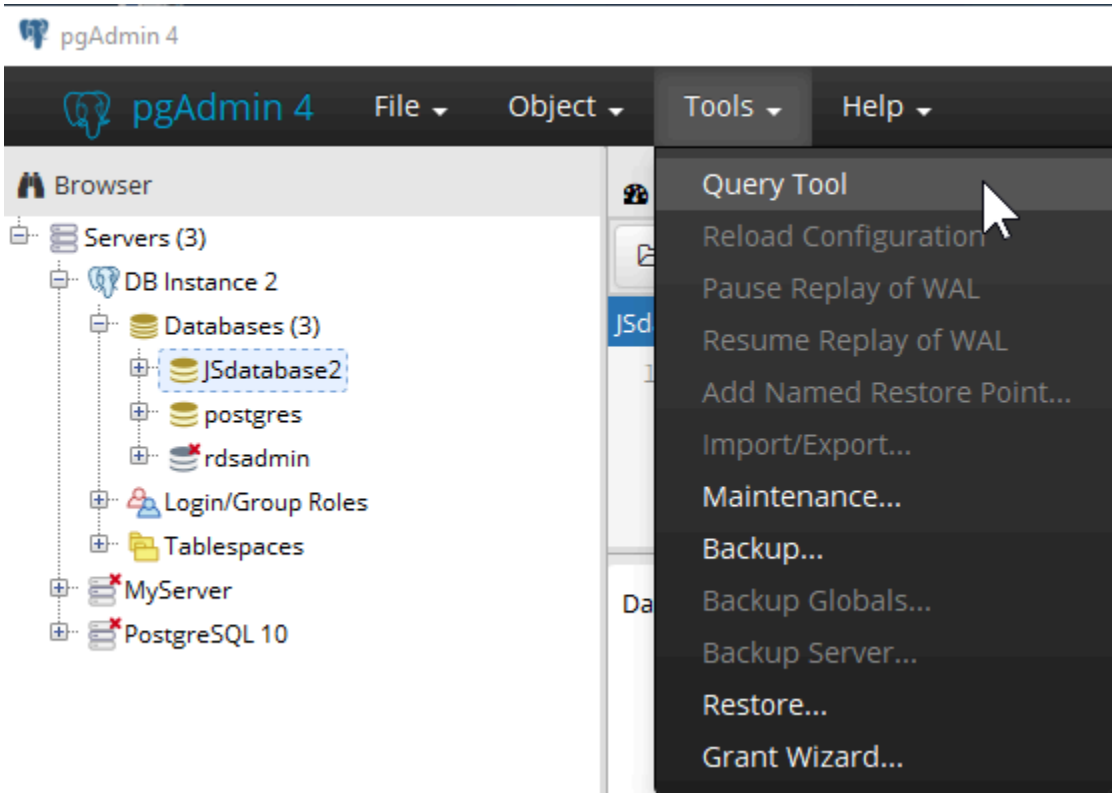
- For **Port**, type the assigned port.
- For **Username**, type the user name that you entered when you created the DB instance (if you changed the 'master username' from the default, `postgres`).
- For **Password**, type the password that you entered when you created the DB instance.



5. Choose **Save**.

   If you have any problems connecting, see [Troubleshooting connections to your RDS for PostgreSQL instance].

6. To access a database in the pgAdmin browser, expand **Servers**, the DB instance, and **Databases**. Choose the DB instance's database name.



7. To open a panel where you can enter SQL commands, choose **Tools**, **Query Tool**.

Using psql to connect to your RDS for PostgreSQL DB instance

You can use a local instance of the psql command line utility to connect to a RDS for PostgreSQL DB instance. You need either PostgreSQL or the psql client installed on your client computer.

You can download the PostgreSQL client from the PostgreSQL website. Follow the instructions specific to your operating system version to install psql.

To connect to your RDS for PostgreSQL DB instance using psql, you need to provide host (DNS) information, access credentials, and the name of the database.

Use one of the following formats to connect to your RDS for PostgreSQL DB instance. When you connect, you're prompted for a password. For batch jobs or scripts, use the `--no-password` option. This option is set for the entire session.

> **Note**
>
> A connection attempt with `--no-password` fails when the server requires password authentication and a password is not available from other sources. For more information, see the psql documentation.

If this is the first time you are connecting to this DB instance, or if you didn't yet create a database for this RDS for PostgreSQL instance, you can connect to the **postgres** database using the 'master username' and password.

For Unix, use the following format.

```
psql \
   --host=<DB instance endpoint> \
   --port=<port> \
   --username=<master username> \
   --password \
   --dbname=<database name>
```

For Windows, use the following format.

```
psql ^
   --host=<DB instance endpoint> ^
   --port=<port> ^
   --username=<master username> ^
   --password ^
   --dbname=<database name>
```

For example, the following command connects to a database called `mypgdb` on a PostgreSQL DB instance called `mypostgresql` using fictitious credentials.

```
psql --host=mypostgresql.c6c8mwvfdgv0.us-west-2.rds.amazonaws.com --port=5432 --
username=awsuser --password --dbname=mypgdb
```

Connecting to RDS for PostgreSQL with the Amazon Web Services (AWS) JDBC Driver

The Amazon Web Services (AWS) JDBC Driver is designed as an advanced JDBC wrapper. This wrapper is complementary to and extends the functionality of an existing JDBC driver. The driver is drop-in compatible with the community pgJDBC driver.

To install the AWS JDBC Driver, append the AWS JDBC Driver .jar file (located in the application CLASSPATH), and keep references to the respective community driver. Update the respective connection URL prefix as follows:

- `jdbc:postgresql://` to `jdbc:aws-wrapper:postgresql://`

For more information about the AWS JDBC Driver and complete instructions for using it, see the Amazon Web Services (AWS) JDBC Driver GitHub repository.

Connecting to RDS for PostgreSQL with the Amazon Web Services (AWS) Python Driver

The Amazon Web Services (AWS) Python Driver is designed as an advanced Python wrapper. This wrapper is complementary to and extends the functionality of the open-source Psycopg driver. The AWS Python Driver supports Python versions 3.8 and higher. You can install the `aws-advanced-python-wrapper` package using the `pip` command, along with the `psycopg` open-source packages.

For more information about the AWS Python Driver and complete instructions for using it, see the Amazon Web Services (AWS) Python Driver GitHub repository.

Troubleshooting connections to your RDS for PostgreSQL instance

Topics

- Error – FATAL: database name does not exist
- Error – Could not connect to server: Connection timed out
- Errors with security group access rules

Error – FATAL: database *name* does not exist

If when trying to connect you receive an error like `FATAL: database name does not exist`, try using the default database name **postgres** for the `--dbname` option.

Error – Could not connect to server: Connection timed out

If you can't connect to the DB instance, the most common error is `Could not connect to server: Connection timed out.` If you receive this error, check the following:

- Check that the host name used is the DB instance endpoint and that the port number used is correct.
- Make sure that the DB instance's public accessibility is set to **Yes** to allow external connections. To modify the **Public access** setting, see Modifying an Amazon RDS DB instance.
- Make sure that the user connecting to the database has CONNECT access to it. You can use the following query to provide connect access to the database.

```
GRANT CONNECT ON DATABASE database name TO username;
```

- Check that the security group assigned to the DB instance has rules to allow access through any firewall your connection might go through. For example, if the DB instance was created using the default port of 5432, your company might have firewall rules blocking connections to that port from external company devices.

  To fix this, modify the DB instance to use a different port. Also, make sure that the security group applied to the DB instance allows connections to the new port. To modify the **Database port** setting, see Modifying an Amazon RDS DB instance.
- Check whether the port you're attempting to use is already occupied by a local instance of PostgreSQL or another service running on your computer. For example, if you have a local PostgreSQL database running on the same port (default is 5432), it might prevent a successful connection to the RDS for PostgreSQL DB instance. Make sure that the port is free, or try connecting with a different port number if possible.
- See also Errors with security group access rules.

Errors with security group access rules

By far the most common connection problem is with the security group's access rules assigned to the DB instance. If you used the default security group when you created the DB instance, the security group likely didn't have access rules that allow you to access the instance.

For the connection to work, the security group you assigned to the DB instance at its creation must allow access to the DB instance. For example, if the DB instance was created in a VPC, it must have a VPC security group that authorizes connections. Check if the DB instance was created using a security group that doesn't authorize connections from the device or Amazon EC2 instance where the application is running.

You can add or edit an inbound rule in the security group. For **Source**, choosing **My IP** allows access to the DB instance from the IP address detected in your browser. For more information, see Provide access to your DB instance in your VPC by creating a security group.

Alternatively, if the DB instance was created outside of a VPC, it must have a database security group that authorizes those connections.

For more information about Amazon RDS security groups, see Controlling access with security groups.

Securing connections to RDS for PostgreSQL with SSL/TLS

RDS for PostgreSQL supports Secure Socket Layer (SSL) encryption for PostgreSQL DB instances. Using SSL, you can encrypt a PostgreSQL connection between your applications and your PostgreSQL DB instances. You can also force all connections to your PostgreSQL DB instance to use SSL. RDS for PostgreSQL also supports Transport Layer Security (TLS), the successor protocol to SSL.

To learn more about Amazon RDS and data protection, including encrypting connections using SSL/TLS, see Data protection in Amazon RDS.

Topics

- Using SSL with a PostgreSQL DB instance
- Updating applications to connect to PostgreSQL DB instances using new SSL/TLS certificates

Using SSL with a PostgreSQL DB instance

Amazon RDS supports Secure Socket Layer (SSL) encryption for PostgreSQL DB instances. Using SSL, you can encrypt a PostgreSQL connection between your applications and your PostgreSQL DB instances. By default, RDS for PostgreSQL uses and expects all clients to connect using SSL/TLS, but you can also require it. RDS for PostgreSQL supports Transport Layer Security (TLS) versions 1.1, 1.2, and 1.3.

For general information about SSL support and PostgreSQL databases, see SSL support in the PostgreSQL documentation. For information about using an SSL connection over JDBC, see Configuring the client in the PostgreSQL documentation.

SSL support is available in all AWS Regions for PostgreSQL. Amazon RDS creates an SSL certificate for your PostgreSQL DB instance when the instance is created. If you enable SSL certificate verification, then the SSL certificate includes the DB instance endpoint as the Common Name (CN) for the SSL certificate to guard against spoofing attacks.

Topics

- Connecting to a PostgreSQL DB instance over SSL
- Requiring an SSL connection to a PostgreSQL DB instance
- Determining the SSL connection status
- SSL cipher suites in RDS for PostgreSQL

Connecting to a PostgreSQL DB instance over SSL

To connect to a PostgreSQL DB instance over SSL

1. Download the certificate.

   For information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
2. Connect to your PostgreSQL DB instance over SSL.

   When you connect using SSL, your client can choose whether to verify the certificate chain. If your connection parameters specify `sslmode=verify-ca` or `sslmode=verify-full`, then your client requires the RDS CA certificates to be in their trust store or referenced in the connection URL. This requirement is to verify the certificate chain that signs your database certificate.

   When a client, such as psql or JDBC, is configured with SSL support, the client first tries to connect to the database with SSL by default. If the client can't connect with SSL, it reverts to connecting without SSL. The default `sslmode` mode used is different between libpq-based clients (such as psql) and JDBC. The libpq-based and JDBC clients default to `prefer`.

   Use the `sslrootcert` parameter to reference the certificate, for example `sslrootcert=rds-ssl-ca-cert.pem`.

The following is an example of using `psql` to connect to a PostgreSQL DB instance using SSL with certificate verification.

```
$ psql "host=db-name.555555555555.ap-southeast-1.rds.amazonaws.com
    port=5432 dbname=testDB user=testuser sslrootcert=rds-ca-rsa2048-g1.pem
 sslmode=verify-full"
```

Requiring an SSL connection to a PostgreSQL DB instance

You can require that connections to your PostgreSQL DB instance use SSL by using the `rds.force_ssl` parameter. The `rds.force_ssl` parameter default value is 1 (on) for RDS for PostgreSQL version 15 and later. For all other RDS for PostgreSQL major versions 14 and older, the default value of this parameter is 0 (off). You can set the `rds.force_ssl` parameter to 1 (on) to require SSL/TLS for connections to your DB cluster. You can set the `rds.force_ssl` parameter to 1 (on) to require SSL for connections to your DB instance.

To change the value of this parameter, you need to create a custom DB parameter group. You then change the value for `rds.force_ssl` in your custom DB parameter group to 1 to turn on this feature. If you prepare the custom DB parameter group before creating your RDS for PostgreSQL DB instance you can choose it (instead of a default parameter group) during the creation process. If you do this after your RDS for PostgreSQL DB instance is already running, you need to reboot the instance so that your instance uses the custom parameter group. For more information, see Parameter groups for Amazon RDS.

When the `rds.force_ssl` feature is active on your DB instance, connection attempts that aren't using SSL are rejected with the following message:

```
$ psql -h db-name.555555555555.ap-southeast-1.rds.amazonaws.com port=5432 dbname=testDB
 user=testuser
psql: error: FATAL: no pg_hba.conf entry for host "w.x.y.z", user "testuser", database
 "testDB", SSL off
```

Determining the SSL connection status

The encrypted status of your connection is shown in the logon banner when you connect to the DB instance:

```
Password for user master:
psql (10.3)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.
postgres=>
```

You can also load the `sslinfo` extension and then call the `ssl_is_used()` function to determine if SSL is being used. The function returns `t` if the connection is using SSL, otherwise it returns `f`.

```
postgres=> CREATE EXTENSION sslinfo;
CREATE EXTENSION
postgres=> SELECT ssl_is_used();
ssl_is_used
---------
t
(1 row)
```

For more detailed information, you can use the following query to get information from `pg_settings`:

```
SELECT name as "Parameter name", setting as value, short_desc FROM pg_settings WHERE
 name LIKE '%ssl%';
             Parameter name             |                  value                  |
                  short_desc
----------------------------------------+-----------------------------------------
+-------------------------------------------------
 ssl                                    | on                                      |
 Enables SSL connections.
 ssl_ca_file                            | /rdsdbdata/rds-metadata/ca-cert.pem     |
 Location of the SSL certificate authority file.
 ssl_cert_file                          | /rdsdbdata/rds-metadata/server-cert.pem |
 Location of the SSL server certificate file.
 ssl_ciphers                            | HIGH:!aNULL:!3DES                       |
 Sets the list of allowed SSL ciphers.
 ssl_crl_file                           |                                         |
 Location of the SSL certificate revocation list file.
 ssl_dh_params_file                     |                                         |
 Location of the SSL DH parameters file.
 ssl_ecdh_curve                         | prime256v1                              |
 Sets the curve to use for ECDH.
 ssl_key_file                           | /rdsdbdata/rds-metadata/server-key.pem  |
 Location of the SSL server private key file.
 ssl_library                            | OpenSSL                                 |
 Name of the SSL library.
 ssl_max_protocol_version               |                                         |
 Sets the maximum SSL/TLS protocol version to use.
 ssl_min_protocol_version               | TLSv1.2                                 |
 Sets the minimum SSL/TLS protocol version to use.
 ssl_passphrase_command                 |                                         |
 Command to obtain passphrases for SSL.
 ssl_passphrase_command_supports_reload | off                                     |
 Also use ssl_passphrase_command during server reload.
 ssl_prefer_server_ciphers              | on                                      |
 Give priority to server ciphersuite order.
(14 rows)
```

You can also collect all the information about your RDS for PostgreSQL DB instance's SSL usage by process, client, and application by using the following query:

```
SELECT datname as "Database name", usename as "User name", ssl, client_addr,
 application_name, backend_type
   FROM pg_stat_ssl
   JOIN pg_stat_activity
```

```
   ON pg_stat_ssl.pid = pg_stat_activity.pid
   ORDER BY ssl;
 Database name | User name | ssl |  client_addr   |    application_name    |
 backend_type
---------------+-----------+-----+----------------+------------------------
+-----------------------------
               |           | f   |                |                        | autovacuum
 launcher
               | rdsadmin  | f   |                |                        | logical
 replication launcher
               |           | f   |                |                        | background
 writer
               |           | f   |                |                        |
 checkpointer
               |           | f   |                |                        | walwriter
 rdsadmin      | rdsadmin  | t   | 127.0.0.1      |                        | client
 backend
 rdsadmin      | rdsadmin  | t   | 127.0.0.1      | PostgreSQL JDBC Driver | client
 backend
 postgres      | postgres  | t   | 204.246.162.36 | psql                   | client
 backend
(8 rows)
```

To identify the cipher used for your SSL connection, you can query as follows:

```
postgres=> SELECT ssl_cipher();
ssl_cipher
--------------------
DHE-RSA-AES256-SHA
(1 row)
```

To learn more about the `sslmode` option, see Database connection control functions in the *PostgreSQL documentation*.

SSL cipher suites in RDS for PostgreSQL

The PostgreSQL configuration parameter ssl_ciphers specifies the categories of cipher suites that are allowed for SSL connections to the database when using TLS 1.2 and lower.

In RDS for PostgreSQL 16 and later, you can modify the `ssl_ciphers` parameter to use specific values from the allowlisted cipher suites. This is a dynamic parameter that doesn't require a

database instance reboot. To view the allowlisted cipher suites, use either the Amazon RDS console or the following AWS CLI command:

```
aws rds describe-db-parameters --db-parameter-group-name <your-parameter-group> --
region <region> --endpoint-url <endpoint-url> --output json | jq '.Parameters[] |
 select(.ParameterName == "ssl_ciphers")'
```

The following table lists both the default cipher suites and the allowed cipher suites for versions that support custom configurations.

PostgreSQL engine version	Default ssl_cipher suite values	Allowlisted custom ssl_cipher suite values
18	HIGH:!aNULL:!3DES	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
17	HIGH:!aNULL:!3DES	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

PostgreSQL engine version	Default ssl_cipher suite values	Allowlisted custom ssl_cipher suite values
		TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
16	HIGH:!aNULL:!3DES	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
15	HIGH:!aNULL:!3DES	Custom ssl_ciphers isn't supported
14	HIGH:!aNULL:!3DES	Custom ssl_ciphers isn't supported
13	HIGH:!aNULL:!3DES	Custom ssl_ciphers isn't supported
12	HIGH:!aNULL:!3DES	Custom ssl_ciphers isn't supported

PostgreSQL engine version	Default ssl_cipher suite values	Allowlisted custom ssl_cipher suite values
11.4 and higher minor versions	`HIGH:MEDIUM:+3DES:!aNULL:!RC4`	Custom ssl_ciphers isn't supported
11.1, 11.2	`HIGH:MEDIUM:+3DES:!aNULL`	Custom ssl_ciphers isn't supported
10.9 and higher minor versions	`HIGH:MEDIUM:+3DES:!aNULL:!RC4`	Custom ssl_ciphers isn't supported
10.7 and lower minor versions	`HIGH:MEDIUM:+3DES:!aNULL`	Custom ssl_ciphers isn't supported

To configure all instance connections to use the `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` cipher suite, modify your parameter group as shown in the following example:

```
aws rds modify-db-parameter-group --db-parameter-group-name <your-parameter-group> --
parameters
 "ParameterName='ssl_ciphers',ParameterValue='TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384',ApplyMet
```

This example uses an ECDSA cipher, which requires your instance to use a certificate authority with elliptic curve cryptography (ECC) to establish a connection. For information about certificate authorities provided by Amazon RDS, see Certificate authorities.

You can verify the ciphers in use through the methods described in Determining the SSL connection status.

Ciphers may have different names depending on the context:

- The allowlisted ciphers that you can configure in your parameter group are referred to with their IANA names.
- The `sslinfo` and `psql` logon banner refer to ciphers using their OpenSSL names.

By default, the value of `ssl_max_protocol_version` in RDS for PostgreSQL 16 and later is TLS v1.3. You must set the value of this parameter to TLS v1.2 as TLS v1.3 doesn't use the cipher

configurations specified in the `ssl_ciphers` parameter. When you set the value as TLS v1.2, connections use only the ciphers that you define in `ssl_ciphers`.

```
aws rds modify-db-parameter-group --db-parameter-group-name <your-parameter-group> --
parameters
 "ParameterName='ssl_max_protocol_version',ParameterValue='TLSv1.2',ApplyMethod=immediate"
```

To ensure database connections use SSL, set the `rds.force_ssl parameter` to 1 in your parameter group. For more information about parameters and parameter groups, see Parameter groups for Amazon RDS.

Updating applications to connect to PostgreSQL DB instances using new SSL/TLS certificates

Certificates used for Secure Socket Layer or Transport Layer Security (SSL/TLS) typically have a set lifetime. When service providers update their Certificate Authority (CA) certificates, clients must update their applications to use the new certificates. Following, you can find information about how to determine if your client applications use SSL/TLS to connect to your Amazon RDS for PostgreSQL DB instance. You also find information about how to check if those applications verify the server certificate when they connect.

> **Note**
>
> A client application that's configured to verify the server certificate before SSL/TLS connection must have a valid CA certificate in the client's trust store. Update the client trust store when necessary for new certificates.

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB instances. We strongly recommend testing these procedures in a nonproduction environment before implementing them in your production environments.

For more information about certificate rotation, see Rotating your SSL/TLS certificate. For more information about downloading certificates, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . For information about using SSL/TLS with PostgreSQL DB instances, see Using SSL with a PostgreSQL DB instance.

Topics

- Determining whether applications are connecting to PostgreSQL DB instances using SSL
- Determining whether a client requires certificate verification in order to connect
- Updating your application trust store
- Using SSL/TLS connections for different types of applications

Determining whether applications are connecting to PostgreSQL DB instances using SSL

Check the DB instance configuration for the value of the `rds.force_ssl` parameter. By default, the `rds.force_ssl` parameter is set to `0` (off) for DB instances using PostgreSQL versions before version 15. By default, `rds.force_ssl` is set to 1 (on) for DB instances using PostgreSQL version 15 and later major versions. If the `rds.force_ssl` parameter is set to 1 (on), clients are required to use SSL/TLS for connections. For more information about parameter groups, see Parameter groups for Amazon RDS.

If you are using RDS PostgreSQL version 9.5 or later major version and `rds.force_ssl` is not set to 1 (on), query the `pg_stat_ssl` view to check connections using SSL. For example, the following query returns only SSL connections and information about the clients using SSL.

```
SELECT datname, usename, ssl, client_addr
  FROM pg_stat_ssl INNER JOIN pg_stat_activity ON pg_stat_ssl.pid =
 pg_stat_activity.pid
  WHERE ssl is true and usename<>'rdsadmin';
```

Only rows using SSL/TLS connections are displayed with information about the connection. The following is sample output.

```
 datname  | usename | ssl | client_addr
----------+---------+-----+-------------
 benchdb  | pgadmin | t   | 53.95.6.13
 postgres | pgadmin | t   | 53.95.6.13
(2 rows)
```

This query displays only the current connections at the time of the query. The absence of results doesn't indicate that no applications are using SSL connections. Other SSL connections might be established at a different time.

Determining whether a client requires certificate verification in order to connect

When a client, such as psql or JDBC, is configured with SSL support, the client first tries to connect to the database with SSL by default. If the client can't connect with SSL, it reverts to connecting without SSL. The default `sslmode` mode used for both libpq-based clients (such as psql) and JDBC is set to `prefer`. The certificate on the server is verified only when `sslrootcert` is provided with `sslmode` set to `verify-ca` or `verify-full`. An error is thrown if the certificate is invalid.

Use PGSSLROOTCERT to verify the certificate with the PGSSLMODE environment variable, with PGSSLMODE set to `verify-ca` or `verify-full`.

```
PGSSLMODE=verify-full PGSSLROOTCERT=/fullpath/ssl-cert.pem psql -h
 pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com -U masteruser -d postgres
```

Use the `sslrootcert` argument to verify the certificate with `sslmode` in connection string format, with `sslmode` set to `verify-ca` or `verify-full` to verify the certificate.

```
psql "host=pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=verify-full
 sslrootcert=/full/path/ssl-cert.pem user=masteruser dbname=postgres"
```

For example, in the preceding case, if you are using an invalid root certificate, then you see an error similar to the following on your client.

```
psql: SSL error: certificate verify failed
```

Updating your application trust store

For information about updating the trust store for PostgreSQL applications, see Secure TCP/IP connections with SSL in the PostgreSQL documentation.

For information about downloading the root certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

For sample scripts that import certificates, see Sample script for importing certificates into your trust store.

> **Note**
>
> When you update the trust store, you can retain older certificates in addition to adding the new certificates.

Using SSL/TLS connections for different types of applications

The following provides information about using SSL/TLS connections for different types of applications:

- **psql**

  The client is invoked from the command line by specifying options either as a connection string or as environment variables. For SSL/TLS connections, the relevant options are `sslmode` (environment variable `PGSSLMODE`), `sslrootcert` (environment variable `PGSSLROOTCERT`).

  For the complete list of options, see Parameter key words in the PostgreSQL documentation. For the complete list of environment variables, see Environment variables in the PostgreSQL documentation.

- **pgAdmin**

  This browser-based client is a more user-friendly interface for connecting to a PostgreSQL database.

  For information about configuring connections, see the pgAdmin documentation.

- **JDBC**

  JDBC enables database connections with Java applications.

  For general information about connecting to a PostgreSQL database with JDBC, see Connecting to the database in the PostgreSQL JDBC driver documentation. For information about connecting with SSL/TLS, see Configuring the client in the PostgreSQL JDBC driver documentation.

- **Python**

  A popular Python library for connecting to PostgreSQL databases is `psycopg2`.

For information about using psycopg2, see the psycopg2 documentation. For a short tutorial on how to connect to a PostgreSQL database, see Psycopg2 tutorial. You can find information about the options the connect command accepts in The psycopg2 module content.

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-rsa2048-g1 certificates. For instructions, see step 3 in Updating your CA certificate by modifying your DB instance or cluster.

Using Kerberos authentication with Amazon RDS for PostgreSQL

You can use Kerberos to authenticate users when they connect to your DB instance running PostgreSQL. To do so, configure your DB instance to use AWS Directory Service for Microsoft Active Directory for Kerberos authentication. AWS Directory Service for Microsoft Active Directory is also called AWS Managed Microsoft AD. It's a feature available with Directory Service. To learn more, see What is Directory Service? in the *AWS Directory Service Administration Guide*.

To start, create an AWS Managed Microsoft AD directory to store user credentials. Then, provide to your PostgreSQL DB instance the Active Directory's domain and other information. When users authenticate with the PostgreSQL DB instance, authentication requests are forwarded to the AWS Managed Microsoft AD directory.

Keeping all of your credentials in the same directory can save you time and effort. You have a centralized location for storing and managing credentials for multiple DB instances. Using a directory can also improve your overall security profile.

In addition, you can access credentials from your own on-premises Microsoft Active Directory. To do so, create a trusting domain relationship so that the AWS Managed Microsoft AD directory trusts your on-premises Microsoft Active Directory. In this way, your users can access your PostgreSQL instances with the same Windows single sign-on (SSO) experience as when they access workloads in your on-premises network.

A database can use password authentication or password authentication with either Kerberos or AWS Identity and Access Management (IAM) authentication. For more information about IAM authentication, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.

> **Note**
>
> RDS for PostgreSQL doesn't support Kerberos authentication for Active Directory groups.

Topics

- Region and version availability
- Overview of Kerberos authentication for PostgreSQL DB instances
- Setting up Kerberos authentication for PostgreSQL DB instances

- Managing an RDS for PostgreSQL DB instance in an Active Directory domain
- Connecting to PostgreSQL with Kerberos authentication

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability of RDS for PostgreSQL with Kerberos authentication, see Supported Regions and DB engines for Kerberos authentication in Amazon RDS.

Overview of Kerberos authentication for PostgreSQL DB instances

To set up Kerberos authentication for a PostgreSQL DB instance, take the following steps, described in more detail later:

1. Use AWS Managed Microsoft AD to create an AWS Managed Microsoft AD directory. You can use the AWS Management Console, the AWS CLI, or the Directory Service API to create the directory. Make sure to open the relevant outbound ports on the directory security group so that the directory can communicate with the instance.
2. Create a role that provides Amazon RDS access to make calls to your AWS Managed Microsoft AD directory. To do so, create an AWS Identity and Access Management (IAM) role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

   For the IAM role to allow access, the AWS Security Token Service (AWS STS) endpoint must be activated in the correct AWS Region for your AWS account. AWS STS endpoints are active by default in all AWS Regions, and you can use them without any further actions. For more information, see Activating and deactivating AWS STS in an AWS Region in the *IAM User Guide*.
3. Create and configure users in the AWS Managed Microsoft AD directory using the Microsoft Active Directory tools. For more information about creating users in your Active Directory, see Manage users and groups in AWS Managed Microsoft AD in the *Directory Service Administration Guide*.
4. If you plan to locate the directory and the DB instance in different AWS accounts or virtual private clouds (VPCs), configure VPC peering. For more information, see What is VPC peering? in the *Amazon VPC Peering Guide*.
5. Create or modify a PostgreSQL DB instance either from the console, CLI, or RDS API using one of the following methods:

- Creating an Amazon RDS DB instance
- Modifying an Amazon RDS DB instance
- Restoring to a DB instance
- Restoring a DB instance to a specified time for Amazon RDS

You can locate the instance in the same Amazon Virtual Private Cloud (VPC) as the directory or in a different AWS account or VPC. When you create or modify the PostgreSQL DB instance, do the following:

- Provide the domain identifier (`d-*` identifier) that was generated when you created your directory.
- Provide the name of the IAM role that you created.
- Ensure that the DB instance security group can receive inbound traffic from the directory security group.

6. Use the RDS master user credentials to connect to the PostgreSQL DB instance. Create the user in PostgreSQL to be identified externally. Externally identified users can log in to the PostgreSQL DB instance using Kerberos authentication.

Setting up Kerberos authentication for PostgreSQL DB instances

You use AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) to set up Kerberos authentication for a PostgreSQL DB instance. To set up Kerberos authentication, take the following steps.

Topics

- Step 1: Create a directory using AWS Managed Microsoft AD
- Step 2: (Optional) Create a trust relationship between your on-premises Active Directory and Directory Service
- Step 3: Create an IAM role for Amazon RDS to access the Directory Service
- Step 4: Create and configure users
- Step 5: Enable cross-VPC traffic between the directory and the DB instance
- Step 6: Create or modify a PostgreSQL DB instance
- Step 7: Create PostgreSQL users for your Kerberos principals
- Step 8: Configure a PostgreSQL client

Step 1: Create a directory using AWS Managed Microsoft AD

Directory Service creates a fully managed Active Directory in the AWS Cloud. When you create an AWS Managed Microsoft AD directory, Directory Service creates two domain controllers and DNS servers for you. The directory servers are created in different subnets in a VPC. This redundancy helps make sure that your directory remains accessible even if a failure occurs.

When you create an AWS Managed Microsoft AD directory, AWS Directory Service performs the following tasks on your behalf:

- Sets up an Active Directory within your VPC.
- Creates a directory administrator account with the user name `Admin` and the specified password. You use this account to manage your directory.

  > **Important**
  >
  > Make sure to save this password. Directory Service doesn't store this password, and it can't be retrieved or reset.

- Creates a security group for the directory controllers. The security group must permit communication with the PostgreSQL DB instance.

When you launch AWS Directory Service for Microsoft Active Directory, AWS creates an Organizational Unit (OU) that contains all of your directory's objects. This OU, which has the NetBIOS name that you entered when you created your directory, is located in the domain root. The domain root is owned and managed by AWS.

The `Admin` account that was created with your AWS Managed Microsoft AD directory has permissions for the most common administrative activities for your OU:

- Create, update, or delete users
- Add resources to your domain such as file or print servers, and then assign permissions for those resources to users in your OU
- Create additional OUs and containers
- Delegate authority
- Restore deleted objects from the Active Directory Recycle Bin

- Run Active Directory and Domain Name Service (DNS) modules for Windows PowerShell on the Active Directory Web Service

The `Admin` account also has rights to perform the following domain-wide activities:

- Manage DNS configurations (add, remove, or update records, zones, and forwarders)
- View DNS event logs
- View security event logs

To create a directory with AWS Managed Microsoft AD

1. In the Directory Service console navigation pane, choose **Directories**, and then choose **Set up directory**.
2. Choose **AWS Managed Microsoft AD**. AWS Managed Microsoft AD is the only option currently supported for use with Amazon RDS.
3. Choose **Next**.
4. On the **Enter directory information** page, provide the following information:

   Edition

   Choose the edition that meets your requirements.

   Directory DNS name

   The fully qualified name for the directory, such as **`corp.example.com`**.

   Directory NetBIOS name

   An optional short name for the directory, such as `CORP`.

   Directory description

   An optional description for the directory.

   Admin password

   The password for the directory administrator. The directory creation process creates an administrator account with the user name `Admin` and this password.

The directory administrator password can't include the word "admin." The password is case-sensitive and must be 8–64 characters in length. It must also contain at least one character from three of the following four categories:

- Lowercase letters (a–z)
- Uppercase letters (A–Z)
- Numbers (0–9)
- Nonalphanumeric characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Confirm password

Retype the administrator password.

> **Important**
>
> Make sure that you save this password. Directory Service doesn't store this password, and it can't be retrieved or reset.

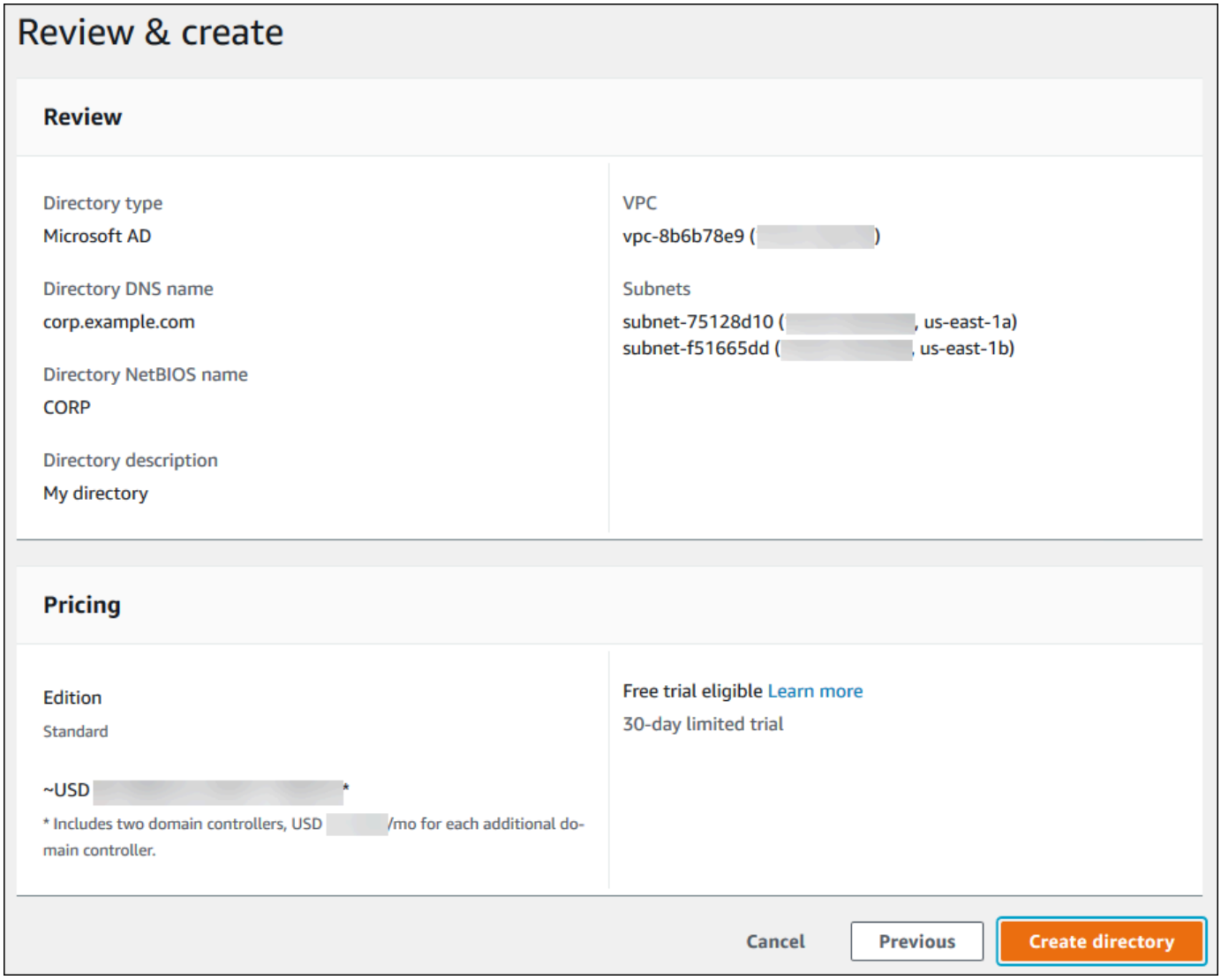
5. Choose **Next**.
6. On the **Choose VPC and subnets** page, provide the following information:

   VPC

   Choose the VPC for the directory. You can create the PostgreSQL DB instance in this same VPC or in a different VPC.

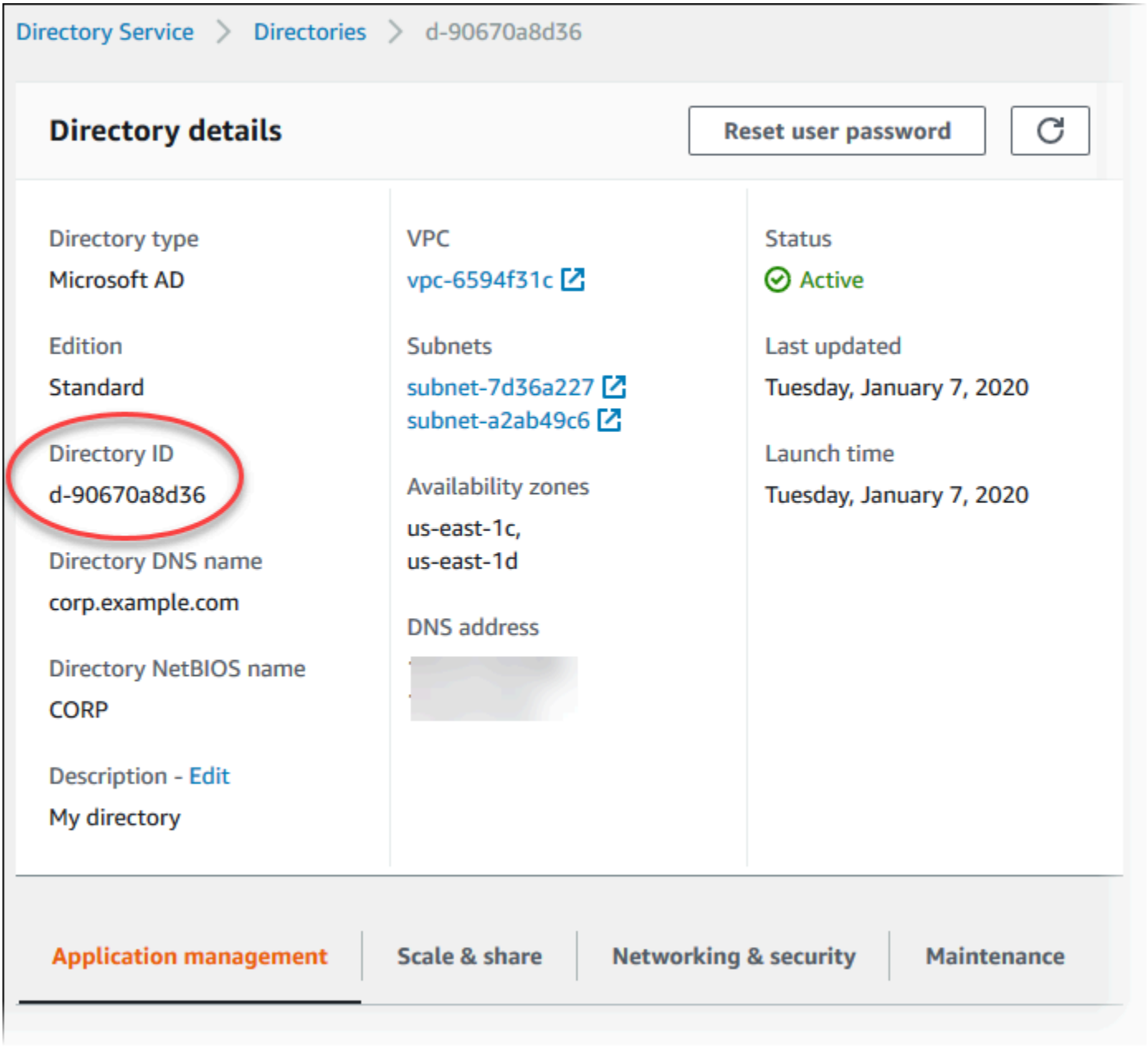
   Subnets

   Choose the subnets for the directory servers. The two subnets must be in different Availability Zones.
7. Choose **Next**.
8. Review the directory information. If changes are needed, choose **Previous** and make the changes. When the information is correct, choose **Create directory**.



It takes several minutes for the directory to be created. When it has been successfully created, the **Status** value changes to **Active**.

To see information about your directory, choose the directory ID in the directory listing. Make a note of the **Directory ID** value. You need this value when you create or modify your PostgreSQL DB instance.



Step 2: (Optional) Create a trust relationship between your on-premises Active Directory and Directory Service

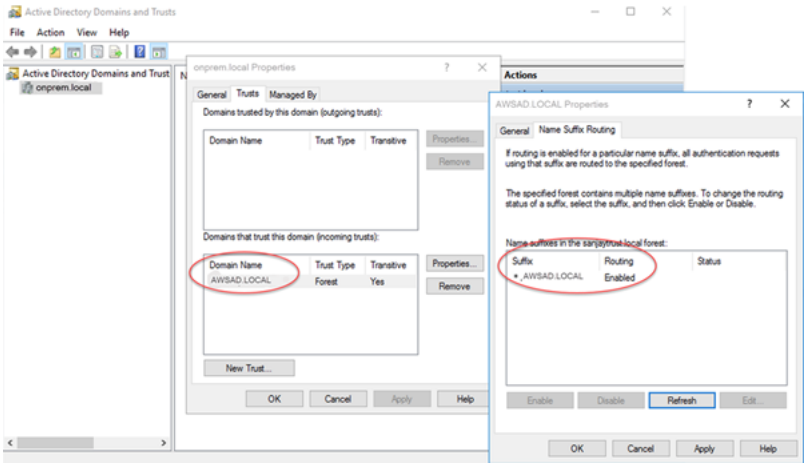
If you don't plan to use your own on-premises Microsoft Active Directory, skip to Step 3: Create an IAM role for Amazon RDS to access the Directory Service.

To get Kerberos authentication using your on-premises Active Directory, you need to create a trusting domain relationship using a forest trust between your on-premises Microsoft Active Directory and the AWS Managed Microsoft AD directory (created in Step 1: Create a directory using AWS Managed Microsoft AD). The trust can be one-way, where the AWS Managed Microsoft

AD directory trusts the on-premises Microsoft Active Directory. The trust can also be two-way, where both Active Directories trust each other. For more information about setting up trusts using Directory Service, see When to create a trust relationship in the *AWS Directory Service Administration Guide*.

> **Note**
>
> If you use an on-premises Microsoft Active Directory, Windows clients connect using the domain name of the Directory Service in the endpoint rather than rds.amazonaws.com. To learn more, see Connecting to PostgreSQL with Kerberos authentication.

Make sure that your on-premises Microsoft Active Directory domain name includes a DNS suffix routing that corresponds to the newly created trust relationship. The following screenshot shows an example.



Step 3: Create an IAM role for Amazon RDS to access the Directory Service

For Amazon RDS to call Directory Service for you, your AWS account needs an IAM role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`. This role allows Amazon RDS to make calls to Directory Service.

When you create a DB instance using the AWS Management Console and your console user account has the `iam:CreateRole` permission, the console creates the needed IAM role automatically. In this case, the role name is `rds-directoryservice-kerberos-access-role`. Otherwise, you must create the IAM role manually. When you create this IAM role, choose `Directory Service`, and attach the AWS managed policy `AmazonRDSDirectoryServiceAccess` to it.

For more information about creating IAM roles for a service, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Note**
>
> The IAM role used for Windows Authentication for RDS for Microsoft SQL Server can't be used for Amazon RDS for PostgreSQL.

As an alternative to using the `AmazonRDSDirectoryServiceAccess` managed policy, you can create policies with the required permissions. In this case, the IAM role must have the following IAM trust policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The role must also have the following IAM role policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
```

```
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

For opt-in AWS Regions, use Region-specific service principals in IAM role trust policies. When you create a trust policy for services in these Regions, specify the Region code in the service principal.

The following example shows a trust policy that includes Region-specific service principals:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.REGION-CODE.amazonaws.com",
          "rds.REGION-CODE.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Replace REGION-CODE with the code for your specific Region. For example, use the following service principals for the Asia Pacific (Melbourne) Region:

```
"Service": [
  "directoryservice.rds.ap-southeast-4.amazonaws.com",
  "rds.ap-southeast-4.amazonaws.com"
]
```

Step 4: Create and configure users

You can create users by using the Active Directory Users and Computers tool. This is one of the Active Directory Domain Services and Active Directory Lightweight Directory Services tools. For more information, see Add Users and Computers to the Active Directory domain in the Microsoft documentation. In this case, users are individuals or other entities, such as their computers that are part of the domain and whose identities are being maintained in the directory.

To create users in an Directory Service directory, you must be connected to a Windows-based Amazon EC2 instance that's a member of the Directory Service directory. At the same time, you must be logged in as a user that has privileges to create users. For more information, see Create a user in the *AWS Directory Service Administration Guide*.

Step 5: Enable cross-VPC traffic between the directory and the DB instance

If you plan to locate the directory and the DB instance in the same VPC, skip this step and move on to Step 6: Create or modify a PostgreSQL DB instance.

If you plan to locate the directory and the DB instance in different VPCs, configure cross-VPC traffic using VPC peering or AWS Transit Gateway.

The following procedure enables traffic between VPCs using VPC peering. Follow the instructions in What is VPC peering? in the *Amazon Virtual Private Cloud Peering Guide*.

To enable cross-VPC traffic using VPC peering

1. Set up appropriate VPC routing rules to ensure that network traffic can flow both ways.
2. Ensure that the DB instance security group can receive inbound traffic from the directory security group.
3. Ensure that there is no network access control list (ACL) rule to block traffic.

If a different AWS account owns the directory, you must share the directory.

To share the directory between AWS accounts

1. Start sharing the directory with the AWS account that the DB instance will be created in by following the instructions in Tutorial: Sharing your AWS Managed Microsoft AD directory for seamless EC2 Domain-join in the *Directory Service Administration Guide*.
2. Sign in to the Directory Service console using the account for the DB instance, and ensure that the domain has the SHARED status before proceeding.
3. While signed into the Directory Service console using the account for the DB instance, note the **Directory ID** value. You use this directory ID to join the DB instance to the domain.

Step 6: Create or modify a PostgreSQL DB instance

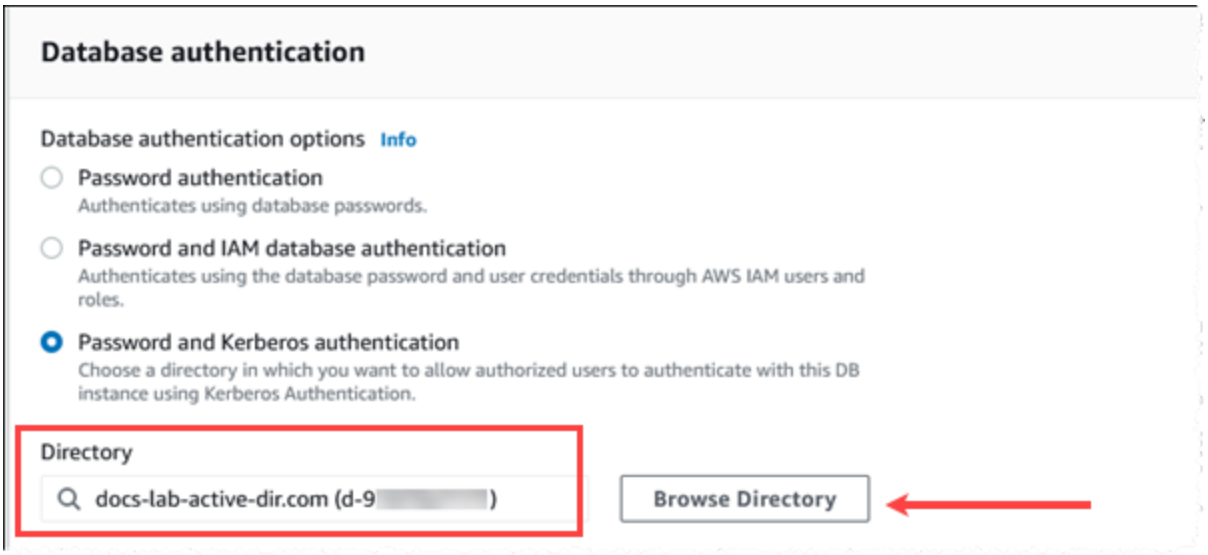
Create or modify a PostgreSQL DB instance for use with your directory. You can use the console, CLI, or RDS API to associate a DB instance with a directory. You can do this in one of the following ways:

- Create a new PostgreSQL DB instance using the console, the create-db-instance CLI command, or the CreateDBInstance RDS API operation. For instructions, see Creating an Amazon RDS DB instance.
- Modify an existing PostgreSQL DB instance using the console, the modify-db-instance CLI command, or the ModifyDBInstance RDS API operation. For instructions, see Modifying an Amazon RDS DB instance.
- Restore a PostgreSQL DB instance from a DB snapshot using the console, the restore-db-instance-from-db-snapshot CLI command, or the RestoreDBInstanceFromDBSnapshot RDS API operation. For instructions, see Restoring to a DB instance.
- Restore a PostgreSQL DB instance to a point-in-time using the console, the restore-db-instance-to-point-in-time CLI command, or the RestoreDBInstanceToPointInTime RDS API operation. For instructions, see Restoring a DB instance to a specified time for Amazon RDS.

Kerberos authentication is only supported for PostgreSQL DB instances in a VPC. The DB instance can be in the same VPC as the directory, or in a different VPC. The DB instance must use a security group that allows ingress and egress within the directory's VPC so the DB instance can communicate with the directory.

Console

When you use the console to create, modify, or restore a DB instance, choose **Password and Kerberos authentication** in the **Database authentication** section. Then choose **Browse Directory**. Select the directory or choose **Create a new directory** to use the Directory Service.



AWS CLI

When you use the AWS CLI, the following parameters are required for the DB instance to be able to use the directory that you created:

- For the `--domain` parameter, use the domain identifier ("d-*" identifier) generated when you created the directory.
- For the `--domain-iam-role-name` parameter, use the role you created that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

For example, the following CLI command modifies a DB instance to use a directory.

```
aws rds modify-db-instance --db-instance-identifier mydbinstance --domain d-Directory-ID --domain-iam-role-name role-name
```

> **Important**
>
> If you modify a DB instance to enable Kerberos authentication, reboot the DB instance after making the change.

Step 7: Create PostgreSQL users for your Kerberos principals

At this point, your RDS for PostgreSQL DB instance is joined to the AWS Managed Microsoft AD domain. The users that you created in the directory in Step 4: Create and configure users need to be set up as PostgreSQL database users and granted privileges to login to the database. You do that by signing in as the database user with `rds_superuser` privileges. For example, if you accepted the defaults when you created your RDS for PostgreSQL DB instance, you use `postgres`, as shown in the following steps.

To create PostgreSQL database users for Kerberos principals

1. Use `psql` to connect to your RDS for PostgreSQL DB instance endpoint using `psql`. The following example uses the default `postgres` account for the `rds_superuser` role.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. Create a database user name for each Kerberos principal (Active Directory username) that you want to have access to the database. Use the canonical username (identity) as defined in the Active Directory instance, that is, a lower-case `alias` (username in Active Directory) and the upper-case name of the Active Directory domain for that user name. The Active Directory user name is an externally authenticated user, so use quotes around the name as shown following.

```
postgres=> CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;
CREATE ROLE
```

3. Grant the `rds_ad` role to the database user.

```
postgres=> GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
GRANT ROLE
```

After you finish creating all the PostgreSQL users for your Active Directory user identities, users can access the RDS for PostgreSQL DB instance by using their Kerberos credentials.

It's required that the database users who authenticate using Kerberos are doing so from client machines that are members of the Active Directory domain.

Database users that have been granted the `rds_ad` role can't also have the `rds_iam` role. This also applies to nested memberships. For more information, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.

Step 8: Configure a PostgreSQL client

To configure a PostgreSQL client, take the following steps:

- Create a krb5.conf file (or equivalent) to point to the domain.
- Verify that traffic can flow between the client host and Directory Service. Use a network utility such as Netcat for the following:
  - Verify traffic over DNS for port 53.
  - Verify traffic over TCP/UDP for port 53 and for Kerberos, which includes ports 88 and 464 for Directory Service.
- Verify that traffic can flow between the client host and the DB instance over the database port. For example, use psql to connect and access the database.

The following is sample krb5.conf content for AWS Managed Microsoft AD.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

The following is sample krb5.conf content for an on-premises Microsoft Active Directory.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
```

```
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

Managing an RDS for PostgreSQL DB instance in an Active Directory domain

You can use the console, the CLI, or the RDS API to manage your DB instance and its relationship with your Microsoft Active Directory. For example, you can associate an Active Directory to enable Kerberos authentication. You can also remove the association for an Active Directory to disable Kerberos authentication. You can also move a DB instance to be externally authenticated by one Microsoft Active Directory to another.

For example, using the CLI, you can do the following:

- To reattempt enabling Kerberos authentication for a failed membership, use the modify-db-instance CLI command. Specify the current membership's directory ID for the `--domain` option.
- To disable Kerberos authentication on a DB instance, use the modify-db-instance CLI command. Specify `none` for the `--domain` option.
- To move a DB instance from one domain to another, use the modify-db-instance CLI command. Specify the domain identifier of the new domain for the `--domain` option.

Understanding Domain membership

After you create or modify your DB instance, it becomes a member of the domain. You can view the status of the domain membership in the console or by running the describe-db-instances CLI command. The status of the DB instance can be one of the following:

- `kerberos-enabled` – The DB instance has Kerberos authentication enabled.
- `enabling-kerberos` – AWS is in the process of enabling Kerberos authentication on this DB instance.

- `pending-enable-kerberos` – Enabling Kerberos authentication is pending on this DB instance.
- `pending-maintenance-enable-kerberos` – AWS will attempt to enable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `pending-disable-kerberos` – Disabling Kerberos authentication is pending on this DB instance.
- `pending-maintenance-disable-kerberos` – AWS will attempt to disable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- `enable-kerberos-failed` – A configuration problem prevented AWS from enabling Kerberos authentication on the DB instance. Correct the configuration problem before reissuing the command to modify the DB instance.
- `disabling-kerberos` – AWS is in the process of disabling Kerberos authentication on this DB instance.

A request to enable Kerberos authentication can fail because of a network connectivity issue or an incorrect IAM role. In some cases, the attempt to enable Kerberos authentication might fail when you create or modify a DB instance. If so, make sure that you are using the correct IAM role, then modify the DB instance to join the domain.

> **Note**
>
> Only Kerberos authentication with RDS for PostgreSQL sends traffic to the domain's DNS servers. All other DNS requests are treated as outbound network access on your DB instances running PostgreSQL. For more information about outbound network access with RDS for PostgreSQL, see Using a custom DNS server for outbound network access.

Connecting to PostgreSQL with Kerberos authentication

You can connect to PostgreSQL with Kerberos authentication with the pgAdmin interface or with a command-line interface such as psql. For more information about connecting, see Connecting to a DB instance running the PostgreSQL database engine . For information about obtaining the endpoint, port number, and other details needed for connection, see Connect to a PostgreSQL DB instance.

> **Note**
>
> GSSAPI authentication and encryption in PostgreSQL are implemented by the Kerberos library `libkrb5.so`. Features such as `postgres_fdw` and `dblink` also rely on this same library for outbound connections with Kerberos authentication or encryption.

pgAdmin

To use pgAdmin to connect to PostgreSQL with Kerberos authentication, take the following steps:

1. Launch the pgAdmin application on your client computer.
2. On the **Dashboard** tab, choose **Add New Server**.
3. In the **Create - Server** dialog box, enter a name on the **General** tab to identify the server in pgAdmin.
4. On the **Connection** tab, enter the following information from your RDS for PostgreSQL database.
   - For **Host**, enter the endpoint for the RDS for PostgreSQL DB instance. An endpoint looks similar to the following:

     ```
     RDS-DB-instance.111122223333.aws-region.rds.amazonaws.com
     ```

     To connect to an on-premises Microsoft Active Directory from a Windows client, you use the domain name of the AWS Managed Active Directory instead of `rds.amazonaws.com` in the host endpoint. For example, suppose that the domain name for the AWS Managed Active Directory is `corp.example.com`. Then for **Host**, the endpoint would be specified as follows:

     ```
     RDS-DB-instance.111122223333.aws-region.corp.example.com
     ```

   - For **Port**, enter the assigned port.
   - For **Maintenance database**, enter the name of the initial database to which the client will connect.
   - For **Username**, enter the user name that you entered for Kerberos authentication in Step 7: Create PostgreSQL users for your Kerberos principals.
5. Choose **Save**.

Psql

To use psql to connect to PostgreSQL with Kerberos authentication, take the following steps:

1. At a command prompt, run the following command.

   ```
   kinit username
   ```

   Replace *username* with the user name. At the prompt, enter the password stored in the Microsoft Active Directory for the user.

2. If the PostgreSQL DB instance is using a publicly accessible VPC, put IP address for your DB instance endpoint in your `/etc/hosts` file on the EC2 client. For example, the following commands obtain the IP address and then put it in the `/etc/hosts` file.

   ```
   % dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
   ;; Truncated, retrying in TCP mode.
   ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
   34.210.197.118

   % echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/
   hosts
   ```

   If you're using an on-premises Microsoft Active Directory from a Windows client, then you need to connect using a specialized endpoint. Instead of using the Amazon domain `rds.amazonaws.com` in the host endpoint, use the domain name of the AWS Managed Active Directory.

   For example, suppose that the domain name for your AWS Managed Active Directory is `corp.example.com`. Then use the format *PostgreSQL-endpoint.AWS-Region.corp.example.com* for the endpoint and put it in the `/etc/hosts` file.

   ```
   % echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/
   hosts
   ```

3. Use the following psql command to log in to a PostgreSQL DB instance that is integrated with Active Directory.

   ```
   psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
   Region.rds.amazonaws.com postgres
   ```

To log in to the PostgreSQL DB cluster from a Windows client using an on-premises Active Directory, use the following psql command with the domain name from the previous step (`corp.example.com`):

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.corp.example.com postgres
```

Using a custom DNS server for outbound network access

RDS for PostgreSQL supports outbound network access on your DB instances and allows Domain Name Service (DNS) resolution from a custom DNS server owned by the customer. You can resolve only fully qualified domain names from your RDS for PostgreSQL DB instance through your custom DNS server.

Topics

- Turning on custom DNS resolution
- Turning off custom DNS resolution
- Setting up a custom DNS server

Turning on custom DNS resolution

To turn on DNS resolution in your customer VPC, first associate a custom DB parameter group to your RDS for PostgreSQL instance. Then turn on the `rds.custom_dns_resolution` parameter by setting it to 1, and then restart the DB instance for the changes to take place.

Turning off custom DNS resolution

To turn off DNS resolution in your customer VPC, first turn off the `rds.custom_dns_resolution` parameter of your custom DB parameter group by setting it to 0. Then restart the DB instance for the changes to take place.

Setting up a custom DNS server

After you set up your custom DNS name server, it takes up to 30 minutes to propagate the changes to your DB instance. After the changes are propagated to your DB instance, all outbound network traffic requiring a DNS lookup queries your DNS server over port 53.

> **Note**
>
> If you don't set up a custom DNS server and `rds.custom_dns_resolution` is set to 1, hosts are resolved using an Amazon Route 53 private zone. For more information, see Working with private hosted zones.

To set up a custom DNS server for your RDS for PostgreSQL DB instance

1. From the Dynamic Host Configuration Protocol (DHCP) options set attached to your VPC, set the `domain-name-servers` option to the IP address of your DNS name server. For more information, see DHCP options sets.

   > **Note**
   >
   > The `domain-name-servers` option accepts up to four values, but your Amazon RDS DB instance uses only the first value.

2. Ensure that your DNS server can resolve all lookup queries, including public DNS names, Amazon EC2 private DNS names, and customer-specific DNS names. If the outbound network traffic contains any DNS lookups that your DNS server can't handle, your DNS server must have appropriate upstream DNS providers configured.
3. Configure your DNS server to produce User Datagram Protocol (UDP) responses of 512 bytes or less.
4. Configure your DNS server to produce Transmission Control Protocol (TCP) responses of 1,024 bytes or less.
5. Configure your DNS server to allow inbound traffic from your Amazon RDS DB instances over port 53. If your DNS server is in an Amazon VPC, the VPC must have a security group that contains inbound rules that allow UDP and TCP traffic on port 53. If your DNS server is not in an Amazon VPC, it must have appropriate firewall settings to allow UDP and TCP inbound traffic on port 53.

   For more information, see Security groups for your VPC and Adding and removing rules.
6. Configure the VPC of your Amazon RDS DB instance to allow outbound traffic over port 53. Your VPC must have a security group that contains outbound rules that allow UDP and TCP traffic on port 53.

   For more information, see Security groups for your VPC and Adding and removing rules in the *Amazon VPC User Guide*.
7. Make sure that the routing path between the Amazon RDS DB instance and the DNS server is configured correctly to allow DNS traffic.

Also, if the Amazon RDS DB instance and the DNS server are not in the same VPC, make sure that a peering connection is set up between them. For more information, see What is VPC peering? in *Amazon VPC Peering Guide*.

Upgrades of the RDS for PostgreSQL DB engine

There are two types of upgrades that you can manage for your PostgreSQL database:

- Operating system updates – Occasionally, Amazon RDS might need to update the underlying operating system of your database to apply security fixes or OS changes. You can decide when Amazon RDS applies OS updates by using the RDS console, AWS Command Line Interface (AWS CLI), or RDS API. For more information about OS updates, see Applying updates to a DB instance .
- Database engine upgrades – When Amazon RDS supports a new version of a database engine, you can upgrade your databases to the new version.

A *database* in this context is an RDS for PostgreSQL DB instance or Multi-AZ DB cluster.

There are two kinds of engine upgrades for PostgreSQL databases: major version upgrades and minor version upgrades.

Major version upgrades

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, you must manually perform major version upgrades of your databases. You can initiate a major version upgrade by modifying your DB instance or Multi-AZ DB cluster. Before you perform a major version upgrade, we recommend that you follow the steps described in Choosing a major version for an RDS for PostgreSQL upgrade.

Amazon RDS handles Multi-AZ major version upgrades in the following ways:

- **Multi-AZ DB instance deployment** – Amazon RDS simultaneously upgrades the primary and any standby instances. Your database might not be available for several minutes while the upgrade completes.
- **Multi-AZ DB cluster deployment** – Amazon RDS simultaneously upgrades the reader and writer instances. Your database might not be available for several minutes while the upgrade completes.

If you upgrade a DB instance that has in-Region read replicas, Amazon RDS upgrades the replicas along with the primary DB instance.

Amazon RDS doesn't upgrade Multi-AZ DB cluster read replicas. If you perform a major version upgrade of a Multi-AZ DB cluster, then the replication state of its read replicas changes to

terminated. You must manually delete and recreate the read replicas after the upgrade completes.

> **Tip**
>
> You can minimize the downtime required for a major version upgrade by using a blue/green deployment. For more information, see *Using Blue/Green Deployments for database updates*.

Minor version upgrades

In contrast, *minor version upgrades* include only changes that are backward-compatible with existing applications. You can initiate a minor version upgrade manually by modifying your database. Or, you can enable the **Auto minor version upgrade** option when you create or modify a database. Doing so means that Amazon RDS automatically upgrades your database after testing and approving the new version.

Amazon RDS handles Multi-AZ minor version upgrades in the following ways:

- **Multi-AZ DB instance deployment** – Amazon RDS simultaneously upgrades the primary and any standby instances. Your database might not be available for several minutes while the upgrade completes.
- **Multi-AZ DB cluster deployment** – Amazon RDS upgrades the reader DB instances one at a time. Then, one of the reader DB instances switches to be the new writer DB instance. Amazon RDS then upgrades the old writer instance (which is now a reader instance). Multi-AZ DB clusters typically reduce the downtime of minor version upgrades to approximately 35 seconds. When used with RDS Proxy, they can further reduce downtime to one second or less. For more information, see *Amazon RDS Proxy*. Alternately, you can use an open source database proxy such as ProxySQL, PgBouncer, or the AWS Advanced JDBC Wrapper Driver.

If your database has read replicas, you must first upgrade all of the read replicas before you upgrade the source instance or cluster.

For more information, see Automatic minor version upgrades for RDS for PostgreSQL. For information about manually performing a minor version upgrade, see Manually upgrading the engine version.

For more information about database engine versions and the policy for deprecating database engine versions, see Database Engine Versions in the Amazon RDS FAQs.

Topics

- Considerations for PostgreSQL upgrades
- Finding valid upgrade targets
- PostgreSQL version numbers
- RDS version numbers in RDS for PostgreSQL
- Choosing a major version for an RDS for PostgreSQL upgrade
- How to perform a major version upgrade for RDS for PostgreSQL
- Automatic minor version upgrades for RDS for PostgreSQL
- Upgrading PostgreSQL extensions in RDS for PostgreSQL databases
- Monitoring RDS for PostgreSQL engine upgrades with events

Considerations for PostgreSQL upgrades

To safely upgrade your databases, Amazon RDS uses the `pg_upgrade` utility described in the PostgreSQL documentation

If your backup retention period is greater than 0, Amazon RDS takes two DB snapshots during the upgrade process. The first DB snapshot is of the database before any upgrade changes have been made. If the upgrade fails for your databases, you can restore this snapshot to create a database running the old version. The second DB snapshot is taken after the upgrade completes. These DB snapshots are deleted automatically once the backup retention period expires.

> **Note**
>
> Amazon RDS takes DB snapshots during the upgrade process only if you have set the backup retention period for your database to a number greater than 0. To change the backup retention period for a DB instance, see the section called “Modifying a DB instance”. You can't configure a custom backup retention period for a Multi-AZ DB cluster.

When you perform a major version upgrade of a DB instance, any in-Region read replicas are also automatically upgraded. After the upgrade workflow starts, the read replicas wait for the `pg_upgrade` to complete successfully on the primary DB instance. Then the primary DB instance

upgrade waits for the read replica upgrades to complete. You experience an outage until the upgrade is complete. When you perform a major version upgrade of a Multi-AZ DB cluster, the replication state of its read replicas changes to **terminated**.

After an upgrade is complete, you can't revert to the previous version of the DB engine. If you want to return to the previous version, restore the DB snapshot that was taken before the upgrade to create a new database.

Finding valid upgrade targets

When you use the AWS Management Console to upgrade a database, it shows the valid upgrade targets for the database. You can also use the following AWS CLI command to identify the valid upgrade targets for a database:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
  --engine postgres \
  --engine-version version-number \
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
  --engine postgres ^
  --engine-version version-number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For example, to identify the valid upgrade targets for a PostgreSQL version 16.1 database, run the following AWS CLI command:

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
  --engine postgres \
  --engine-version 16.1 \
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
  --engine postgres ^
  --engine-version 16.1 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

PostgreSQL version numbers

The version numbering sequence for the PostgreSQL database engine is as follows:

- For PostgreSQL versions 10 and higher, the engine version number is in the form *major.minor*. The major version number is the integer part of the version number. The minor version number is the fractional part of the version number.

  A major version upgrade increases the integer part of the version number, such as upgrading from 10.*minor* to 11.*minor*.
- For PostgreSQL versions lower than 10, the engine version number is in the form *major.major.minor*. The major engine version number is both the integer and the first fractional part of the version number. For example, 9.6 is a major version. The minor version number is the third part of the version number. For example, for version 9.6.12, the 12 is the minor version number.

  A major version upgrade increases the major part of the version number. For example, an upgrade from *9.6*.12 to 11.14 is a major version upgrade, where *9.6* and *11* are the major version numbers.

For information about RDS Extended Support version numbering, see Amazon RDS Extended Support version naming.

RDS version numbers in RDS for PostgreSQL

RDS version numbers use the *major*.*minor*.*patch* naming scheme. An RDS patch version includes important bug fixes added to a minor version after its release. For information about RDS Extended Support version numbering, see Amazon RDS Extended Support version naming.

To identify the Amazon RDS version number of your database, you must first create the `rds_tools` extension by using the following command:

```
CREATE EXTENSION rds_tools;
```

Starting with the release of PostgreSQL version 15.2-R2, you can find out the RDS version number of your RDS for PostgreSQL database with the following SQL query:

```
postgres=> SELECT rds_tools.rds_version();
```

For example, querying an RDS for PostgreSQL 15.2 database returns the following:

```
rds_version
----------------
 15.2.R2
(1 row)
```

Choosing a major version for an RDS for PostgreSQL upgrade

Major version upgrades can contain changes that are not backward-compatible with previous versions of the database. New functionality can cause your existing applications to stop working correctly. For this reason, Amazon RDS doesn't apply major version upgrades automatically. To perform a major version upgrade, you modify your database manually. Make sure that you thoroughly test any upgrade to verify that your applications work correctly before applying the upgrade to your production databases. When you do a PostgreSQL major version upgrade, we recommend that you follow the steps described in How to perform a major version upgrade for RDS for PostgreSQL.

When you upgrade a PostgreSQL Single-AZ DB instance or Multi-AZ DB instance deployment to its next major version, any read replicas associated with the database are also upgraded to that next major version. In some cases, you can skip to a higher major version when upgrading. If your upgrade skips a major version, the read replicas are also upgraded to that target major version. Upgrades to version 11 that skip other major versions have certain limitations. You can find the details in the steps described in How to perform a major version upgrade for RDS for PostgreSQL.

Most PostgreSQL extensions aren't upgraded during a PostgreSQL engine upgrade. These must be upgraded separately. For more information, see Upgrading PostgreSQL extensions in RDS for PostgreSQL databases.

You can find out which major versions are available for your RDS for PostgreSQL database by running the following AWS CLI query:

```
aws rds describe-db-engine-versions --engine postgres  --engine-version your-version
 --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" --
output text
```

The following table summarizes the results of this query for all available versions. An asterisk (*) on the version number means that version is no longer supported. If your current version is unsupported, we recommend that you upgrade to the newest minor version upgrade target or to one of the other available upgrade targets for that version.

Current source version	Upgrade targets
17.6	None
17.5	17.6
17.4	17.6, 17.5
17.3*, 17.2	17.6, 17.5, 17.4
17.1*	17.6, 17.5, 17.4, 17.2
16.10	17.6
16.9	17.6, 17.5 16.10
16.8	17.6, 17.5, 17.4 16.10, 16.9
16.7*	17.6, 17.5, 17.4 16.10, 16.9, 16.8
16.7	17.6, 17.5, 17.4 16.10, 16.9, 16.8
16.6	17.6, 17.5, 17.4, 17.2

Current source version	Upgrade targets
	16.10, 16.9, 16.8
16.5*, 16.4	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6
16.3	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4
16.2*, 16.1*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4, 16.3
15.14	17.6 16.10
15.13	17.6, 17.5 16.10, 16.9 15.14
15.12, 15.11*	17.6, 17.5, 17.4 16.10, 16.9, 16.8 15.14, 15.13
15.10	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6 15.14, 15.13, 15.12
15.9*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6 15.14, 15.13, 15.12, 15.10

Current source version	Upgrade targets
15.8	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4 15.14, 15.13, 15.12, 15.10
15.7	16.9, 16.8, 16.7, 16.6, 16.5, 16.4, 16.3 15.13, 15.12, 15.11, 15.10, 15.9, 15.8
15.6*, 15.5*, 15.4*, 15.3*, 15.2*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4, 16.3 15.14, 15.13, 15.12, 15.10, 15.8, 15.7
14.19	17.6 16.10 15.14
14.18	17.6, 17.5 16.10, 16.9 15.14, 15.13 14.19
14.17, 14.16*	17.6, 17.5, 17.4 16.10, 16.9, 16.8 15.14, 15.13, 15.12 14.19, 14.18

Current source version	Upgrade targets
14.15	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6 15.14, 15.13, 15.12, 15.10 14.19, 14.18, 14.17
14.14*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6 15.14, 15.13, 15.12, 15.10 14.19, 14.18, 14.17, 14.15
14.13	16.4 15.13, 15.12, 15.11, 15.10, 15.9, 15.8 14.18, 14.17, 14.16, 14.15, 14.14
14.12	16.3 15.13, 15.12, 15.11, 15.10, 15.9, 15.8, 15.7 14.18, 14.17, 14.16, 14.15, 14.14, 14.13
14.11*, 14.10*, 14.9*, 14.8*, 14.7*, 14.6*, 14.5*, 14.4*, 14.3*, 14.2*, 14.1*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4, 16.3 15.14, 15.13, 15.12, 15.10, 15.8, 15.7 14.19, 14.18, 14.17, 14.15, 14.13, 14.12

Current source version	Upgrade targets
13.22	17.6 16.10 15.14 14.19
13.21	17.6, 17.5 16.10, 16.9 15.14, 15.13 14.19, 14.18 13.22
13.20, 13.19*	17.6, 17.5, 17.4 16.10, 16.9, 16.8 15.14, 15.13, 15.12 14.19, 14.18, 14.17 13.22, 13.21
13.18, 13.17*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6 15.14, 15.13, 15.12, 15.10 14.19, 14.18, 14.17, 14.15 13.22, 13.21, 13.20

Current source version	Upgrade targets
13.16	16.4 15.8 14.18, 14.17, 14.15, 14.14, 14.13 13.21, 13.20, 13.19, 13.18, 13.17
13.15	16.3 15.8, 15.7 14.18, 14.17, 14.15, 14.14, 14.13, 14.12 13.21, 13.20, 13.19, 13.18, 13.17, 13.16
13.14*, 13.13*, 13.12*, 13.11*, 13.10*, 13.9*, 13.8*, 13.7*, 13.6*, 13.5*, 13.4*, 13.3*, 13.2*, 13.1*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4, 16.3 15.14, 15.13, 15.12, 15.10, 15.8, 15.7 14.19, 14.18, 14.17, 14.15, 14.13, 14.12 13.22, 13.21, 13.20, 13.18, 13.16, 13.15
12.22-rds.20250508	17.6, 17.5 16.10, 16.9 15.14, 15.13 14.19, 14.18 13.22, 13.21

Current source version	Upgrade targets
12.22-rds.20250220	17.6, 17.5, 17.4 16.10, 16.9, 16.8 15.14, 15.13, 15.12 14.19, 14.18, 14.17 13.22, 13.21, 13.20 12.22-rds.20250508
12.22, 12.21*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6 15.14, 15.13, 15.12, 15.10 14.19, 14.18, 14.17, 14.15 13.22, 13.21, 13.20, 13.18 12.22-rds.20250220, 12.22-rds.20250508
12.20*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4 15.14, 15.13, 15.12, 15.10, 15.8 14.19, 14.18, 14.17, 14.15, 14.13 13.22, 13.21, 13.20, 13.18, 13.16 12.22, 12.22-rds.20250220, 12.22-rds.20250508

Current source version	Upgrade targets
12.19*, 12.18*, 12.17*, 12.16*, 12.15*, 12.14*, 12.13*, 12.12*, 12.11*, 12.10*, 12.9*, 12.8*, 12.7*, 12.6*, 12.5*, 12.4*, 12.3*, 12.2*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4, 16.3 15.14, 15.13, 15.12, 15.10, 15.8, 15.7 14.19, 14.18, 14.17, 14.15, 14.13, 14.12 13.22, 13.21, 13.20, 13.18, 13.16, 13.15 12.22, 12.22-rds.20250220, 12.22-rds.20250508
11.22-rds.20250508	17.6, 17.5 16.10, 16.9 15.14, 15.13 14.19, 14.18 13.22, 13.21 12.22-rds.20250508
11.22-rds.20250220	17.6, 17.5, 17.4 16.10, 16.9, 16.8 15.14, 15.13, 15.12 14.19, 14.18, 14.17 13.22, 13.21, 13.20 12.22-rds.20250220, 12.22-rds.20250508 11.22-rds.20250508

Current source version	Upgrade targets
11.22-rds.20240509	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4, 16.3 15.14, 15.13, 15.12, 15.10, 15.8, 15.7 14.19, 14.18, 14.17, 14.15, 14.13, 14.12 13.22, 13.21, 13.20, 13.18, 13.16, 13.15 12.22, 12.22-rds.20250220, 12.22-rds.20250508 11.22-rds.20240808, 11.22-rds.20241121, 11.22-rds.20250220, 11.22-rds.20250508
11.22, 11.21*, 11.20*, 11.19*, 11.18*, 11.17*, 11.16*, 11.15*, 11.14*, 11.13*, 11.12*, 11.11*, 11.10*, 11.9*, 11.8*, 11.7*, 11.6*, 11.5*, 11.4*, 11.2*, 11.1*	17.6, 17.5, 17.4, 17.2 16.10, 16.9, 16.8, 16.6, 16.4, 16.3 15.14, 15.13, 15.12, 15.10, 15.8, 15.7 14.19, 14.18, 14.17, 14.15, 14.13, 14.12 13.22, 13.21, 13.20, 13.18, 13.16, 13.15 12.22, 12.22-rds.20250220, 12.22-rds.20250508 11.22-rds.20240418, 11.22-rds.20240509, 11.22-rds.20240808, 11.22-rds.20241121, 11.22-rds.20250220, 11.22-rds.20250508

* This version is no longer supported.

How to perform a major version upgrade for RDS for PostgreSQL

We recommend the following process when performing a major version upgrade on an Amazon RDS for PostgreSQL database:

1. **Have a version-compatible parameter group ready** – If you are using a custom parameter group, you have two options. You can specify a default parameter group for the new DB engine version. Or you can create your own custom parameter group for the new DB engine version. For more information, see the section called "Parameter groups" and the section called "DB cluster parameter groups".
2. **Check for unsupported database classes** – Check that your database's instance class is compatible with the PostgreSQL version you are upgrading to. For more information, see Supported DB engines for DB instance classes.
3. **Check for unsupported usage:**
   - **Prepared transactions** – Commit or roll back all open prepared transactions before attempting an upgrade.

     You can use the following query to verify that there are no open prepared transactions on your database.

     ```
     SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
     ```

   - **Reg* data types** – Remove all uses of the *reg** data types before attempting an upgrade. Except for `regtype` and `regclass`, you can't upgrade the *reg** data types. The `pg_upgrade` utility can't persist this data type, which is used by Amazon RDS to do the upgrade.

     To verify that there are no uses of unsupported *reg** data types, use the following query for each database.

     ```
     SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
      pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid
           AND NOT a.attisdropped
           AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
                              'pg_catalog.regprocedure'::pg_catalog.regtype,
                              'pg_catalog.regoper'::pg_catalog.regtype,
                              'pg_catalog.regoperator'::pg_catalog.regtype,
                              'pg_catalog.regconfig'::pg_catalog.regtype,
                              'pg_catalog.regdictionary'::pg_catalog.regtype)
           AND c.relnamespace = n.oid
           AND n.nspname NOT IN ('pg_catalog', 'information_schema');
     ```

4. **Check for invalid databases:**

- Ensure there are no invalid databases. The `datconnlimit` column in the `pg_database` catalog includes a value of `-2` to mark databases as invalid that were interrupted during a `DROP DATABASE` operation.

  Use the following query to check for invalid databases:

  ```
  SELECT datname FROM pg_database WHERE datconnlimit = - 2;
  ```

- The previous query returns invalid database names. You can use `DROP DATABASE` *`invalid_db_name`*`;` to drop invalid databases. You can also use the following command to drop invalid databases:

  ```
  SELECT 'DROP DATABASE ' || quote_ident(datname) || ';' FROM pg_database WHERE
   datconnlimit = -2 \gexec
  ```

  For more information about invalid databases, see Understanding the behavior of autovacuum with invalid databases.

5. **Handle logical replication slots** – An upgrade can't occur if the database has any logical replication slots. Logical replication slots are typically used for AWS DMS migration and for replicating tables from the database to data lakes, BI tools, and other targets. Before upgrading, make sure that you know the purpose of any logical replication slots that are in use, and confirm that it's okay to delete them. If the logical replication slots are still being used, you shouldn't delete them, and you can't proceed with the upgrade.

   If the logical replication slots aren't needed, you can delete them using the following SQL:

   ```
   SELECT * FROM pg_replication_slots WHERE slot_type NOT LIKE 'physical';
   SELECT pg_drop_replication_slot(slot_name);
   ```

   Logical replication setups that use the `pglogical` extension also need to have slots dropped for a successful major version upgrade. For information about how to identify and drop slots created using the `pglogical` extension, see Managing logical replication slots for RDS for PostgreSQL.

   On source version 17 and later, logical replication slots on non-read-replicas can be retained through upgrades. Logical replication slots created on read replicas are not retained through upgrades.

Ensure that all transactions and logical decoding messages have been consumed from the slot before initiating the upgrade. If there are unconsumed write-ahead log files (WAL) held by logical replication slots, the upgrade will fail with a message identifying the problem slots. See the PostgreSQL documentation for further details.

On Multi-AZ clusters with source versions earlier than 17.8 or 18.2, ensure that `flow_control` is disabled. For more information, see Turning on and turning off flow control for Multi-AZ DB clusters. You can turn off flow control by removing the extension from the `shared_preload_libraries` and rebooting your DB instance.

6. **Handle read replicas** – An upgrade of a Single-AZ DB instance or Multi-AZ DB instance deployment also upgrades the in-Region read replicas along with the primary DB instance. Amazon RDS doesn't upgrade Multi-AZ DB cluster read replicas.

   You can't upgrade read replicas separately. If you could, it could lead to situations where the primary and replica databases have different PostgreSQL major versions. However, read replica upgrades might increase downtime on the primary DB instance. To prevent a read replica upgrade, promote the replica to a standalone instance or delete it before starting the upgrade process.

   The upgrade process recreates the read replica's parameter group based on the read replica's current parameter group. You can apply a custom parameter group to a read replica only after the upgrade completes by modifying the read replica. For more information about read replicas, see Working with read replicas for Amazon RDS for PostgreSQL.

7. **Handle large objects** – In PostgreSQL, large objects (also known as BLOBs) are used to store and manage large binary objects (like files, images, videos, etc.) that are larger than the maximum size allowed for regular column data types. For more information see PostgreSQL Large Objects documentation.

   An upgrade can run out of memory and fail if there are millions of large objects and the instance cannot handle them during an upgrade. The PostgreSQL major version upgrade process comprises of two broad phases: dumping the schema via pg_dump and restoring it through pg_restore. If your database has millions of large objects you need to ensure your instance has sufficient memory to handle the pg_dump and pg_restore during an upgrade and scale it to a larger instance type.

   Before you begin an upgrade, check if your database is having any large objects. The catalog `pg_largeobject_metadata` holds metadata associated with large objects. The actual large

object data is stored in `pg_largeobject`. Use the following query to check for the number of large objects:

```
SELECT count(*) FROM pg_largeobject_metadata;
```

To cleanup existing large objects or orphaned large objects, see Managing large objects with the lo module.

When planning a major version upgrade, we recommend using an instance type with at least 32 GB of memory if your database contains 25 to 30 million large objects. This recommendation is based on our tests and can vary depending on your specific workload and database configuration. If your database includes additional objects (such as tables, indexes, or materialized views), we recommend selecting a larger instance type to ensure optimal performance during the upgrade process.

8. **Handle zero-ETL integrations** – If you have an existing zero-ETL integration, delete it before performing a major version upgrade. Then, after completing the upgrade, recreate the integration.

   On source versions majors 17 and up, the zero-ETL integration can be retained through the upgrade.

9. **Perform a backup** – We recommend that you perform a backup before performing the major version upgrade so that you have a known restore point for your database. If your backup retention period is greater than 0, the upgrade process creates DB snapshots of your database before and after upgrading. To change your backup retention period, see Modifying an Amazon RDS DB instance and the section called "Modifying a Multi-AZ DB cluster".

   To perform a backup manually, see the section called "Creating a DB snapshot for a Single-AZ DB instance" and the section called "Creating a Multi-AZ DB cluster snapshot".

10. **Upgrade certain extensions before a major version upgrade** – If you plan to skip a major version with the upgrade, you need to update certain extensions *before* performing the major version upgrade. For example, upgrading from versions 9.5.x or 9.6.x to version 11.x skips a major version. The extensions to update include PostGIS and related extensions for processing spatial data.

    - `address_standardizer`
    - `address_standardizer_data_us`
    - `postgis_raster`

- `postgis_tiger_geocoder`
- `postgis_topology`

You cannot directly upgrade to PostgreSQL version 17 if you are using `rdkit` version 4.6.0 and lower, and PostgreSQL version 16 and lower, due to `rdkit` incompatibility. Below are the upgrade options:

- If you are on PostgreSQL version 13 and lower, you need to perform a major version upgrade to version 14.14 and higher 14 versions, 15.9 and higher 15 versions, or 16.5 and higher 16 versions first, and then perform the version upgrade to PostgreSQL 17.
- If you are on PostgreSQL version 14, 15, or 16, you need to perform a minor version upgrade to 14.14 and higher 14 versions, 15.9 and higher 15 versions, or 16.5 and higher 16 versions, and then upgrade to PostgreSQL version 17.

Run the following command for each extension that you're using:

```
ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version';
```

For more information, see Upgrading PostgreSQL extensions in RDS for PostgreSQL databases. To learn more about upgrading PostGIS, see Step 6: Upgrade the PostGIS extension.

11 **Drop certain extensions before the major version upgrade** – Extensions that are not supported on the target version must be dropped, or else the upgrade will fail.

The `plrust` extension is removed starting in RDS PostgreSQL 18. The `postgis_topology` extension is unavailable on RDS PostgreSQL versions 18.1 and 18.2 due to known issues [1], [2]. These extensions must be removed prior to upgrading.

An upgrade that skips a major version to version 11.x doesn't support updating the `pgRouting` extension. Upgrading from versions 9.4.x, 9.5.x, or 9.6.x to versions 11.x skips a major version. It's safe to drop the `pgRouting` extension and then reinstall it to a compatible version after the upgrade. For the extension versions you can update to, see Supported PostgreSQL extension versions.

The `tsearch2` and `chkpass` extensions are no longer supported for PostgreSQL versions 11 or later.

You can check if an extension is installed with the following query:

```
SELECT * FROM pg_extension WHERE extname in ('extension_name');
```

12 **Drop unknown data types** – Drop unknown data types depending on the target version.

PostgreSQL version 10 stopped supporting the unknown data type. If a version 9.6 database uses the unknown data type, an upgrade to a version 10 shows an error message such as the following:

```
Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
The instance could not be upgraded because the 'unknown' data type is used in user
 tables.
Please remove all usages of the 'unknown' data type and try again."
```

To find the unknown data type in your database so you can remove the offending column or change it to a supported data type, use the following SQL:

```
SELECT DISTINCT data_type FROM information_schema.columns WHERE data_type ILIKE
 'unknown';
```

13 **Perform an upgrade dry run** – We highly recommend testing a major version upgrade on a duplicate of your production database before attempting the upgrade on your production database. You can monitor the execution plans on the duplicate test database for any possible execution plan regressions and to evaluate its performance. To create a duplicate test instance, you can either restore your database from a recent snapshot or do a point-in-time restore of your database to its latest restorable time.

For more information, see the section called “Restoring from a snapshot” or the section called “Point-in-time recovery”. For Multi-AZ DB clusters, see the section called “Restoring from a snapshot to a Multi-AZ DB cluster” or the section called “Restoring a Multi-AZ DB cluster to a specified time”.

For details on performing the upgrade, see the section called “Manually upgrading the engine version”.

In upgrading a version 9.6 database to version 10, be aware that PostgreSQL 10 enables parallel queries by default. You can test the impact of parallelism *before* the upgrade by changing the `max_parallel_workers_per_gather` parameter on your test database to 2.

> **Note**
>
> The default value for `max_parallel_workers_per_gather` parameter in the `default.postgresql10` DB parameter group is 2.

For more information, see Parallel Query in the PostgreSQL documentation. To disable parallelism on version 10, set the `max_parallel_workers_per_gather` parameter to 0.

During the major version upgrade, the `public` and `template1` databases and the `public` schema in every database are temporarily renamed. These objects appear in the logs with their original name and a random string appended. The string is appended so that custom settings such as `locale` and `owner` are preserved during the major version upgrade. After the upgrade completes, the objects are renamed back to their original names.

> **Note**
>
> During the major version upgrade process, you can't do a point-in-time restore of your DB instance or Multi-AZ DB cluster. After Amazon RDS performs the upgrade, it takes an automatic backup of the database. You can perform a point-in-time restore to times before the upgrade began and after the automatic backup of your database has completed.

14 **If an upgrade fails with precheck procedure errors, resolve the issues** – During the major version upgrade process, Amazon RDS for PostgreSQL first runs a precheck procedure to identify any issues that might cause the upgrade to fail. The precheck procedure checks all potential incompatible conditions across all databases in the instance.

If the precheck encounters an issue, it creates a log event indicating the upgrade precheck failed. The precheck process details are in an upgrade log named `pg_upgrade_precheck.log` for all the databases of a database. Amazon RDS appends a timestamp to the file name. For more information about viewing logs, see Monitoring Amazon RDS log files.

If a read replica upgrade fails at precheck, replication on the failed read replica is broken and the read replica is put in the terminated state. Delete the read replica and recreate a new read replica based on the upgraded primary DB instance.

Resolve all of the issues identified in the precheck log and then retry the major version upgrade. The following is an example of a precheck log.

```
-------------------------------------------------------------------------
Upgrade could not be run on Wed Apr 4 18:30:52 2018
-------------------------------------------------------------------------
The instance could not be upgraded from 9.6.11 to 10.6 for the following reasons.
Please take appropriate action on databases that have usage incompatible with the
 requested major engine version upgrade and try the upgrade again.

* There are uncommitted prepared transactions. Please commit or rollback all prepared
 transactions.* One or more role names start with 'pg_'. Rename all role names that
 start with 'pg_'.

* The following issues in the database 'my"million$"db' need to be corrected before
 upgrading:** The ["line","reg*"] data types are used in user tables. Remove all
 usage of these data types.
** The database name contains characters that are not supported by RDS for
 PostgreSQL. Rename the database.
** The database has extensions installed that are not supported on the target
 database version. Drop the following extensions from your database: ["tsearch2"].

* The following issues in the database 'mydb' need to be corrected before
 upgrading:** The database has views or materialized views that depend on
 'pg_stat_activity'. Drop the views.
```

15 **If a read replica upgrade fails while upgrading the database, resolve the issue** – A failed read replica is placed in the `incompatible-restore` state and replication is terminated on the database. Delete the read replica and recreate a new read replica based on the upgraded primary DB instance.

> **Note**
>
> Amazon RDS doesn't upgrade read replicas for Multi-AZ DB clusters. If you perform a major version upgrade on a Multi-AZ DB cluster, then the replication state of its read replicas changes to **terminated**.

A read replica upgrade might fail for the following reasons:

- It was unable to catch up with the primary DB instance even after a wait time.

- It was in a terminal or incompatible lifecycle state such as storage-full, incompatible-restore, and so on.
- When the primary DB instance upgrade started, there was a separate minor version upgrade running on the read replica.
- The read replica used incompatible parameters.
- The read replica was unable to communicate with the primary DB instance to synchronize the data folder.

16. **Upgrade your production database** – When the dry-run major version upgrade is successful, you should be able to upgrade your production database with confidence. For more information, see Manually upgrading the engine version.

17. Run the ANALYZE operation to refresh the `pg_statistic` table. You should do this for every database on all your PostgreSQL databases. Optimizer statistics aren't transferred during a major version upgrade, so you need to regenerate all statistics to avoid performance issues. Run the command without any parameters to generate statistics for all regular tables in the current database, as follows:

```
ANALYZE VERBOSE;
```

The VERBOSE flag is optional, but using it shows you the progress. For more information, see ANALYZE in the PostgreSQL documentation.

When analyzing specific tables instead of using ANALYZE VERBOSE, run the ANALYZE command for each table as follows:

```
ANALYZE table_name;
```

For partitioned tables, always analyze the parent table. This process:

- Automatically samples rows across all partitions
- Updates statistics for each partition recursively
- Maintains essential planning statistics at the parent level

While parent tables store no actual data, analyzing them is vital for query optimization. Running ANALYZE only on individual partitions can lead to poor query performance since the optimizer won't have the comprehensive statistics needed for efficient cross-partition planning.

> **Note**
>
> Run ANALYZE on your system after the upgrade to avoid performance issues.

After the major version upgrade is complete, we recommend the following:

- A PostgreSQL upgrade doesn't upgrade any PostgreSQL extensions. To upgrade extensions, see Upgrading PostgreSQL extensions in RDS for PostgreSQL databases.
- Optionally, use Amazon RDS to view two logs that the `pg_upgrade` utility produces. These are `pg_upgrade_internal.log` and `pg_upgrade_server.log`. Amazon RDS appends a timestamp to the file name for these logs. You can view these logs as you can any other log. For more information, see Monitoring Amazon RDS log files.

  You can also upload the upgrade logs to Amazon CloudWatch Logs. For more information, see Publishing PostgreSQL logs to Amazon CloudWatch Logs.
- To verify that everything works as expected, test your application on the upgraded database with a similar workload. After the upgrade is verified, you can delete this test instance.

Automatic minor version upgrades for RDS for PostgreSQL

If you enable the **Auto minor version upgrade** option when creating or modifying a DB instance or Multi-AZ DB cluster, you can have your database automatically upgraded.

Amazon RDS also supports upgrade rollout policy to manage automatic minor version upgrades across multiple database resources and AWS accounts. For more information, see Using AWS Organizations upgrade rollout policy for automatic minor version upgrades.

For each RDS for PostgreSQL major version, one minor version is designated by RDS as the automatic upgrade version. After a minor version has been tested and approved by Amazon RDS, the minor version upgrade occurs automatically during your maintenance window. RDS doesn't automatically set newer released minor versions as the automatic upgrade version. Before RDS designates a newer automatic upgrade version, several criteria are considered, such as the following:

- Known security issues
- Bugs in the PostgreSQL community version

- Overall fleet stability since the minor version was released

You can use the following AWS CLI command to determine the current automatic minor upgrade target version for a specified PostgreSQL minor version in a specific AWS Region.

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine postgres \
--engine-version minor-version \
--region region \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine postgres ^
--engine-version minor-version ^
--region region ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output text
```

For example, the following AWS CLI command determines the automatic minor upgrade target for PostgreSQL minor version 16.1 in the US East (Ohio) AWS Region (us-east-2).

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine postgres \
--engine-version 16.1 \
--region us-east-2 \
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" \
--output table
```

For Windows:

```
aws rds describe-db-engine-versions ^
```

```
--engine postgres ^
--engine-version 16.1 ^
--region us-east-2 ^
--query "DBEngineVersions[*].ValidUpgradeTarget[*].
{AutoUpgrade:AutoUpgrade,EngineVersion:EngineVersion}" ^
--output table
```

Your output is similar to the following.

```
---------------------------------
|    DescribeDBEngineVersions    |
+--------------+-----------------+
|  AutoUpgrade |  EngineVersion  |
+--------------+-----------------+
|  False       |  16.2           |
|  True        |  16.3           |
|  False       |  16.4           |
|  False       |  16.5           |
|  False       |  16.6           |
|  False       |  17.1           |
|  False       |  17.2           |
+--------------+-----------------+
```

In this example, the `AutoUpgrade` value is `True` for PostgreSQL version 16.3. So, the automatic minor upgrade target is PostgreSQL version 16.3, which is highlighted in the output.

A PostgreSQL database is automatically upgraded during your maintenance window if the following criteria are met:

- The database has the **Auto minor version upgrade** option enabled.
- The database is running a minor DB engine version that is less than the current automatic upgrade minor version.

For more information, see Automatically upgrading the minor engine version.

> **Note**
>
> A PostgreSQL upgrade doesn't upgrade PostgreSQL extensions. To upgrade extensions, see Upgrading PostgreSQL extensions in RDS for PostgreSQL databases.

Upgrading PostgreSQL extensions in RDS for PostgreSQL databases

A PostgreSQL engine upgrade doesn't upgrade most PostgreSQL extensions. To update an extension after a version upgrade, use the `ALTER EXTENSION UPDATE` command.

> **Note**
>
> For information about updating the PostGIS extension, see Managing spatial data with the PostGIS extension (Step 6: Upgrade the PostGIS extension).
> To update the `pg_repack` extension, drop the extension and then create the new version in the upgraded database. For more information, see pg_repack installation in the `pg_repack` documentation.

To upgrade an extension, use the following command.

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

For the list of supported versions of PostgreSQL extensions, see Supported PostgreSQL extension versions.

To list your currently installed extensions, use the PostgreSQL pg_extension catalog in the following command.

```
SELECT * FROM pg_extension;
```

To view a list of the specific extension versions that are available for your installation, use the PostgreSQL pg_available_extension_versions view in the following command.

```
SELECT * FROM pg_available_extension_versions;
```

Monitoring RDS for PostgreSQL engine upgrades with events

When you upgrade the engine version of a RDS for PostgreSQL database, Amazon RDS emits a specific event during each phase of the process. To track the progress of an upgrade, you can view or subscribe to these events.

For more information about RDS events, see Monitoring Amazon RDS events.

For detailed information about a specific Amazon RDS event that occurs during your engine upgrade, see Amazon RDS event categories and event messages.

Upgrading a PostgreSQL DB snapshot engine version

With Amazon RDS, you can create a storage volume DB snapshot of your PostgreSQL DB instance. When you create a DB snapshot, the snapshot is based on the engine version used by your Amazon RDS instance. You can upgrade the engine version for your DB snapshots.

After restoring a DB snapshot upgraded to a new engine version, make sure to test that the upgrade was successful. For more information about a major version upgrade, see Upgrades of the RDS for PostgreSQL DB engine. To learn how to restore a DB snapshot, see Restoring to a DB instance.

You can upgrade manual DB snapshots that are either encrypted or not encrypted.

To view the available engine versions for your RDS for PostgreSQL DB snapshot, use the following AWS CLI example.

```
aws rds describe-db-engine-versions --engine postgres  --engine-version example-
engine-version --query "DBEngineVersions[*].ValidUpgradeTarget[*].
{EngineVersion:EngineVersion}" --output text --include-all
```

For more information about available engine versions for RDS for PostgreSQL DB snapshots, see Choosing a major version for an RDS for PostgreSQL upgrade.

> **Note**
>
> You can't upgrade automated DB snapshots that are created during the automated backup process.

Console

To upgrade a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Snapshots**.
3. Choose the snapshot that you want to upgrade.
4. For **Actions**, choose **Upgrade snapshot**. The **Upgrade snapshot** page appears.

5. Choose the **New engine version** to upgrade to.
6. Choose **Save changes** to upgrade the snapshot.

   During the upgrade process, all snapshot actions are disabled for this DB snapshot. Also, the DB snapshot status changes from **available** to **upgrading**, and then changes to **active** upon completion. If the DB snapshot can't be upgraded because of snapshot corruption issues, the status changes to **unavailable**. You can't recover the snapshot from this state.

   > **Note**
   >
   > If the DB snapshot upgrade fails, the snapshot is rolled back to the original state with the original version.

AWS CLI

To upgrade a DB snapshot to a new database engine version, use the AWS CLI modify-db-snapshot command.

Parameters

- `--db-snapshot-identifier` – The identifier of the DB snapshot to upgrade. The identifier must be a unique Amazon Resource Name (ARN). For more information, see Amazon Resource Names (ARNs) in Amazon RDS.
- `--engine-version` – The engine version to upgrade the DB snapshot to.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-snapshot \
    --db-snapshot-identifier my_db_snapshot \
    --engine-version new_version
```

For Windows:

```
aws rds modify-db-snapshot ^
    --db-snapshot-identifier my_db_snapshot ^
```

```
    --engine-version new_version
```

RDS API

To upgrade a DB snapshot to a new database engine version, call the Amazon RDS API ModifyDBSnapshot operation.

- `DBSnapshotIdentifier` – The identifier of the DB snapshot to upgrade. The identifier must be a unique Amazon Resource Name (ARN). For more information, see Amazon Resource Names (ARNs) in Amazon RDS.
- `EngineVersion` – The engine version to upgrade the DB snapshot to.

Working with read replicas for Amazon RDS for PostgreSQL

You can scale reads for your Amazon RDS for PostgreSQL DB instances by adding read replicas to the instances. As with other Amazon RDS database engines, RDS for PostgreSQL uses native replication mechanisms of PostgreSQL to keep read replicas up to date with changes on the source DB. For general information about read replicas and Amazon RDS, see Working with DB instance read replicas.

Following, you can find information specific to working with read replicas with RDS for PostgreSQL.

Read replica limitations with PostgreSQL

The following are limitations for PostgreSQL read replicas:

- PostgreSQL read replicas are read-only. Although a read replica isn't a writeable DB instance, you can promote it to become a standalone RDS for PostgreSQL DB instance. However, the process isn't reversible.
- You can't create a read replica from another read replica if your RDS for PostgreSQL DB instance is running a PostgreSQL version earlier than 14.1. RDS for PostgreSQL supports cascading read replicas on RDS for PostgreSQL version 14.1 and higher releases only. For more information, see Using cascading read replicas with RDS for PostgreSQL.
- If you promote a PostgreSQL read replica, it becomes a writable DB instance. It stops receiving write-ahead log (WAL) files from a source DB instance, and it's no longer a read-only instance. You can create new read replicas from the promoted DB instance as you do for any RDS for PostgreSQL DB instance. For more information, see Promoting a read replica to be a standalone DB instance.
- If you promote a PostgreSQL read replica from within a replication chain (a series of cascading read replicas), any existing downstream read replicas continue receiving WAL files from the promoted instance automatically. For more information, see Using cascading read replicas with RDS for PostgreSQL.
- If no user transactions are running on the source DB instance, the associated PostgreSQL read replica reports a replication lag of up to five minutes. The replica lag is calculated as `currentTime - lastCommitedTransactionTimestamp`, which means that when no transactions are being processed, the value of replica lag increases for a period of time until the write-ahead log (WAL) segment switches. By default RDS for PostgreSQL switches the WAL

segment every 5 minutes, which results in a transaction record and a decrease in the reported lag.

- You can't turn on automated backups for PostgreSQL read replicas for RDS for PostgreSQL versions earlier than 14.1. Automated backups for read replicas are supported for RDS for PostgreSQL 14.1 and higher versions only. For RDS for PostgreSQL 13 and earlier versions, create a snapshot from a read replica if you want a backup of it.
- Point-in-time recovery (PITR) isn't supported for read replicas. You can use PITR with a primary (writer) instance only, not a read replica. To learn more, see Restoring a DB instance to a specified time for Amazon RDS.
- Read replicas for PostgreSQL versions 12 and lower automatically reboot during the 60-90 day maintenance window to apply password rotation. If the replica loses connection to the source before the scheduled reboot, it still reboots to resume replication. For PostgreSQL versions 13 and higher, read replicas might experience brief replication disconnections and reconnections during the password rotation process.

Read replica configuration with PostgreSQL

RDS for PostgreSQL uses PostgreSQL native streaming replication to create a read-only copy of a source DB instance. This read replica DB instance is an asynchronously created physical replica of the source DB instance. It's created by a special connection that transmits write ahead log (WAL) data from the source DB instance to the read replica. For more information, see Streaming Replication in the PostgreSQL documentation.

PostgreSQL asynchronously streams database changes to this secure connection as they're made on the source DB instance. You can encrypt communications from your client applications to the source DB instance or any read replicas by setting the `ssl` parameter to `1`. For more information, see Using SSL with a PostgreSQL DB instance .

PostgreSQL uses a *replication* role to perform streaming replication. The role is privileged, but you can't use it to modify any data. PostgreSQL uses a single process for handling replication.

You can create a PostgreSQL read replica without affecting operations or users of the source DB instance. Amazon RDS sets the necessary parameters and permissions for you, on the source DB instance and the read replica, without affecting the service. A snapshot is taken of the source DB instance, and this snapshot is used to create the read replica. If you delete the read replica at some point in the future, no outage occurs.

You can create up to 15 read replicas from one source DB instance within the same Region. As of RDS for PostgreSQL 14.1, you can also create up to three levels of read replica in a chain (cascade) from a source DB instance. For more information, see Using cascading read replicas with RDS for PostgreSQL. In all cases, the source DB instance needs to have automated backups configured. You do this by setting the backup retention period on your DB instance to any value other than 0. For more information, see Creating a read replica.

You can create read replicas for your RDS for PostgreSQL DB instance in the same AWS Region as your source DB instance. This is known as *in-Region* replication. You can also create read replicas in different AWS Regions than the source DB instance. This is known as *cross-Region* replication. For more information about setting up cross-Region read replicas, see Creating a read replica in a different AWS Region. The various mechanisms supporting the replication process for in-Region and cross-Region differ slightly depending on the RDS for PostgreSQL version as explained in How streaming replication works for different RDS for PostgreSQL versions.

For replication to operate effectively, each read replica should have the same amount of compute and storage resources as the source DB instance. If you scale the source DB instance, be sure to also scale the read replicas.

Amazon RDS overrides any incompatible parameters on a read replica if they prevent the read replica from starting. For example, suppose that the `max_connections` parameter value is higher on the source DB instance than on the read replica. In that case, Amazon RDS updates the parameter on the read replica to be the same value as that on the source DB instance.

RDS for PostgreSQL read replicas have access to external databases that are available through foreign data wrappers (FDWs) on the source DB instance. For example, suppose that your RDS for PostgreSQL DB instance is using the `mysql_fdw` wrapper to access data from RDS for MySQL. If so, your read replicas can also access that data. Other supported FDWs include `oracle_fdw`, `postgres_fdw`, and `tds_fdw`. For more information, see Working with the supported foreign data wrappers for Amazon RDS for PostgreSQL.

Using RDS for PostgreSQL read replicas with Multi-AZ configurations

You can create a read replica from a single-AZ or Multi-AZ DB instance. You can use Multi-AZ deployments to improve the durability and availability of critical data, with a standby replica. A *standby replica* is a dedicated read replica that can assume the workload if the source DB fails over. You can't use your standby replica to serve read traffic. However, you can create read replicas from high-traffic Multi-AZ DB instances to offload read-only queries. To learn more about Multi-AZ deployments, see Multi-AZ DB instance deployments for Amazon RDS.

If the source DB instance of a Multi-AZ deployment fails over to a standby, the associated read replicas switch to using the standby (now primary) as their replication source. The read replicas might need to restart, depending on the RDS for PostgreSQL version, as follows:

- **PostgreSQL 13 and higher versions** – Restarting isn't required. The read replicas are automatically synchronized with the new primary. However, in some cases your client application might cache Domain Name Service (DNS) details for your read replicas. If so, set the time-to-live (TTL) value to less than 30 seconds. Doing this prevents the read replica from holding on to a stale IP address (and thus, prevents it from synchronizing with the new primary). To learn more about this and other best practices, see Amazon RDS basic operational guidelines.
- **PostgreSQL 12 and all earlier versions** – The read replicas restart automatically after a fail over to the standby replica because the standby (now primary) has a different IP address and a different instance name. Restarting synchronizes the read replica with the new primary.

To learn more about failover, see Failing over a Multi-AZ DB instance for Amazon RDS. To learn more about how read replicas work in a Multi-AZ deployment, see Working with DB instance read replicas.

To provide failover support for a read replica, you can create the read replica as a Multi-AZ DB instance so that Amazon RDS creates a standby of your replica in another Availability Zone (AZ). Creating your read replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ DB instance.

Logical decoding on a read replica

RDS for PostgreSQL supports logical replication from standbys with PostgreSQL 16.1. This allows you to create logical decoding from a read-only standby that reduces the load on the primary DB instance. You can achieve higher-availability for your applications that need to synchronize data across multiple systems. This feature boosts the performance of your data warehouse and data analytics.

Also, replication slots on a given standby persist the promotion of that standby to a primary. This means that in the event of a primary DB instance failover or the promotion of a standby to be the new primary, the replication slots will persist and the former standby subscribers will not be affected.

To create logical decoding on a read replica

1. **Turn on logical replication** – To create logical decoding on a standby, you must turn on logical replication on your source DB instance and its physical replica. For more information, see Read replica configuration with PostgreSQL.

   - **To turn on logical replication for a newly created RDS for PostgreSQL DB instance** – Create a new DB custom parameter group and set the static parameter `rds.logical_replication` to 1. Then, associate this DB parameter group with the Source DB instance and its physical read replica. For more information, see Associating a DB parameter group with a DB instance in Amazon RDS.
   - **To turn on logical replication for an existing RDS for PostgreSQL DB instance** – Modify the DB custom parameter group of the source DB instance and its physical read replica to set the static parameter `rds.logical_replication` to 1. For more information, see Modifying parameters in a DB parameter group in Amazon RDS.

   > **Note**
   >
   > You must reboot the DB instance to apply these parameter changes.

   You can use the following query to verify the values for `wal_level` and `rds.logical_replication` on the source DB instance and its physical read replica.

```
Postgres=>SELECT name,setting FROM pg_settings WHERE name IN
 ('wal_level','rds.logical_replication');

 name                    | setting
-------------------------+---------
 rds.logical_replication | on
 wal_level               | logical
(2 rows)
```

2. **Create a table in the source database** – Connect to the database in your source DB instance. For more information, see Connecting to a DB instance running the PostgreSQL database engine.

   Use the following queries to create table in your source database and to insert values:

```
Postgres=>CREATE TABLE LR_test (a int PRIMARY KEY);
CREATE TABLE
```

```
Postgres=>INSERT INTO LR_test VALUES (generate_series(1,10000));
INSERT 0 10000
```

3. **Create a publication for the source table** – Use the following query to create a publication for the table on the source DB instance.

```
Postgres=>CREATE PUBLICATION testpub FOR TABLE LR_test;
CREATE PUBLICATION
```

Use a SELECT query to verify the details of the publication that was created on both the source DB instance and the physical read replica instance.

```
Postgres=>SELECT * from pg_publication;

oid    | pubname | pubowner | puballtables | pubinsert | pubupdate | pubdelete |
 pubtruncate | pubviaroot
-------+---------+----------+--------------+-----------+-----------+-----------
+-------------+------------
 16429 | testpub |    16413 | f            | t         | t         | t         | t
          | f
(1 row)
```

4. **Create a subscription from logical replica instance** – Create another RDS for PostgreSQL DB instance as the logical replica instance. Make sure that VPC is setup correctly to ensure that this logical replica instance can access the physical read replica instance. For more information, see Amazon VPC and Amazon RDS. If your source DB instance is idle, connectivity issues might occur and the primary doesn't send the data to standby.

```
Postgres=>CREATE SUBSCRIPTION testsub CONNECTION 'host=Physical replica host name
 port=port
                dbname=source_db_name user=user password=password'
 PUBLICATION testpub;
NOTICE:  created replication slot "testsub" on publisher
CREATE SUBSCRIPTION
```

```
Postgres=>CREATE TABLE LR_test (a int PRIMARY KEY);
```

```
CREATE TABLE
```

Use a SELECT query to verify the details of the subscription on the logical replica instance.

```
Postgres=>SELECT oid,subname,subenabled,subslotname,subpublications FROM
 pg_subscription;

oid    | subname | subenabled | subslotname | subpublications
-------+---------+------------+-------------+-----------------
 16429 | testsub | t          | testsub     | {testpub}
(1 row)
postgres=> select count(*) from LR_test;
 count
-------
 10000
(1 row)
```

5. **Inspect logical replication slot state** – You can only see the physical replication slot on your source DB instance.

```
Postgres=>select slot_name, slot_type, confirmed_flush_lsn from
 pg_replication_slots;

slot_name                                    | slot_type | confirmed_flush_lsn
---------------------------------------------+-----------+---------------------
 rds_us_west_2_db_dhqfsmo5wbbjqrn3m6b6ivdhu4 | physical  |
(1 row)
```

However, on your read replica instance, you can see the logical replication slot and the `confirmed_flush_lsn` value changes as the application actively consumes logical changes.

```
Postgres=>select slot_name, slot_type, confirmed_flush_lsn from
 pg_replication_slots;

slot_name | slot_type | confirmed_flush_lsn
----------+-----------+---------------------
 testsub   | logical   | 0/500002F0
(1 row)
```

```
Postgres=>select slot_name, slot_type, confirmed_flush_lsn from
 pg_replication_slots;

slot_name | slot_type | confirmed_flush_lsn
-----------+-----------+---------------------
 testsub   | logical   | 0/5413F5C0
(1 row)
```

Using cascading read replicas with RDS for PostgreSQL

As of version 14.1, RDS for PostgreSQL supports cascading read replicas. With *cascading read replicas*, you can scale reads without adding overhead to your source RDS for PostgreSQL DB instance. Updates to the WAL log aren't sent by the source DB instance to each read replica. Instead, each read replica in a cascading series sends WAL log updates to the next read replica in the series. This reduces the burden on the source DB instance.

With cascading read replicas, your RDS for PostgreSQL DB instance sends WAL data to the first read replica in the chain. That read replica then sends WAL data to the second replica in the chain, and so on. The end result is that all read replicas in the chain have the changes from the RDS for PostgreSQL DB instance, but without the overhead solely on the source DB instance.

You can create a series of up to three read replicas in a chain from a source RDS for PostgreSQL DB instance. For example, suppose that you have an RDS for PostgreSQL 14.1 DB instance, `rpg-db-main`. You can do the following:

- Starting with `rpg-db-main`, create the first read replica in the chain, `read-replica-1`.
- Next, from `read-replica-1`, create the next read replica in the chain, `read-replica-2`.
- Finally, from `read-replica-2`, create the third read replica in the chain, `read-replica-3`.

You can't create another read replica beyond this third cascading read replica in the series for `rpg-db-main`. A complete series of instances from an RDS for PostgreSQL source DB instance through to the end of a series of cascading read replicas can consist of at most four DB instances.

For cascading read replicas to work, turn on automatic backups on your RDS for PostgreSQL. Create the read replica first and then turn on automatic backups on the RDS for PostgreSQL DB instance. The process is the same as for other Amazon RDS DB engines. For more information, see Creating a read replica.

As with any read replica, you can promote a read replica that's part of a cascade. Promoting a read replica from within a chain of read replicas removes that replica from the chain. For example, suppose that you want to move some of the workload off of your `rpg-db-main` DB instance to a new instance for use by the accounting department only. Assuming the chain of three read replicas from the example, you decide to promote `read-replica-2`. The chain is affected as follows:

- Promoting `read-replica-2` removes it from the replication chain.
  - It is now a full read/write DB instance.
  - It continues replicating to `read-replica-3`, just as it was doing before promotion.
- Your `rpg-db-main` continues replicating to `read-replica-1`.

For more information about promoting read replicas, see Promoting a read replica to be a standalone DB instance.

Note

- RDS for PostgreSQL doesn't support major version upgrades for cascading replicas. Before performing a major version upgrade, you need to remove cascading replicas. You can recreate them after completing the upgrade on your source DB instance and first-level replicas.
- For cascading read replicas, RDS for PostgreSQL supports 15 read replicas for each source DB instance at first level of replication, and 5 read replicas for each source DB instance at the second and third level of replication.

Creating cross-Region cascading read replicas with RDS for PostgreSQL

RDS for PostgreSQL supports cross-Region cascading read replicas. You can create a cross-Region replica from the source DB instance, and then create same-Region replicas from it. You can also create a same-Region replica from the source DB instance, and then create cross-Region replicas from it.

Create a cross-Region replica and then create same-Region replicas

You can use an RDS for PostgreSQL DB instance with version 14.1 or higher, `rpg-db-main`, to do the following:

1. Start with `rpg-db-main` (US-EAST-1), create the first cross-Region read replica in the chain, `read-replica-1` (US-WEST-2).
2. Using the first cross-Region `read-replica-1` (US-WEST-2), create the second read replica in the chain, `read-replica-2` (US-WEST-2).
3. Using `read-replica-2`, create the third read replica in the chain, `read-replica-3` (US-WEST-2).

Create a same-Region replica and then create cross-Region replicas

You can use an RDS for PostgreSQL DB instance with version 14.1 or higher, `rpg-db-main`, to do the following:

1. Starting with `rpg-db-main` (US-EAST-1), create the first read replica in the chain, `read-replica-1` (US-EAST-1).
2. Using `read-replica-1` (US-EAST-1), create the first cross-Region read replica in the chain, `read-replica-2` (US-WEST-2).
3. Using `read-replica-2` (US-WEST-2), create the third read replica in the chain, `read-replica-3` (US-WEST-2).

Limitations in creating cross-Region read replicas

- A cross-Region cascading chain of database replicas can span a maximum of two Regions, with a maximum of four levels. The four levels include the database source and three read replicas.

Advantages of using cascading read replicas

- Improved read scalability – By distributing read queries across multiple replicas, cascading replication helps balance the load. This improves performance, especially in read-heavy applications, by reducing the strain on the writer database.
- Geographical distribution – Cascading replicas can be located in different geographic locations. This reduces latency for users located far from the primary database and provides a local read replica, enhancing performance and user experience.
- High availability and disaster recovery – In the event of a primary server failure, replicas can be promoted to primary, ensuring continuity. cascading replication further enhances this by providing multiple layers of failover options, improving the overall resilience of the system.

- Flexibility and modular growth – As the system grows, new replicas can be added at different levels without major reconfiguration of the primary database. This modular approach allows for scalable and manageable growth of the replication setup.

Best practice for using cross-Region read replicas

- Before promoting a replica, create additional replicas. This will save time, and provide efficient handling of the workload.

How streaming replication works for different RDS for PostgreSQL versions

As discussed in Read replica configuration with PostgreSQL, RDS for PostgreSQL uses PostgreSQL's native streaming replication protocol to send WAL data from the source DB instance. It sends source WAL data to read replicas for both in-Region and cross-Region read replicas. With version 9.4, PostgreSQL introduced physical replication slots as a supporting mechanism for the replication process.

A *physical replication slot* prevents a source DB instance from removing WAL data before it's consumed by all read replicas. Each read replica has its own physical slot on the source DB instance. The slot keeps track of the oldest WAL (by logical sequence number, LSN) that might be needed by the replica. After all slots and DB connections have progressed beyond a given WAL (LSN), that LSN becomes a candidate for removal at the next checkpoint.

Amazon RDS uses Amazon S3 to archive WAL data. For in-Region read replicas, you can use this archived data to recover the read replica when necessary. An example of when you might do so is if the connection between source DB and read replica is interrupted for any reason.

In the following table, you can find a summary of differences between PostgreSQL versions and the supporting mechanisms for in-Region and cross-Region used by RDS for PostgreSQL.

Version	In-Region	Cross-Region
PostgreSQL 14.1 and higher versions	• Replication slots • Amazon S3 archive	• Replication slots

Version	In-Region	Cross-Region
PostgreSQL 13 and lower versions	• Amazon S3 archive	• Replication slots

For more information, see Monitoring and tuning the replication process.

Understanding the parameters that control PostgreSQL replication

The following parameters affect the replication process and determine how well read replicas stay up to date with the source DB instance:

max_wal_senders

The `max_wal_senders` parameter specifies the maximum number of connections that the source DB instance can support at the same time over the streaming replication protocol.

The default value varies for RDS for PostgreSQL versions:

- For versions 13, 14, and 15, the default value is 20.
- For versions 16 and above, the default value is 35.

This parameter should be set to slightly higher than the actual number of read replicas. If this parameter is set too low for the number of read replicas, replication stops.

For more information, see max_wal_senders in the PostgreSQL documentation.

> **Note**
>
> `max_wal_senders` is a static parameter that requires a DB instance reboot for changes to take effect.

wal_keep_segments

The `wal_keep_segments` parameter specifies the number of write-ahead log (WAL) files that the source DB instance keeps in the `pg_wal` directory. The default setting is 32.

If `wal_keep_segments` isn't set to a large enough value for your deployment, a read replica can fall so far behind that streaming replication stops. If that happens, Amazon RDS generates a replication error and begins recovery on the read replica. It does so by replaying the source

DB instance's archived WAL data from Amazon S3. This recovery process continues until the read replica has caught up enough to continue streaming replication. You can see this process in action as captured by the PostgreSQL log in Example: How a read replica recovers from replication interruptions.

> **Note**
>
> In PostgreSQL version 13, the `wal_keep_segments` parameter is named `wal_keep_size`. It serves the same purpose as `wal_keep_segments`, but its default value is in megabytes (MB) (2048 MB) rather than the number of files. For more information, see wal_keep_segments and wal_keep_size in the PostgreSQL documentation.

max_slot_wal_keep_size

The `max_slot_wal_keep_size` parameter controls the quantity of WAL data that the RDS for PostgreSQL DB instance retains in the `pg_wal` directory to serve slots. This parameter is used for configurations that use replication slots. The default value for this parameter is `-1`, meaning that there's no limit to how much WAL data is kept on the source DB instance. For information about monitoring your replication slots, see Monitoring replication slots for your RDS for PostgreSQL DB instance.

For more information about this parameter, see max_slot_wal_keep_size in the PostgreSQL documentation.

Whenever the stream that provides WAL data to a read replica is interrupted, PostgreSQL switches into recovery mode. It restores the read replica by using archived WAL data from Amazon S3 or by using the WAL data associated with the replication slot. When this process is complete, PostgreSQL re-establishes streaming replication.

Example: How a read replica recovers from replication interruptions

In the following example, you find the log details that demonstrate the recovery process for a read replica. The example is from an RDS for PostgreSQL DB instance running PostgreSQL version 12.9 in the same AWS Region as the source DB, so replication slots aren't used. The recovery process is the same for other RDS for PostgreSQL DB instances running PostgreSQL earlier than version 14.1 with in-Region read replicas.

When the read replica lost contact with the source DB instance, Amazon RDS records the issue in the log as `FATAL: could not receive data from WAL stream` message, along with the `ERROR: requested WAL segment ... has already been removed`. As shown in the bold line, Amazon RDS recovers the replica by replaying an archived WAL file.

```
2014-11-07 19:01:10 UTC::@:[23180]:DEBUG:  switched WAL source from archive to stream
 after failure
2014-11-07 19:01:10 UTC::@:[11575]:LOG: started streaming WAL from primary at 1A/
D3000000 on timeline 1
2014-11-07 19:01:10 UTC::@:[11575]:FATAL: could not receive data from WAL stream:
ERROR:  requested WAL segment 00000001000000001A000000D3 has already been removed
2014-11-07 19:01:10 UTC::@:[23180]:DEBUG: could not restore file "00000002.history"
 from archive: return code 0
2014-11-07 19:01:15 UTC::@:[23180]:DEBUG: switched WAL source from stream to archive
 after failure recovering 00000001000000001A000000D3
2014-11-07 19:01:16 UTC::@:[23180]:LOG:  restored log file "00000001000000001A000000D3"
 from archive
```

When Amazon RDS replays enough archived WAL data on the replica to catch up, streaming to the read replica begins again. When streaming resumes, Amazon RDS writes an entry to the log file similar to the following.

```
2014-11-07 19:41:36 UTC::@:[24714]:LOG:started streaming WAL from primary at 1B/
B6000000 on timeline 1
```

Setting the parameters that control shared memory

The parameters you set determine the size of shared memory for tracking transaction IDs, locks, and prepared transactions. **The shared memory structure of a standby instance must be equal or greater than that of a primary instance.** This ensures that the former doesn't run out of shared memory during recovery. If the parameter values on the replica are less than the parameter values on the primary, Amazon RDS will automatically adjust the replica parameters and restart the engine.

The parameters affected are:

- max_connections
- max_worker_processes
- max_wal_senders

- max_prepared_transactions
- max_locks_per_transaction

To avoid RDS reboots of replicas due to insufficient memory, we recommend applying the parameter changes as a rolling reboot to each replica. You must apply the following rules, when you set the parameters:

- **Increasing the parameter values:**
  - You should always increase the parameter values of all the read replicas first, and perform a rolling reboot of all replicas. Then, apply the parameter changes on the primary instance and reboot.
- **Decreasing the parameter values:**
  - You should first decrease the parameter values of the primary instance and perform a reboot. Then, apply the parameter changes to all the associated read replicas and perform a rolling reboot.

Monitoring and tuning the replication process

We strongly recommend that you routinely monitor your RDS for PostgreSQL DB instance and read replicas. You need to ensure that your read replicas are keeping up with changes on the source DB instance. Amazon RDS transparently recovers your read replicas when interruptions to the replication process occur. However, it's best to avoid needing to recover at all. Recovering using replication slots is faster than using the Amazon S3 archive, but any recovery process can affect read performance.

To determine how well your read replicas are keeping up with the source DB instance, you can do the following:

- **Check the amount of `ReplicaLag` between source DB instance and replicas.** *Replica lag* is the amount of time, in seconds, that a read replica lags behind its source DB instance. This metric reports the result of the following query.

  ```
  SELECT extract(epoch from now() - pg_last_xact_replay_timestamp()) AS "ReplicaLag";
  ```

  Replica lag is an indication of how well a read replica is keeping up with the source DB instance. It's the amount of latency between the source DB instance and a specific read instance. A high

value for replica lag can indicate a mismatch between the DB instance classes or storage types (or both) used by the source DB instance and its read replicas. The DB instance class and storage types for DB source instance and all read replicas should be the same.

Replica lag can also be the result of intermittent connection issues. You can monitor replication lag in Amazon CloudWatch by viewing the Amazon RDS `ReplicaLag` metric. To learn more about `ReplicaLag` and other metrics for Amazon RDS, see Amazon CloudWatch metrics for Amazon RDS.

- **Check the PostgreSQL log for information you can use to adjust your settings.** At every checkpoint, the PostgreSQL log captures the number of recycled transaction log files, as shown in the following example.

```
2014-11-07 19:59:35 UTC::@:[26820]:LOG:  checkpoint complete: wrote 376 buffers
 (0.2%);
0 transaction log file(s) added, 0 removed, 1 recycled; write=35.681 s, sync=0.013 s,
 total=35.703 s;
sync files=10, longest=0.013 s, average=0.001 s
```

  You can use this information to figure out how many transaction files are being recycled in a given time period. You can then change the setting for `wal_keep_segments` if necessary. For example, suppose that the PostgreSQL log at `checkpoint complete` shows `35 recycled` for a 5-minute interval. In this case, the `wal_keep_segments` default value of 32 isn't sufficient to keep pace with the streaming activity, so you should increase the value of this parameter.

- **Use Amazon CloudWatch to monitor metrics that can predict replication issues.** Rather than analyzing the PostgreSQL log directly, you can use Amazon CloudWatch to check metrics that have been collected. For example, you can check the value of the `TransactionLogsGeneration` metric to see how much WAL data is being generated by the source DB instance. In some cases, the workload on your DB instance might generate a large amount of WAL data. If so, you might need to change the DB instance class for your source DB instance and read replicas. Using an instance class with high (10 Gbps) network performance can reduce replica lag.

Monitoring replication slots for your RDS for PostgreSQL DB instance

All versions of RDS for PostgreSQL use replication slots for cross-Region read replicas. RDS for PostgreSQL 14.1 and higher versions use replication slots for in-Region read replicas. In-region read replicas also use Amazon S3 to archive WAL data. In other words, if your DB instance and read

replicas are running PostgreSQL 14.1 or higher, replication slots and Amazon S3 archives are both available for recovering the read replica. Recovering a read replica using its replication slot is faster than recovering from Amazon S3 archive. So, we recommend that you monitor the replication slots and related metrics.

You can view the replication slots on your RDS for PostgreSQL DB instances by querying the `pg_replication_slots` view, as follows.

```
postgres=> SELECT * FROM pg_replication_slots;
slot_name                  | plugin | slot_type | datoid | database | temporary |
 active | active_pid | xmin | catalog_xmin | restart_lsn | confirmed_flush_lsn |
 wal_status | safe_wal_size | two_phase
---------------------------+--------+-----------+--------+----------+-----------
+--------+------------+------+--------------+-------------+---------------------
+------------+---------------+-----------
rds_us_west_1_db_555555555 |        | physical  |        |          | f         | t
  |      13194 |      |              | 23/D8000060 |                     | reserved   |
               | f
(1 row)
```

The `wal_status` of `reserved` value means that the amount of WAL data held by the slot is within the bounds of the `max_wal_size` parameter. In other words, the replication slot is properly sized. Other possible status values are as follows:

- `extended` – The slot exceeds the `max_wal_size` setting, but the WAL data is retained.
- `unreserved` – The slot no longer has the all required WAL data. Some of it will be removed at the next checkpoint.
- `lost` – Some required WAL data has been removed. The slot is no longer usable.

The `unreserved` and `lost` states of the `wal_status` are seen only when `max_slot_wal_keep_size` is non-negative.

The `pg_replication_slots` view shows you the current state of your replication slots. To assess the performance of your replication slots, you can use Amazon CloudWatch and monitor the following metrics:

- **`OldestReplicationSlotLag`** – Shows the amount of Write-Ahead Log (WAL) data on the source that hasn't been consumed by the most lagging replica.

- **`TransactionLogsDiskUsage`** – Shows how much storage is being used for WAL data. When a read replica lags significantly, the value of this metric can increase substantially.

To learn more about using Amazon CloudWatch and its metrics for RDS for PostgreSQL, see Monitoring Amazon RDS metrics with Amazon CloudWatch. For more information about monitoring streaming replication on your RDS for PostgreSQL DB instances, see Best practices for Amazon RDS PostgreSQL replication on the *AWS Database Blog*.

Configuring delayed replication with RDS for PostgreSQL

Overview and Benefits

The delayed replication feature in RDS for PostgreSQL allows you to intentionally delay the replication of data changes from your primary database to one or more standby (read replica) servers. This provides valuable protection against data corruption, accidental data loss, or erroneous transactions that could otherwise be immediately propagated to all replicas.

Delayed replication is supported in the following RDS for PostgreSQL versions:

- 14.19 and higher 14 versions
- 15.14 and higher 15 versions
- 16.10 and higher 16 versions
- 17.6 and higher 17 versions

By introducing a time lag in the replication process, you gain a window of opportunity to detect and respond to data-related incidents before they affect your entire DB cluster. Key benefits of delayed replication include the following:

- Allows you to recover from accidental deletions, updates, or other logical mistakes.
- Provides a buffer against the spread of corrupted data across your DB cluster.
- Offers an additional recovery point option to complement your traditional backup strategies.
- Allows you to configure the delay period based on your organization's specific needs and risk tolerance.

Enabling and Configuring Delayed Replication

To enable delayed replication on an RDS for PostgreSQL read replica, follow these steps:

> **Note**
>
> For cascaded read replicas, use the same `recovery_min_apply_delay` parameter and steps described below.

To enable delayed replication

1. Create a new custom parameter group or modify an existing one. For more information, see DB parameter groups for Amazon RDS DB instances.
2. In the parameter group, configure the `recovery_min_apply_delay` parameter:
   - Set the value to the desired delay in milliseconds. For example, 3600000 for a 1-hour delay.
   - Allowed range: 0 to 86400000 ms (0 to 24 hours)
   - Default: 0
3. Apply the parameter group to the read replica instance you want to configure for delayed replication.
4. Reboot the read replica instance for the changes to take effect.

> **Note**
>
> The `recovery_min_apply_delay` parameter is dynamic. If you modify an existing parameter group that's already attached to the instance, the changes take effect immediately without requiring a reboot. However, when applying a new parameter group to the instance, you must reboot for the changes to take effect.

Managing Delayed Replication Recovery

Delayed replication is particularly useful in scenarios where traditional point-in-time recovery methods may be insufficient or too time-consuming.

During the delayed replication period, you can use the following PostgreSQL functions to manage the recovery process:

- `pg_wal_replay_pause()`: Request to pause the recovery process on the delayed replica.
- `pg_wal_replay_resume()`: Restart the recovery process if it was previously paused.

- `pg_is_wal_replay_paused()`: Check if the recovery process is currently paused.
- `pg_get_wal_replay_pause_state()`: Get the current state of the recovery process (not paused, pause requested, or paused).

Users with the `rds_superuser` role have EXECUTE privileges on `pg_wal_replay_pause()` and `pg_wal_replay_resume()`. If other database users need access to these functions, you must grant them the `rds_superuser` role. For more information about the `rds_superuser` role, see Understanding the rds_superuser role.

Access to other functions like `pg_is_wal_replay_paused()` and `pg_get_wal_replay_pause_state()` doesn't require the `rds_superuser` role.

You can use the following recovery target parameters to precisely control the point in time to which the delayed replica is recovered. These parameters are static and require a database reboot to apply changes:

- recovery_target
- recovery_target_lsn
- recovery_target_name
- recovery_target_time
- recovery_target_xid
- recovery_target_inclusive

> **Important**
>
> You can specify only one recovery target parameter at a time. Configuring multiple recovery target parameters in the configuration file results in an error.

Planning considerations

Consider the following when planning delayed replication with RDS for PostgreSQL:

- During the automatic rotation of `rdsrepladmin` credentials (which occurs every 90 days), delayed read replicas may temporarily enter a `REPLICATION_ERROR` state. If the delayed replica has sufficient WAL logs to maintain the configured delay, it may pause WAL receiver process,

causing WAL accumulation on the source. You should monitor the replication status on the replica and the storage consumption on the source to avoid hitting storage-full.

- When delayed read replicas encounter system events (such as reboot or restart), they enter a `REPLICATION_ERROR` state where the WAL receiver process remains inactive until the configured delay period expires. This behavior can cause WAL accumulation on the source instance, potentially leading to storage exhaustion. Consider the following preventive measures:
  - Configure CloudWatch alarms to monitor storage utilization on source instances.
  - Enable storage auto-scaling to handle unexpected WAL growth.
  - Set the `max_slot_wal_keep_size` parameter on the source instance to limit WAL retention per replication slot.
  - Monitor replication lag and slot status regularly.
- Longer delays increase WAL logs on replicas, consuming more storage. Monitor storage space using CloudWatch alarms, enable auto-scaling, or catch up replicas when needed.
- When promoting a delayed read replica, the `recovery_min_apply_delay` parameter is not honored, and all pending WAL records are immediately applied.
- The `recovery_min_apply_delay` parameter is independent on each level of a cascading replication setup. Setting a delay on a replica does not add to the delay of any cascaded replicas.

For more information, see the RDS for PostgreSQL Read Replicas documentation and the RDS for PostgreSQL Disaster Recovery documentation.

Understanding limitations

The delayed replication feature for Amazon RDS for PostgreSQL has the following limitations:

- Blue/Green deployments have the following limitations when configuring delayed replication:
  - **Green source instance** — The `recovery_min_apply_delay parameter` is disregarded, even if configured in the parameter group. Any delay settings on the green source instance do not take effect.
  - **Green replica instance** — The `recovery_min_apply_delay parameter` is fully supported and applied to the PostgreSQL configuration file. Delay settings function as expected during the switchover workflow.
  - RDS Blue/Green deployments for major version upgrades
- During major version upgrades, any delayed read replicas will be automatically terminated to allow the source instance to proceed with the upgrade process in order to ensure minimal

downtime. After the source instance completed the upgrade, you must manually recreate the delayed replicas.

- Delayed replication is not compatible with the following features.
  - RDS for PostgreSQL Logical Replication
  - RDS for PostgreSQL Multi-AZ Clusters (including both inbound and outbound replication)
  - Aurora PostgreSQL

Troubleshooting for RDS for PostgreSQL read replica

Following, you can find troubleshooting ideas for some common RDS for PostgreSQL read replica issues.

Terminate the query that causes the read replica lag

Transactions either in active or idle in transaction state that are running for a long time in the database might interfere with the WAL replication process, thereby increasing the replication lag. Therefore, be sure to monitor the runtime of these transactions with the PostgreSQL `pg_stat_activity` view.

Run a query on the primary instance similar to the following to find the process ID (PID) of the query that's running for a long time:

```
SELECT datname, pid,usename, client_addr, backend_start,
xact_start, current_timestamp - xact_start AS xact_runtime, state,
backend_xmin FROM pg_stat_activity WHERE state='active';
```

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
WHERE state  = 'idle in transaction'
AND   xact_start is not null
ORDER BY 1 DESC;
```

After identifying the PID of the query, you can choose to end the query.

Run a query on the primary instance similar to the following to terminate the query that's running for a long time:

```
SELECT pg_terminate_backend(PID);
```

Improving query performance for RDS for PostgreSQL with Amazon RDS Optimized Reads

You can achieve faster query processing for RDS for PostgreSQL with Amazon RDS Optimized Reads. An RDS for PostgreSQL DB instance or Multi-AZ DB cluster that uses RDS Optimized Reads can achieve up to 50% faster query processing compared to one that doesn't use it.

Topics

- Overview of RDS Optimized Reads in PostgreSQL
- Use cases for RDS Optimized Reads
- Best practices for RDS Optimized Reads
- Using RDS Optimized Reads
- Monitoring DB instances that use RDS Optimized Reads
- Limitations for RDS Optimized Reads in PostgreSQL

Overview of RDS Optimized Reads in PostgreSQL

Optimized Reads is available by default on RDS for PostgreSQL versions 15.2 and higher, 14.7 and higher, and 13.10 and higher when using NVMe-based DB instance classes. For hardware specifications that indicate which instances use NVMe, see Hardware specifications for DB instance classes.

When you use an RDS for PostgreSQL DB instance or Multi-AZ DB cluster that has RDS Optimized Reads turned on, it achieves up to 50% faster query performance using the local Non-Volatile Memory Express (NVMe) based solid state drive (SSD) block-level storage. You can achieve faster query processing by placing the temporary tables that are generated by PostgreSQL on the local storage, which reduces the traffic to Elastic Block Storage (EBS) over the network.

In PostgreSQL, temporary objects are assigned to a temporary namespace that drops automatically at the end of the session. The temporary namespace while dropping removes any objects that are session-dependent, including schema-qualified objects, such as tables, functions, operators, or even extensions.

In RDS for PostgreSQL, the `temp_tablespaces` parameter is configured for this temporary work area where the temporary objects are stored.

The following queries return the name of the tablespace and its location.

```
postgres=> show temp_tablespaces;
temp_tablespaces
---------------------
rds_temp_tablespace
(1 row)
```

The `rds_temp_tablespace` is a tablespace configured by RDS that points to the NVMe local storage. You can always switch back to Amazon EBS storage by modifying this parameter in the `Parameter group` using the AWS Management Console to point to any tablespace other than `rds_temp_tablespace`. For more information, see Modifying parameters in a DB parameter group in Amazon RDS. You can also use the SET command to modify the value of the `temp_tablespaces` parameter to `pg_default` at the session level using SET command. Modifying the parameter redirects the temporary work area to Amazon EBS. Switching back to Amazon EBS helps when the local storage for your RDS instance or cluster isn't sufficient to perform a specific SQL operation.

```
postgres=> SET temp_tablespaces TO 'pg_default';
SET
```

```
postgres=> show temp_tablespaces;

 temp_tablespaces
------------------
 pg_default
```

Use cases for RDS Optimized Reads

The following are some use cases that can benefit from Optimized Reads:

- Analytical queries that include Common Table Expressions (CTEs), derived tables, and grouping operations.
- Read replicas that handle the unoptimized queries for an application.
- On-demand or dynamic reporting queries with complex operations such as GROUP BY and ORDER BY that can't always use appropriate indexes.
- Other workloads that use internal temporary tables.
- `CREATE INDEX` or `REINDEX` operations for sorting.

Best practices for RDS Optimized Reads

Use the following best practices for RDS Optimized Reads:

- Add retry logic for read-only queries in case they fail because the instance store is full during the execution.
- Monitor the storage space available on the instance store with the CloudWatch metric `FreeLocalStorage`. If the instance store is reaching its limit because of the workload on the DB instance or Multi-AZ DB cluster, modify it to use a larger DB instance class.

Using RDS Optimized Reads

When you provision an RDS for PostgreSQL DB instance with one of the NVMe based DB instance classes in a Single-AZ DB instance deployment, Multi-AZ DB instance deployment, or Multi-AZ DB cluster deployment, the DB instance automatically uses RDS Optimized Reads.

For more information about Multi-AZ deployment, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

To turn on RDS Optimized Reads, do one of the following:

- Create an RDS for PostgreSQL DB instance or Multi-AZ DB cluster using one of the NVMe based DB instance classes. For more information, see Creating an Amazon RDS DB instance.
- Modify an existing RDS for PostgreSQL DB instance or Multi-AZ DB cluster to use one of the NVMe based DB instance classes. For more information, see Modifying an Amazon RDS DB instance.

RDS Optimized Reads is available in all AWS Regions where one or more of the DB instance classes with local NVMe SSD storage are supported. For more information, see DB instance classes.

To switch back to a non-optimized reads RDS instance, modify the DB instance class of your RDS instance or cluster to the similar instance class that only supports EBS storage for your database workloads. For example, if the current DB instance class is db.r6gd.4xlarge, choose db.r6g.4xlarge to switch back. For more information, see Modifying an Amazon RDS DB instance.

Monitoring DB instances that use RDS Optimized Reads

You can monitor DB instances that use RDS Optimized Reads using the following CloudWatch metrics:

- `FreeLocalStorage`
- `ReadIOPSLocalStorage`
- `ReadLatencyLocalStorage`
- `ReadThroughputLocalStorage`
- `WriteIOPSLocalStorage`
- `WriteLatencyLocalStorage`
- `WriteThroughputLocalStorage`

These metrics provide data about available instance store storage, IOPS, and throughput. For more information about these metrics, see Amazon CloudWatch instance-level metrics for Amazon RDS.

To monitor current usage of your local storage, lo in to your database and run the following query:

```
SELECT
    spcname AS "Name",
    pg_catalog.pg_size_pretty(pg_catalog.pg_tablespace_size(oid)) AS "size"
FROM
    pg_catalog.pg_tablespace
WHERE
    spcname IN ('rds_temp_tablespace');
```

For more information about the temporary files and their usage, see Managing temporary files with PostgreSQL.

Limitations for RDS Optimized Reads in PostgreSQL

The following limitation apply to RDS Optimized Reads in PostgreSQL:

- Transactions can fail when the instance store is full.

Importing data into PostgreSQL on Amazon RDS

Suppose that you have an existing PostgreSQL deployment that you want to move to Amazon RDS. The complexity of your task depends on the size of your database and the types of database objects that you're transferring. For example, consider a database that contains datasets on the order of gigabytes, along with stored procedures and triggers. Such a database is going to be more complicated than a simple database with only a few megabytes of test data and no triggers or stored procedures.

We recommend that you use native PostgreSQL database migration tools under the following conditions:

- You have a homogeneous migration, where you are migrating from a database with the same database engine as the target database.
- You are migrating an entire database.
- The native tools allow you to migrate your system with minimal downtime.

In most other cases, performing a database migration using AWS Database Migration Service (AWS DMS) is the best approach. AWS DMS can migrate databases without downtime and, for many database engines, continue ongoing replication until you are ready to switch over to the target database. You can migrate to either the same database engine or a different database engine using AWS DMS. If you are migrating to a different database engine than your source database, you can use the AWS Schema Conversion Tool (AWS SCT). You use AWS SCT to migrate schema objects that are not migrated by AWS DMS. For more information about AWS DMS, see What is AWS Database Migration Service?

Modify your DB parameter group to include the following settings *for your import only*. You should test the parameter settings to find the most efficient settings for your DB instance size. You also need to revert back to production values for these parameters after your import completes.

Modify your DB instance settings to the following:

- Disable DB instance backups (set backup_retention to 0).
- Disable Multi-AZ.

Modify your DB parameter group to include the following settings. You should only use these settings when importing data. You should test the parameter settings to find the most efficient

settings for your DB instance size. You also need to revert back to production values for these parameters after your import completes.

Parameter	Recommended value when importing	Description
maintenance_work_mem	524288, 1048576, 2097152, or 4194304 (in KB). These settings are comparable to 512 MB, 1 GB, 2 GB, and 4 GB.	The value for this setting depends on the size of your host. This parameter is used during CREATE INDEX statements and each parallel command can use this much memory. Calculate the best value so that you don't set this value so high that you run out of memory.
max_wal_size	256 (for version 9.6), 4096 (for versions 10 and higher)	Maximum size to let the WAL grow during automatic checkpoints. Increasing this parameter can increase the amount of time needed for crash recovery. This parameter replaces checkpoint_segments for PostgreSQL 9.6 and later. For PostgreSQL version 9.6, this value is in 16 MB units. For later versions, the value is in 1 MB units. For example, in version 9.6, 128 means 128 chunks that are each 16 MB in size. In version 12.4, 2048 means 2048 chunks that are each 1 MB in size.
checkpoint_timeout	1800	The value for this setting allows for less frequent WAL rotation.
synchronous_commit	Off	Disable this setting to speed up writes. Turning this parameter off can increase the risk of data loss in the event of a server crash (do not turn off FSYNC).
wal_buffers	8192	This is value is in 8 KB units. This again helps your WAL generation speed

Parameter	Recommended value when importing	Description
`autovacuum`	0	Disable the PostgreSQL auto vacuum parameter while you are loading data so that it doesn't use resources

Use the `pg_dump -Fc` (compressed) or `pg_restore -j` (parallel) commands with these settings.

> **Note**
>
> The PostgreSQL command `pg_dumpall` requires super_user permissions that are not granted when you create a DB instance, so it cannot be used for importing data.

Topics

- Importing a PostgreSQL database from an Amazon EC2 instance
- Using the \copy command to import data to a table on a PostgreSQL DB instance
- Importing data from Amazon S3 into an RDS for PostgreSQL DB instance
- Transporting PostgreSQL databases between DB instances

Importing a PostgreSQL database from an Amazon EC2 instance

If you have data in a PostgreSQL server on an Amazon EC2 instance and want to move it to a PostgreSQL DB instance, you can follow this process to migrate the data.

1. Create a file using pg_dump that contains the data to be loaded
2. Create the target DB instance
3. Use *psql* to create the database on the DB instance and load the data
4. Create a DB snapshot of the DB instance

The following sections provide more details on each step listed above.

Step 1: Create a file using pg_dump that contains the data to load

The pg_dump utility uses the COPY command to create a schema and data dump of a PostgreSQL database. The dump script generated by pg_dump loads data into a database with the same name and recreates the tables, indexes, and foreign keys. You can use the pg_restore command and the -d parameter to restore the data to a database with a different name.

Before you create the data dump, you should query the tables to be dumped to get a row count so you can confirm the count on the target DB instance.

The following command creates a dump file called mydb2dump.sql for a database called mydb2.

```
prompt>pg_dump dbname=mydb2 -f mydb2dump.sql
```

Step 2: Create the target DB instance

Create the target PostgreSQL DB instance using either the Amazon RDS console, AWS CLI, or API. Create the instance with the backup retention setting set to 0 and disable Multi-AZ. Doing so allows faster data import. You must create a database on the instance before you can dump the data. The database can have the same name as the database that is contained the dumped data. Alternatively, you can create a database with a different name. In this case, you use the pg_restore command and the -d parameter to restore the data into the newly named database.

For example, the following commands can be used to dump, restore, and rename a database.

```
pg_dump -Fc -v -h [endpoint of instance] -U [master username] [database]
 > [database].dump
createdb [new database name]
pg_restore -v -h [endpoint of instance] -U [master username] -d [new database
 name] [database].dump
```

Step 3: Use psql to create the database on the DB instance and load data

You can use the same connection you used to run the pg_dump command to connect to the target DB instance and recreate the database. Using *psql*, you can use the master user name and master password to create the database on the DB instance

The following example uses *psql* and a dump file named mydb2dump.sql to create a database called mydb2 on a PostgreSQL DB instance called mypginstance:

For Linux, macOS, or Unix:

```
psql \
   -f mydb2dump.sql \
   --host mypginstance.555555555555.aws-region.rds.amazonaws.com \
   --port 8199 \
   --username myawsuser \
   --password password \
   --dbname mydb2
```

For Windows:

```
psql ^
   -f mydb2dump.sql ^
   --host mypginstance.555555555555.aws-region.rds.amazonaws.com ^
   --port 8199 ^
   --username myawsuser ^
   --password password ^
   --dbname mydb2
```

> **Note**
>
> Specify a password other than the prompt shown here as a security best practice.

Step 4: Create a DB snapshot of the DB instance

Once you have verified that the data was loaded into your DB instance, we recommend that you create a DB snapshot of the target PostgreSQL DB instance. DB snapshots are complete backups of your DB instance that can be used to restore your DB instance to a known state. A DB snapshot taken immediately after the load protects you from having to load the data again in case of a mishap. You can also use such a snapshot to seed new DB instances. For information about creating a DB snapshot, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Using the \copy command to import data to a table on a PostgreSQL DB instance

The PostgreSQL `\copy` command is a meta-command available from the `psql` interactive client tool. You can use `\copy` to import data into a table on your RDS for PostgreSQL DB instance. To

use the \copy command, you need to first create the table structure on the target DB instance so that \copy has a destination for the data being copied.

You can use \copy to load data from a comma-separated values (CSV) file, such as one that's been exported and saved to your client workstation.

To import the CSV data to the target RDS for PostgreSQL DB instance, first connect to the target DB instance using psql.

```
psql --host=db-instance.111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=target-db
```

You then run \copy command with the following parameters to identify the target for the data and its format.

- target_table – The name of the table that should receive the data being copied from the CSV file.
- column_list – Column specifications for the table.
- 'filename' – The complete path to the CSV file on your local workstation.

```
 \copy target_table from '/path/to/local/filename.csv' WITH DELIMITER ',' CSV;
```

If your CSV file has column heading information, you can use this version of the command and parameters.

```
\copy target_table (column-1, column-2, column-3, ...)
    from '/path/to/local/filename.csv' WITH DELIMITER ',' CSV HEADER;
```

If the \copy command fails, PostgreSQL outputs error messages.

Creating a new DB instance in the Database Preview environment using psql command with the \copy meta-command as shown in the following examples. This example uses *source-table* as the source table name, *source-table.csv* as the .csv file, and *target-db* as the target database:

For Linux, macOS, or Unix:

```
$psql target-db \
    -U <admin user> \
```

```
    -p <port> \
    -h <DB instance name> \
    -c "\copy source-table from 'source-table.csv' with DELIMITER ','"
```

For Windows:

```
$psql target-db ^
    -U <admin user> ^
    -p <port> ^
    -h <DB instance name> ^
    -c "\copy source-table from 'source-table.csv' with DELIMITER ','"
```

For complete details about the `\copy` command, see the psql page in the PostgreSQL documentation, in the *Meta-Commands* section.

Importing data from Amazon S3 into an RDS for PostgreSQL DB instance

You can import data that's been stored using Amazon Simple Storage Service into a table on an RDS for PostgreSQL DB instance. To do this, you first install the RDS for PostgreSQL `aws_s3` extension. This extension provides the functions that you use to import data from an Amazon S3 bucket. A *bucket* is an Amazon S3 container for objects and files. The data can be in a comma-separate value (CSV) file, a text file, or a compressed (gzip) file. Following, you can learn how to install the extension and how to import data from Amazon S3 into a table.

Your database must be running PostgreSQL version 10.7 or higher to import from Amazon S3 into RDS for PostgreSQL.

If you don't have data stored on Amazon S3, you need to first create a bucket and store the data. For more information, see the following topics in the *Amazon Simple Storage Service User Guide*.

- Create a bucket
- Add an object to a bucket

Cross-account import from Amazon S3 is supported. For more information, see Granting cross-account permissions in the *Amazon Simple Storage Service User Guide*.

You can use the customer managed key for encryption while importing data from S3. For more information, see KMS keys stored in AWS KMS in the *Amazon Simple Storage Service User Guide*.

Topics

- Installing the aws_s3 extension
- Overview of importing data from Amazon S3 data
- Setting up access to an Amazon S3 bucket
- Importing data from Amazon S3 to your RDS for PostgreSQL DB instance
- Function reference

Installing the aws_s3 extension

Before you can use Amazon S3 with your RDS for PostgreSQL DB instance, you need to install the `aws_s3` extension. This extension provides functions for importing data from an Amazon S3. It also provides functions for exporting data from an RDS for PostgreSQL DB instance to an Amazon S3 bucket. For more information, see Exporting data from an RDS for PostgreSQL DB instance to Amazon S3. The `aws_s3` extension depends on some of the helper functions in the `aws_commons` extension, which is installed automatically when needed.

To install the `aws_s3` extension

1. Use psql (or pgAdmin) to connect to the RDS for PostgreSQL DB instance as a user that has `rds_superuser` privileges. If you kept the default name during the setup process, you connect as `postgres`.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. To install the extension, run the following command.

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. To verify that the extension is installed, you can use the psql `\dx` metacommand.

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                 Description
-------------+---------+------------+---------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
```

```
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

The functions for importing data from Amazon S3 and exporting data to Amazon S3 are now available to use.

Overview of importing data from Amazon S3 data

To import S3 data into Amazon RDS

First, gather the details that you need to supply to the function. These include the name of the table on your RDS for PostgreSQL DB instance, and the bucket name, file path, file type, and AWS Region where the Amazon S3 data is stored. For more information, see View an object in the *Amazon Simple Storage Service User Guide*.

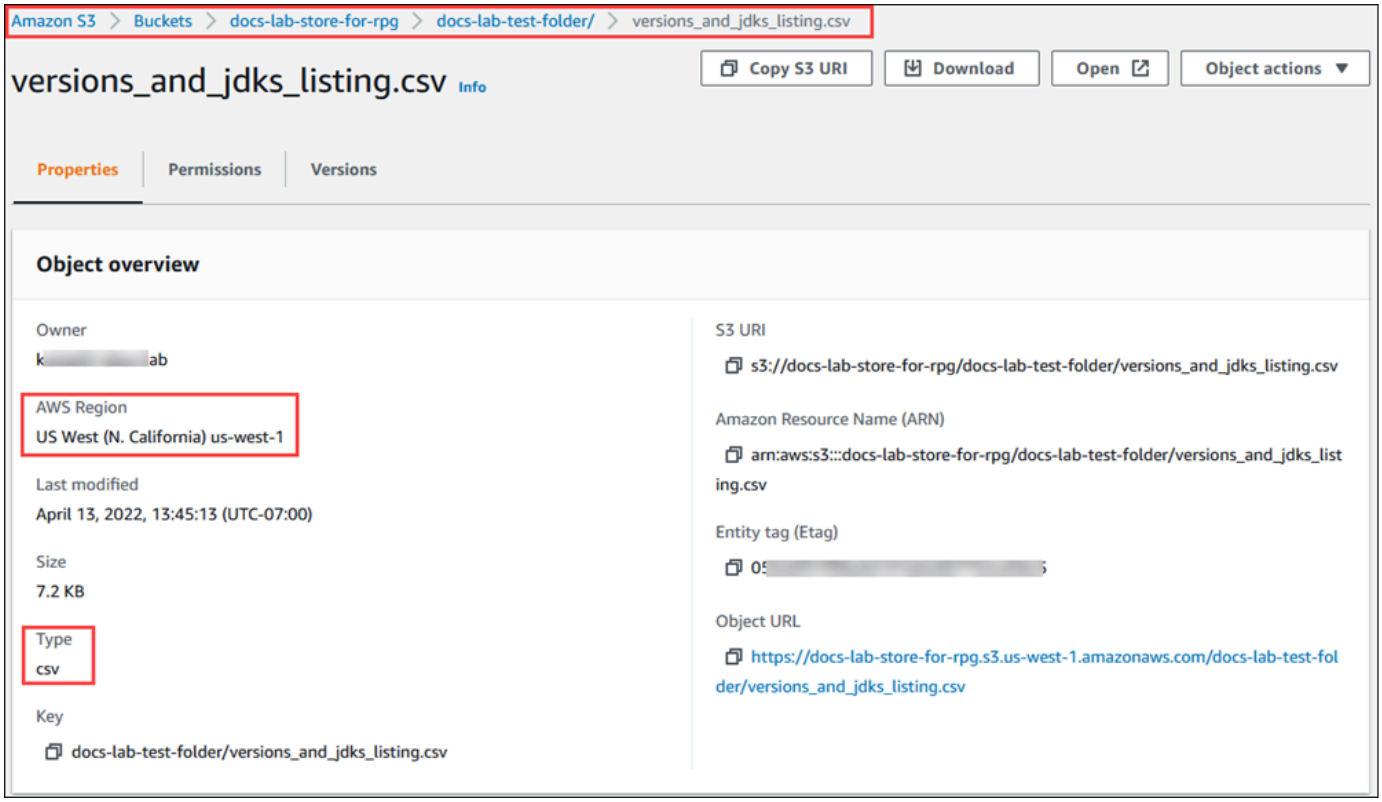
> **Note**
>
> Multi part data import from Amazon S3 isn't currently supported.

1. Get the name of the table into which the `aws_s3.table_import_from_s3` function is to import the data. As an example, the following command creates a table `t1` that can be used in later steps.

   ```
   postgres=> CREATE TABLE t1
       (col1 varchar(80),
       col2 varchar(80),
       col3 varchar(80));
   ```

2. Get the details about the Amazon S3 bucket and the data to import. To do this, open the Amazon S3 console at https://console.aws.amazon.com/s3/, and choose **Buckets**. Find the bucket containing your data in the list. Choose the bucket, open its Object overview page, and then choose Properties.

   Make a note of the bucket name, path, the AWS Region, and file type. You need the Amazon Resource Name (ARN) later, to set up access to Amazon S3 through an IAM role. For more more information, see Setting up access to an Amazon S3 bucket. The image following shows an example.



3. You can verify the path to the data on the Amazon S3 bucket by using the AWS CLI command `aws s3 cp`. If the information is correct, this command downloads a copy of the Amazon S3 file.

```
aws s3 cp s3://amzn-s3-demo-bucket/sample_file_path ./
```

4. Set up permissions on your RDS for PostgreSQL DB instance to allow access to the file on the Amazon S3 bucket. To do so, you use either an AWS Identity and Access Management (IAM) role or security credentials. For more information, see Setting up access to an Amazon S3 bucket.

5. Supply the path and other Amazon S3 object details gathered (see step 2) to the `create_s3_uri` function to construct an Amazon S3 URI object. To learn more about this function, see aws_commons.create_s3_uri. The following is an example of constructing this object during a psql session.

```
postgres=> SELECT aws_commons.create_s3_uri(
   'docs-lab-store-for-rpg',
   'versions_and_jdks_listing.csv',
   'us-west-1'
) AS s3_uri \gset
```

In the next step, you pass this object (`aws_commons._s3_uri_1`) to the `aws_s3.table_import_from_s3` function to import the data to the table.

6. Invoke the `aws_s3.table_import_from_s3` function to import the data from Amazon S3 into your table. For reference information, see aws_s3.table_import_from_s3. For examples, see Importing data from Amazon S3 to your RDS for PostgreSQL DB instance.

Setting up access to an Amazon S3 bucket

To import data from an Amazon S3 file, give the RDS for PostgreSQL DB instance permission to access the Amazon S3 bucket containing the file. You provide access to an Amazon S3 bucket in one of two ways, as described in the following topics.

Topics

- Using an IAM role to access an Amazon S3 bucket
- Using security credentials to access an Amazon S3 bucket
- Troubleshooting access to Amazon S3

Using an IAM role to access an Amazon S3 bucket

Before you load data from an Amazon S3 file, give your RDS for PostgreSQL DB instance permission to access the Amazon S3 bucket the file is in. This way, you don't have to manage additional credential information or provide it in the aws_s3.table_import_from_s3 function call.

To do this, create an IAM policy that provides access to the Amazon S3 bucket. Create an IAM role and attach the policy to the role. Then assign the IAM role to your DB instance.

To give an RDS for PostgreSQL DB instance access to Amazon S3 through an IAM role

1. Create an IAM policy.

   This policy provides the bucket and object permissions that allow your RDS for PostgreSQL DB instance to access Amazon S3.

   Include in the policy the following required actions to allow the transfer of files from an Amazon S3 bucket to Amazon RDS:

   - `s3:GetObject`

- `s3:ListBucket`

Include in the policy the following resources to identify the Amazon S3 bucket and objects in the bucket. This shows the Amazon Resource Name (ARN) format for accessing Amazon S3.

- arn:aws:s3:::*amzn-s3-demo-bucket*
- arn:aws:s3:::*amzn-s3-demo-bucket*/*

For more information on creating an IAM policy for RDS for PostgreSQL, see Creating and using an IAM policy for IAM database access. See also Tutorial: Create and attach your first customer managed policy in the *IAM User Guide*.

The following AWS CLI command creates an IAM policy named `rds-s3-import-policy` with these options. It grants access to a bucket named *amzn-s3-demo-bucket*.

> **Note**
>
> Make a note of the Amazon Resource Name (ARN) of the policy returned by this command. You need the ARN in a subsequent step when you attach the policy to an IAM role.

Example

For Linux, macOS, or Unix:

```
aws iam create-policy \
   --policy-name rds-s3-import-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
```

```
            "arn:aws:s3:::amzn-s3-demo-bucket",
            "arn:aws:s3:::amzn-s3-demo-bucket/*"
          ]
        }
      ]
    }'
```

For Windows:

```
aws iam create-policy ^
   --policy-name rds-s3-import-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::amzn-s3-demo-bucket",
           "arn:aws:s3:::amzn-s3-demo-bucket/*"
         ]
       }
     ]
   }'
```

2. Create an IAM role.

   You do this so Amazon RDS can assume this IAM role to access your Amazon S3 buckets. For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.

   We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based policies to limit the service's permissions to a specific resource. This is the most effective way to protect against the confused deputy problem.

   If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

- Use `aws:SourceArn` if you want cross-service access for a single resource.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

In the policy, be sure to use the `aws:SourceArn` global condition context key with the full ARN of the resource. The following example shows how to do so using the AWS CLI command to create a role named `rds-s3-import-role`.

Example

For Linux, macOS, or Unix:

```
aws iam create-role \
   --role-name rds-s3-import-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
             }
       }
     ]
   }'
```

For Windows:

```
aws iam create-role ^
   --role-name rds-s3-import-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
```

```
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
               "aws:SourceAccount": "111122223333",
               "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
               }
            }
      }
    ]
  }'
```

3. Attach the IAM policy that you created to the IAM role that you created.

   The following AWS CLI command attaches the policy created in the previous step to the role named `rds-s3-import-role` Replace *your-policy-arn* with the policy ARN that you noted in an earlier step.

   Example

   For Linux, macOS, or Unix:

```
aws iam attach-role-policy \
   --policy-arn your-policy-arn \
   --role-name rds-s3-import-role
```

   For Windows:

```
aws iam attach-role-policy ^
   --policy-arn your-policy-arn ^
   --role-name rds-s3-import-role
```

4. Add the IAM role to the DB instance.

   You do so by using the AWS Management Console or AWS CLI, as described following.

Console

To add an IAM role for a PostgreSQL DB instance using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose the PostgreSQL DB instance name to display its details.
3. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the role to add under **Add IAM roles to this instance** .
4. Under **Feature**, choose **s3Import**.
5. Choose **Add role**.

AWS CLI

To add an IAM role for a PostgreSQL DB instance using the CLI

- Use the following command to add the role to the PostgreSQL DB instance named `my-db-instance`. Replace *`your-role-arn`* with the role ARN that you noted in a previous step. Use `s3Import` for the value of the `--feature-name` option.

  Example

  For Linux, macOS, or Unix:

```
aws rds add-role-to-db-instance \
   --db-instance-identifier my-db-instance \
   --feature-name s3Import \
   --role-arn your-role-arn   \
   --region your-region
```

  For Windows:

```
aws rds add-role-to-db-instance ^
   --db-instance-identifier my-db-instance ^
   --feature-name s3Import ^
   --role-arn your-role-arn ^
   --region your-region
```

RDS API

To add an IAM role for a PostgreSQL DB instance using the Amazon RDS API, call the AddRoleToDBInstance operation.

Using security credentials to access an Amazon S3 bucket

If you prefer, you can use security credentials to provide access to an Amazon S3 bucket instead of providing access with an IAM role. You do so by specifying the `credentials` parameter in the aws_s3.table_import_from_s3 function call.

The `credentials` parameter is a structure of type `aws_commons._aws_credentials_1`, which contains AWS credentials. Use the aws_commons.create_aws_credentials function to set the access key and secret key in an `aws_commons._aws_credentials_1` structure, as shown following.

```
postgres=> SELECT aws_commons.create_aws_credentials(
   'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

After creating the `aws_commons._aws_credentials_1` structure, use the aws_s3.table_import_from_s3 function with the `credentials` parameter to import the data, as shown following.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   :'creds'
);
```

Or you can include the aws_commons.create_aws_credentials function call inline within the `aws_s3.table_import_from_s3` function call.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Troubleshooting access to Amazon S3

If you encounter connection problems when attempting to import data from Amazon S3, see the following for recommendations:

- Troubleshooting Amazon RDS identity and access
- Troubleshooting Amazon S3 in the *Amazon Simple Storage Service User Guide*
- Troubleshooting Amazon S3 and IAM in the *IAM User Guide*

Importing data from Amazon S3 to your RDS for PostgreSQL DB instance

You import data from your Amazon S3 bucket by using the `table_import_from_s3` function of the aws_s3 extension. For reference information, see aws_s3.table_import_from_s3.

> **Note**
>
> The following examples use the IAM role method to allow access to the Amazon S3 bucket. Thus, the `aws_s3.table_import_from_s3` function calls don't include credential parameters.

The following shows a typical example.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't1',
   '',
   '(format csv)',
   :'s3_uri'
);
```

The parameters are the following:

- `t1` – The name for the table in the PostgreSQL DB instance to copy the data into.
- `''` – An optional list of columns in the database table. You can use this parameter to indicate which columns of the S3 data go in which table columns. If no columns are specified, all the columns are copied to the table. For an example of using a column list, see Importing an Amazon S3 file that uses a custom delimiter.

- `(format csv)` – PostgreSQL COPY arguments. The copy process uses the arguments and format of the PostgreSQL COPY command to import the data. Choices for format include comma-separated value (CSV) as shown in this example, text, and binary. The default is text.
- `s3_uri` – A structure that contains the information identifying the Amazon S3 file. For an example of using the aws_commons.create_s3_uri function to create an `s3_uri` structure, see Overview of importing data from Amazon S3 data.

For more information about this function, see aws_s3.table_import_from_s3.

The `aws_s3.table_import_from_s3` function returns text. To specify other kinds of files for import from an Amazon S3 bucket, see one of the following examples.

> **Note**
>
> Importing 0 bytes file will cause an error.

Topics

- Importing an Amazon S3 file that uses a custom delimiter
- Importing an Amazon S3 compressed (gzip) file
- Importing an encoded Amazon S3 file

Importing an Amazon S3 file that uses a custom delimiter

The following example shows how to import a file that uses a custom delimiter. It also shows how to control where to put the data in the database table using the `column_list` parameter of the aws_s3.table_import_from_s3 function.

For this example, assume that the following information is organized into pipe-delimited columns in the Amazon S3 file.

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

To import a file that uses a custom delimiter

1. Create a table in the database for the imported data.

```
postgres=> CREATE TABLE test (a text, b text, c text, d text, e text);
```

2. Use the following form of the aws_s3.table_import_from_s3 function to import data from the Amazon S3 file.

   You can include the aws_commons.create_s3_uri function call inline within the `aws_s3.table_import_from_s3` function call to specify the file.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   'test',
   'a,b,d,e',
   'DELIMITER ''|''',
   aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'pipeDelimitedSampleFile', 'us-
east-2')
);
```

The data is now in the table in the following columns.

```
postgres=> SELECT * FROM test;
a | b | c | d | e
---+------+---+---+------+-----------
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

Importing an Amazon S3 compressed (gzip) file

The following example shows how to import a file from Amazon S3 that is compressed with gzip. The file that you import needs to have the following Amazon S3 metadata:

- Key: `Content-Encoding`
- Value: `gzip`

If you upload the file using the AWS Management Console, the metadata is typically applied by the system. For information about uploading files to Amazon S3 using the AWS Management Console, the AWS CLI, or the API, see Uploading objects in the *Amazon Simple Storage Service User Guide*.

For more information about Amazon S3 metadata and details about system-provided metadata, see Editing object metadata in the Amazon S3 console in the *Amazon Simple Storage Service User Guide*.

Import the gzip file into your RDS for PostgreSQL DB instance as shown following.

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_gzip', '', '(format csv)',
 'amzn-s3-demo-bucket', 'test-data.gz', 'us-east-2'
);
```

Importing an encoded Amazon S3 file

The following example shows how to import a file from Amazon S3 that has Windows-1252 encoding.

```
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_table', '', 'encoding ''WIN1252''',
 aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'SampleFile', 'us-east-2')
);
```

Function reference

Functions

- aws_s3.table_import_from_s3
- aws_commons.create_s3_uri
- aws_commons.create_aws_credentials

aws_s3.table_import_from_s3

Imports Amazon S3 data into an Amazon RDS table. The `aws_s3` extension provides the `aws_s3.table_import_from_s3` function. The return value is text.

Syntax

The required parameters are `table_name`, `column_list` and `options`. These identify the database table and specify how the data is copied into the table.

You can also use the following parameters:

- The `s3_info` parameter specifies the Amazon S3 file to import. When you use this parameter, access to Amazon S3 is provided by an IAM role for the PostgreSQL DB instance.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1
)
```

- The `credentials` parameter specifies the credentials to access Amazon S3. When you use this parameter, you don't use an IAM role.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1,
   credentials aws_commons._aws_credentials_1
)
```

Parameters

table_name

A required text string containing the name of the PostgreSQL database table to import the data into.

column_list

A required text string containing an optional list of the PostgreSQL database table columns in which to copy the data. If the string is empty, all columns of the table are used. For an example, see Importing an Amazon S3 file that uses a custom delimiter.

options

A required text string containing arguments for the PostgreSQL COPY command. These arguments specify how the data is to be copied into the PostgreSQL table. For more details, see the PostgreSQL COPY documentation.

s3_info

An `aws_commons._s3_uri_1` composite type containing the following information about the S3 object:

- `bucket` – The name of the Amazon S3 bucket containing the file.
- `file_path` – The Amazon S3 file name including the path of the file.
- `region` – The AWS Region that the file is in. For a listing of AWS Region names and associated values, see Regions, Availability Zones, and Local Zones.

credentials

An `aws_commons._aws_credentials_1` composite type containing the following credentials to use for the import operation:

- Access key
- Secret key
- Session token

For information about creating an `aws_commons._aws_credentials_1` composite structure, see aws_commons.create_aws_credentials.

Alternate syntax

To help with testing, you can use an expanded set of parameters instead of the `s3_info` and `credentials` parameters. Following are additional syntax variations for the `aws_s3.table_import_from_s3` function:

- Instead of using the `s3_info` parameter to identify an Amazon S3 file, use the combination of the `bucket`, `file_path`, and `region` parameters. With this form of the function, access to Amazon S3 is provided by an IAM role on the PostgreSQL DB instance.

```
aws_s3.table_import_from_s3 (
   table_name text,
```

```
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text
)
```

- Instead of using the `credentials` parameter to specify Amazon S3 access, use the combination of the `access_key`, `session_key`, and `session_token` parameters.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text,
   access_key text,
   secret_key text,
   session_token text
)
```

Alternate parameters

bucket

A text string containing the name of the Amazon S3 bucket that contains the file.

file_path

A text string containing the Amazon S3 file name including the path of the file.

region

A text string identifying the AWS Region location of the file. For a listing of AWS Region names and associated values, see Regions, Availability Zones, and Local Zones.

access_key

A text string containing the access key to use for the import operation. The default is NULL.

secret_key

A text string containing the secret key to use for the import operation. The default is NULL.

session_token

(Optional) A text string containing the session key to use for the import operation. The default is NULL.

aws_commons.create_s3_uri

Creates an `aws_commons._s3_uri_1` structure to hold Amazon S3 file information. Use the results of the `aws_commons.create_s3_uri` function in the `s3_info` parameter of the aws_s3.table_import_from_s3 function.

Syntax

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

Parameters

bucket

A required text string containing the Amazon S3 bucket name for the file.

file_path

A required text string containing the Amazon S3 file name including the path of the file.

region

A required text string containing the AWS Region that the file is in. For a listing of AWS Region names and associated values, see Regions, Availability Zones, and Local Zones.

aws_commons.create_aws_credentials

Sets an access key and secret key in an `aws_commons._aws_credentials_1` structure. Use the results of the `aws_commons.create_aws_credentials` function in the `credentials` parameter of the aws_s3.table_import_from_s3 function.

Syntax

```
aws_commons.create_aws_credentials(
```

```
    access_key text,
    secret_key text,
    session_token text
)
```

Parameters

access_key

A required text string containing the access key to use for importing an Amazon S3 file. The default is NULL.

secret_key

A required text string containing the secret key to use for importing an Amazon S3 file. The default is NULL.

session_token

An optional text string containing the session token to use for importing an Amazon S3 file. The default is NULL. If you provide an optional `session_token`, you can use temporary credentials.

Transporting PostgreSQL databases between DB instances

By using PostgreSQL transportable databases for Amazon RDS, you can move a PostgreSQL database between two DB instances. This is a very fast way to migrate large databases between different DB instances. To use this approach, your DB instances must both run the same major version of PostgreSQL.

This capability requires that you install the `pg_transport` extension on both the source and the destination DB instance. The `pg_transport` extension provides a physical transport mechanism that moves the database files with minimal processing. This mechanism moves data much faster than traditional dump and load processes, with less downtime.

> **Note**
>
> PostgreSQL transportable databases are available in RDS for PostgreSQL 11.5 and higher, and RDS for PostgreSQL version 10.10 and higher.

To transport a PostgreSQL DB instance from one RDS for PostgreSQL DB instance to another, you first set up the source and destination instances as detailed in Setting up a DB instance for transport. You can then transport the database by using the function described in Transporting a PostgreSQL database.

Topics

- What happens during database transport
- Limitations for using PostgreSQL transportable databases
- Setting up to transport a PostgreSQL database
- Transporting a PostgreSQL database to the destination from the source
- Transportable databases function reference
- Transportable databases parameter reference

What happens during database transport

The PostgreSQL transportable databases feature uses a pull model to import the database from the source DB instance to the destination. The `transport.import_from_server` function creates the in-transit database on the destination DB instance. The in-transit database is inaccessible on the destination DB instance for the duration of the transport.

When transport begins, all current sessions on the source database are ended. Any databases other than the source database on the source DB instance aren't affected by the transport.

The source database is put into a special read-only mode. While it's in this mode, you can connect to the source database and run read-only queries. However, write-enabled queries and some other types of commands are blocked. Only the specific source database that is being transported is affected by these restrictions.

During transport, you can't restore the destination DB instance to a point in time. This is because the transport isn't transactional and doesn't use the PostgreSQL write-ahead log to record changes. If the destination DB instance has automatic backups enabled, a backup is automatically taken after transport completes. Point-in-time restores are available for times *after* the backup finishes.

If the transport fails, the `pg_transport` extension attempts to undo all changes to the source and destination DB instances. This includes removing the destination's partially transported database.

Depending on the type of failure, the source database might continue to reject write-enabled queries. If this happens, use the following command to allow write-enabled queries.

```
ALTER DATABASE db-name SET default_transaction_read_only = false;
```

Limitations for using PostgreSQL transportable databases

Transportable databases have the following limitations:

- **Read replicas** – You can't use transportable databases on read replicas or parent instances of read replicas.
- **Unsupported column types** – You can't use the `reg` data types in any database tables that you plan to transport with this method. These types depend on system catalog object IDs (OIDs), which often change during transport.
- **Tablespaces** – All source database objects must be in the default `pg_default` tablespace.
- **Compatibility** – Both the source and destination DB instances must run the same major version of PostgreSQL.
- **Extensions** – The source DB instance can have only the `pg_transport` installed.
- **Roles and ACLs** – The source database's access privileges and ownership information aren't carried over to the destination database. All database objects are created and owned by the local destination user of the transport.
- **Concurrent transports** – A single DB instance can support up to 32 concurrent transports, including both imports and exports, if worker processes have been configured properly.
- **RDS for PostgreSQL DB instances only** – PostgreSQL transportable databases are supported on RDS for PostgreSQL DB instances only. You can't use it with on-premises databases or databases running on Amazon EC2.

Setting up to transport a PostgreSQL database

Before you begin, make sure that your RDS for PostgreSQL DB instances meet the following requirements:

- The RDS for PostgreSQL DB instances for source and destination must run the same version of PostgreSQL.
- The destination DB can't have a database of the same name as the source DB that you want to transport.

- The account you use to run the transport needs `rds_superuser` privileges on both the source DB and the destination DB.
- The security group for the source DB instance must allow inbound access from the destination DB instance. This might already be the case if your source and destination DB instances are located in the VPC. For more information about security groups, see Controlling access with security groups.

Transporting databases from a source DB instance to a destination DB instance requires several changes to the DB parameter group associated with each instance. That means that you must create a custom DB parameter group for the source DB instance and create a custom DB parameter group for the destination DB instance.

> **Note**
>
> If your DB instances are already configured using custom DB parameter groups, you can start with step 2 in the following procedure.

To configure the custom DB group parameters for transporting databases

For the following steps, use an account that has `rds_superuser` privileges.

1. If the source and destination DB instances use a default DB parameter group, you need to create a custom DB parameter group using the appropriate version for your instances. You do this so you can change values for several parameters. For more information, see Parameter groups for Amazon RDS.
2. In the custom DB parameter group, change values for the following parameters:
   - `shared_preload_libraries` – Add `pg_transport` to the list of libraries.
   - `pg_transport.num_workers` – The default value is 3. Increase or reduce this value as needed for your database. For a 200 GB database, we recommend no larger than 8. Keep in mind that if you increase the default value for this parameter, you should also increase the value of `max_worker_processes`.
   - `pg_transport.work_mem` – The default value is either 128 MB or 256 MB, depending on the PostgreSQL version. The default setting can typically be left unchanged.
   - `max_worker_processes` – The value of this parameter needs to be set using the following calculation:

```
(3 * pg_transport.num_workers) + 9
```

This value is required on the destination to handle various background worker processes involved in the transport. To learn more about `max_worker_processes`, see Resource Consumption in the PostgreSQL documentation.

For more information about `pg_transport` parameters, see Transportable databases parameter reference.

3. Reboot the source RDS for PostgreSQL DB instance and the destination instance so that the settings for the parameters take effect.
4. Connect to your RDS for PostgreSQL source DB instance.

```
psql --host=source-instance.111122223333.aws-region.rds.amazonaws.com --port=5432
 --username=postgres --password
```

5. Remove extraneous extensions from the public schema of the DB instance. Only the `pg_transport` extension is allowed during the actual transport operation.
6. Install the `pg_transport` extension as follows:

```
postgres=> CREATE EXTENSION pg_transport;
CREATE EXTENSION
```

7. Connect to your RDS for PostgreSQL destination DB instance. Remove any extraneous extensions, and then install the `pg_transport` extension.

```
postgres=> CREATE EXTENSION pg_transport;
CREATE EXTENSION
```

Transporting a PostgreSQL database to the destination from the source

After you complete the process described in Setting up to transport a PostgreSQL database, you can start the transport. To do so, run the `transport.import_from_server` function on the destination DB instance. In the syntax following you can find the function parameters.

```
SELECT transport.import_from_server(
   'source-db-instance-endpoint',
```

```
    source-db-instance-port,
   'source-db-instance-user',
   'source-user-password',
   'source-database-name',
   'destination-user-password',
   false);
```

The `false` value shown in the example tells the function that this is not a dry run. To test your transport setup, you can specify `true` for the `dry_run` option when you call the function, as shown following:

```
postgres=> SELECT transport.import_from_server(
    'docs-lab-source-db.666666666666aws-region.rds.amazonaws.com', 5432,
    'postgres', '********', 'labdb', '******', true);
INFO:  Starting dry-run of import of database "labdb".
INFO:  Created connections to remote database        (took 0.03 seconds).
INFO:  Checked remote cluster compatibility          (took 0.05 seconds).
INFO:  Dry-run complete                          (took 0.08 seconds total).
 import_from_server
--------------------

(1 row)
```

The INFO lines are output because the `pg_transport.timing` parameter is set to its default value, `true`. Set the `dry_run` to `false` when you run the command and the source database is imported to the destination, as shown following:

```
INFO:  Starting import of database "labdb".
INFO:  Created connections to remote database        (took 0.02 seconds).
INFO:  Marked remote database as read only           (took 0.13 seconds).
INFO:  Checked remote cluster compatibility          (took 0.03 seconds).
INFO:  Signaled creation of PITR blackout window     (took 2.01 seconds).
INFO:  Applied remote database schema pre-data       (took 0.50 seconds).
INFO:  Created connections to local cluster          (took 0.01 seconds).
INFO:  Locked down destination database              (took 0.00 seconds).
INFO:  Completed transfer of database files          (took 0.24 seconds).
INFO:  Completed clean up                            (took 1.02 seconds).
INFO:  Physical transport complete               (took 3.97 seconds total).
import_from_server
--------------------
(1 row)
```

This function requires that you provide database user passwords. Thus, we recommend that you change the passwords of the user roles you used after transport is complete. Or, you can use SQL bind variables to create temporary user roles. Use these temporary roles for the transport and then discard the roles afterwards.

If your transport isn't successful, you might see an error message similar to the following:

```
pg_transport.num_workers=8 25% of files transported failed to download file data
```

The "failed to download file data" error message indicates that the number of worker processes isn't set correctly for the size of the database. You might need to increase or decrease the value set for `pg_transport.num_workers`. Each failure reports the percentage of completion, so you can see the impact of your changes. For example, changing the setting from 8 to 4 in one case resulted in the following:

```
pg_transport.num_workers=4 75% of files transported failed to download file data
```

Keep in mind that the `max_worker_processes` parameter is also taken into account during the transport process. In other words, you might need to modify both `pg_transport.num_workers` and `max_worker_processes` to successfully transport the database. The example shown finally worked when the `pg_transport.num_workers` was set to 2:

```
pg_transport.num_workers=2 100% of files transported
```

For more information about the `transport.import_from_server` function and its parameters, see Transportable databases function reference.

Transportable databases function reference

The `transport.import_from_server` function transports a PostgreSQL database by importing it from a source DB instance to a destination DB instance. It does this by using a physical database connection transport mechanism.

Before starting the transport, this function verifies that the source and the destination DB instances are the same version and are compatible for the migration. It also confirms that the destination DB instance has enough space for the source.

Syntax

```
transport.import_from_server(
   host text,
   port int,
   username text,
   password text,
   database text,
   local_password text,
   dry_run bool
)
```

Return Value

None.

Parameters

You can find descriptions of the `transport.import_from_server` function parameters in the following table.

Parameter	Description
`host`	The endpoint of the source DB instance.
`port`	An integer representing the port of the source DB instance. PostgreSQL DB instances often use port 5432.
`username`	The user of the source DB instance. This user must be a member of the `rds_superuser` role.
`password`	The user password of the source DB instance.
`database`	The name of the database in the source DB instance to transport.
`local_password`	The local password of the current user for the destination DB instance. This user must be a member of the `rds_superuser` role.
`dry_run`	An optional Boolean value specifying whether to perform a dry run. The default is `false`, which means the transport proceeds. To confirm compatibility between the source and destination DB instances without performing the actual transport, set `dry_run` to `true`.

Example

For an example, see Transporting a PostgreSQL database to the destination from the source.

Transportable databases parameter reference

Several parameters control the behavior of the `pg_transport` extension. Following, you can find descriptions of these parameters.

`pg_transport.num_workers`

The number of workers to use for the transport process. The default is 3. Valid values are 1–32. Even the largest database transports typically require fewer than 8 workers. The value of this setting on the destination DB instance is used by both destination and source during transport.

`pg_transport.timing`

Specifies whether to report timing information during the transport. The default is `true`, meaning that timing information is reported. We recommend that you leave this parameter set to `true` so you can monitor progress. For example output, see Transporting a PostgreSQL database to the destination from the source.

`pg_transport.work_mem`

The maximum amount of memory to allocate for each worker. The default is 131072 kilobytes (KB) or 262144 KB (256 MB), depending on the PostgreSQL version. The minimum value is 64 megabytes (65536 KB). Valid values are in kilobytes (KBs) as binary base-2 units, where 1 KB = 1024 bytes.

The transport might use less memory than is specified in this parameter. Even large database transports typically require less than 256 MB (262144 KB) of memory per worker.

Exporting data from an RDS for PostgreSQL DB instance to Amazon S3

You can query data from an RDS for PostgreSQL DB instance and export it directly into files stored in an Amazon S3 bucket. To do this, you first install the RDS for PostgreSQL `aws_s3` extension. This extension provides you with the functions that you use to export the results of queries to Amazon S3. Following, you can find out how to install the extension and how to export data to Amazon S3.

> **Note**
>
> Cross-account export to Amazon S3 isn't supported.

All currently available versions of RDS for PostgreSQL support exporting data to Amazon Simple Storage Service. For detailed version information, see Amazon RDS for PostgreSQL updates in the *Amazon RDS for PostgreSQL Release Notes*.

If you don't have a bucket set up for your export, see the following topics the *Amazon Simple Storage Service User Guide*.

- Setting up Amazon S3
- Creating a bucket

By default, the data exported from RDS for PostgreSQL to Amazon S3 uses server-side encryption with an AWS managed key. If you are using bucket encryption, the Amazon S3 bucket must be encrypted with an AWS Key Management Service (AWS KMS) key (SSE-KMS). Currently, buckets encrypted with Amazon S3 managed keys (SSE-S3) are not supported.

> **Note**
>
> You can save DB snapshot data to Amazon S3 using the AWS Management Console, AWS CLI, or Amazon RDS API. For more information, see Exporting DB snapshot data to Amazon S3 for Amazon RDS.

Topics

- Installing the aws_s3 extension
- Overview of exporting data to Amazon S3
- Specifying the Amazon S3 file path to export to
- Setting up access to an Amazon S3 bucket
- Exporting query data using the aws_s3.query_export_to_s3 function
- Function reference
- Troubleshooting access to Amazon S3

Installing the aws_s3 extension

Before you can use Amazon Simple Storage Service with your RDS for PostgreSQL DB instance, you need to install the `aws_s3` extension. This extension provides functions for exporting data from an RDS for PostgreSQL DB instance to an Amazon S3 bucket. It also provides functions for importing data from an Amazon S3. For more information, see Importing data from Amazon S3 into an RDS for PostgreSQL DB instance. The `aws_s3` extension depends on some of the helper functions in the `aws_commons` extension, which is installed automatically when needed.

To install the `aws_s3` extension

1. Use psql (or pgAdmin) to connect to the RDS for PostgreSQL DB instance as a user that has `rds_superuser` privileges. If you kept the default name during the setup process, you connect as `postgres`.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. To install the extension, run the following command.

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. To verify that the extension is installed, you can use the psql `\dx` metacommand.

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                Description
-------------+---------+------------+-------------------------------------------
```

```
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

The functions for importing data from Amazon S3 and exporting data to Amazon S3 are now available to use.

Verify that your RDS for PostgreSQL version supports exports to Amazon S3

You can verify that your RDS for PostgreSQL version supports export to Amazon S3 by using the `describe-db-engine-versions` command. The following example verifies support for version 10.14.

```
aws rds describe-db-engine-versions --region us-east-1
--engine postgres --engine-version 10.14 | grep s3Export
```

If the output includes the string `"s3Export"`, then the engine supports Amazon S3 exports. Otherwise, the engine doesn't support them.

Overview of exporting data to Amazon S3

To export data stored in an RDS for PostgreSQL database to an Amazon S3 bucket, use the following procedure.

To export RDS for PostgreSQL data to S3

1. Identify an Amazon S3 file path to use for exporting data. For details about this process, see Specifying the Amazon S3 file path to export to.
2. Provide permission to access the Amazon S3 bucket.

   To export data to an Amazon S3 file, give the RDS for PostgreSQL DB instance permission to access the Amazon S3 bucket that the export will use for storage. Doing this includes the following steps:

   1. Create an IAM policy that provides access to an Amazon S3 bucket that you want to export to.
   2. Create an IAM role.
   3. Attach the policy you created to the role you created.

4. Add this IAM role to your DB instance.

   For details about this process, see Setting up access to an Amazon S3 bucket.

3. Identify a database query to get the data. Export the query data by calling the `aws_s3.query_export_to_s3` function.

   After you complete the preceding preparation tasks, use the aws_s3.query_export_to_s3 function to export query results to Amazon S3. For details about this process, see Exporting query data using the aws_s3.query_export_to_s3 function.

Specifying the Amazon S3 file path to export to

Specify the following information to identify the location in Amazon S3 where you want to export data to:

- Bucket name – A *bucket* is a container for Amazon S3 objects or files.

  For more information on storing data with Amazon S3, see Creating a bucket and Working with objects in the *Amazon Simple Storage Service User Guide*.

- File path – The file path identifies where the export is stored in the Amazon S3 bucket. The file path consists of the following:
  - An optional path prefix that identifies a virtual folder path.
  - A file prefix that identifies one or more files to be stored. Larger exports are stored in multiple files, each with a maximum size of approximately 6 GB. The additional file names have the same file prefix but with `_part`*XX* appended. The *XX* represents 2, then 3, and so on.

  For example, a file path with an `exports` folder and a `query-1-export` file prefix is `/exports/query-1-export`.

- AWS Region (optional) – The AWS Region where the Amazon S3 bucket is located. If you don't specify an AWS Region value, then Amazon RDS saves your files into Amazon S3 in the same AWS Region as the exporting DB instance.

> **Note**
>
> Currently, the AWS Region must be the same as the region of the exporting DB instance.

For a listing of AWS Region names and associated values, see Regions, Availability Zones, and Local Zones.

To hold the Amazon S3 file information about where the export is to be stored, you can use the aws_commons.create_s3_uri function to create an `aws_commons._s3_uri_1` composite structure as follows.

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

You later provide this `s3_uri_1` value as a parameter in the call to the aws_s3.query_export_to_s3 function. For examples, see Exporting query data using the aws_s3.query_export_to_s3 function.

Setting up access to an Amazon S3 bucket

To export data to Amazon S3, give your PostgreSQL DB instance permission to access the Amazon S3 bucket that the files are to go in.

To do this, use the following procedure.

To give a PostgreSQL DB instance access to Amazon S3 through an IAM role

1. Create an IAM policy.

   This policy provides the bucket and object permissions that allow your PostgreSQL DB instance to access Amazon S3.

   As part of creating this policy, take the following steps:

   a. Include in the policy the following required actions to allow the transfer of files from your PostgreSQL DB instance to an Amazon S3 bucket:

      - `s3:PutObject`
      - `s3:AbortMultipartUpload`

b. Include the Amazon Resource Name (ARN) that identifies the Amazon S3 bucket and objects in the bucket. The ARN format for accessing Amazon S3 is: `arn:aws:s3:::amzn-s3-demo-bucket/*`

For more information on creating an IAM policy for Amazon RDS for PostgreSQL, see Creating and using an IAM policy for IAM database access. See also Tutorial: Create and attach your first customer managed policy in the *IAM User Guide*.

The following AWS CLI command creates an IAM policy named `rds-s3-export-policy` with these options. It grants access to a bucket named *amzn-s3-demo-bucket*.

> **Warning**
>
> We recommend that you set up your database within a private VPC that has endpoint policies configured for accessing specific buckets. For more information, see Using endpoint policies for Amazon S3 in the Amazon VPC User Guide.
> We strongly recommend that you do not create a policy with all-resource access. This access can pose a threat for data security. If you create a policy that gives `S3:PutObject` access to all resources using `"Resource":"*"`, then a user with export privileges can export data to all buckets in your account. In addition, the user can export data to *any publicly writable bucket within your AWS Region*.

After you create the policy, note the Amazon Resource Name (ARN) of the policy. You need the ARN for a subsequent step when you attach the policy to an IAM role.

```
aws iam create-policy  --policy-name rds-s3-export-policy  --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3export",
         "Action": [
           "s3:PutObject*",
           "s3:ListBucket",
           "s3:GetObject*",
           "s3:DeleteObject*",
           "s3:GetBucketLocation",
           "s3:AbortMultipartUpload"
         ],
```

```
        "Effect": "Allow",
        "Resource": [
          "arn:aws:s3:::amzn-s3-demo-bucket/*"
        ]
      }
    ]
  }'
```

2. Create an IAM role.

   You do this so Amazon RDS can assume this IAM role on your behalf to access your Amazon S3 buckets. For more information, see Creating a role to delegate permissions to an IAM user in the *IAM User Guide*.

   We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based policies to limit the service's permissions to a specific resource. This is the most effective way to protect against the confused deputy problem.

   If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

   - Use `aws:SourceArn` if you want cross-service access for a single resource.
   - Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

   In the policy, be sure to use the `aws:SourceArn` global condition context key with the full ARN of the resource. The following example shows how to do so using the AWS CLI command to create a role named `rds-s3-export-role`.

   Example

   For Linux, macOS, or Unix:

```
aws iam create-role  \
    --role-name rds-s3-export-role  \
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
```

```
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
             }
       }
     ]
   }'
```

For Windows:

```
aws iam create-role  ^
    --role-name rds-s3-export-role  ^
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
             }
       }
     ]
   }'
```

3. Attach the IAM policy that you created to the IAM role that you created.

   The following AWS CLI command attaches the policy created earlier to the role named `rds-s3-export-role`. Replace *your-policy-arn* with the policy ARN that you noted in an earlier step.

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

4. Add the IAM role to the DB instance. You do so by using the AWS Management Console or AWS CLI, as described following.

Console

To add an IAM role for a PostgreSQL DB instance using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose the PostgreSQL DB instance name to display its details.
3. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the role to add under **Add IAM roles to this instance**.
4. Under **Feature**, choose **s3Export**.
5. Choose **Add role**.

AWS CLI

To add an IAM role for a PostgreSQL DB instance using the CLI

- Use the following command to add the role to the PostgreSQL DB instance named `my-db-instance`. Replace *`your-role-arn`* with the role ARN that you noted in a previous step. Use `s3Export` for the value of the `--feature-name` option.

  Example

  For Linux, macOS, or Unix:

```
aws rds add-role-to-db-instance \
   --db-instance-identifier my-db-instance \
   --feature-name s3Export \
   --role-arn your-role-arn   \
   --region your-region
```

  For Windows:

```
aws rds add-role-to-db-instance ^
   --db-instance-identifier my-db-instance ^
   --feature-name s3Export ^
   --role-arn your-role-arn ^
   --region your-region
```

Exporting query data using the aws_s3.query_export_to_s3 function

Export your PostgreSQL data to Amazon S3 by calling the aws_s3.query_export_to_s3 function.

Topics

- Prerequisites
- Calling aws_s3.query_export_to_s3
- Exporting to a CSV file that uses a custom delimiter
- Exporting to a binary file with encoding

Prerequisites

Before you use the `aws_s3.query_export_to_s3` function, be sure to complete the following prerequisites:

- Install the required PostgreSQL extensions as described in Overview of exporting data to Amazon S3.
- Determine where to export your data to Amazon S3 as described in Specifying the Amazon S3 file path to export to.
- Make sure that the DB instance has export access to Amazon S3 as described in Setting up access to an Amazon S3 bucket.

The examples following use a database table called `sample_table`. These examples export the data into a bucket called *amzn-s3-demo-bucket*. The example table and data are created with the following SQL statements in psql.

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid,name) VALUES (1, 'Monday'), (2,'Tuesday'), (3,
 'Wednesday');
```

Calling aws_s3.query_export_to_s3

The following shows the basic ways of calling the aws_s3.query_export_to_s3 function.

These examples use the variable `s3_uri_1` to identify a structure that contains the information identifying the Amazon S3 file. Use the aws_commons.create_s3_uri function to create the structure.

```
psql=> SELECT aws_commons.create_s3_uri(
   'amzn-s3-demo-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

Although the parameters vary for the following two `aws_s3.query_export_to_s3` function calls, the results are the same for these examples. All rows of the `sample_table` table are exported into a bucket called *amzn-s3-demo-bucket*.

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1');

psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1', options :='format text');
```

The parameters are described as follows:

- `'SELECT * FROM sample_table'` – The first parameter is a required text string containing an SQL query. The PostgreSQL engine runs this query. The results of the query are copied to the S3 bucket identified in other parameters.
- `:'s3_uri_1'` – This parameter is a structure that identifies the Amazon S3 file. This example uses a variable to identify the previously created structure. You can instead create the structure by including the `aws_commons.create_s3_uri` function call inline within the `aws_s3.query_export_to_s3` function call as follows.

  ```
  SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
     aws_commons.create_s3_uri('amzn-s3-demo-bucket', 'sample-filepath', 'us-west-2')
  );
  ```

- `options :='format text'` – The `options` parameter is an optional text string containing PostgreSQL COPY arguments. The copy process uses the arguments and format of the PostgreSQL COPY command.

If the file specified doesn't exist in the Amazon S3 bucket, it's created. If the file already exists, it's overwritten. The syntax for accessing the exported data in Amazon S3 is the following.

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

Larger exports are stored in multiple files, each with a maximum size of approximately 6 GB. The additional file names have the same file prefix but with `_part`*XX* appended. The *XX* represents 2, then 3, and so on. For example, suppose that you specify the path where you store data files as the following.

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
```

If the export has to create three data files, the Amazon S3 bucket contains the following data files.

```
s3-us-west-2://amzn-s3-demo-bucket/my-prefix
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part2
s3-us-west-2://amzn-s3-demo-bucket/my-prefix_part3
```

For the full reference for this function and additional ways to call it, see aws_s3.query_export_to_s3. For more about accessing files in Amazon S3, see View an object in the *Amazon Simple Storage Service User Guide*.

Exporting to a CSV file that uses a custom delimiter

The following example shows how to call the aws_s3.query_export_to_s3 function to export data to a file that uses a custom delimiter. The example uses arguments of the PostgreSQL COPY command to specify the comma-separated value (CSV) format and a colon (:) delimiter.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format csv, delimiter $$:$$');
```

Exporting to a binary file with encoding

The following example shows how to call the aws_s3.query_export_to_s3 function to export data to a binary file that has Windows-1253 encoding.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format binary, encoding WIN1253');
```

Function reference

Functions

- aws_s3.query_export_to_s3
- aws_commons.create_s3_uri

aws_s3.query_export_to_s3

Exports a PostgreSQL query result to an Amazon S3 bucket. The `aws_s3` extension provides the `aws_s3.query_export_to_s3` function.

The two required parameters are `query` and `s3_info`. These define the query to be exported and identify the Amazon S3 bucket to export to. An optional parameter called `options` provides for defining various export parameters. For examples of using the `aws_s3.query_export_to_s3` function, see Exporting query data using the aws_s3.query_export_to_s3 function.

Syntax

```
aws_s3.query_export_to_s3(
    query text,
    s3_info aws_commons._s3_uri_1,
    options text,
    kms_key text
)
```

Input parameters

query

A required text string containing an SQL query that the PostgreSQL engine runs. The results of this query are copied to an S3 bucket identified in the `s3_info` parameter.

s3_info

An `aws_commons._s3_uri_1` composite type containing the following information about the S3 object:

- `bucket` – The name of the Amazon S3 bucket to contain the file.
- `file_path` – The Amazon S3 file name and path.
- `region` – The AWS Region that the bucket is in. For a listing of AWS Region names and associated values, see Regions, Availability Zones, and Local Zones.

  Currently, this value must be the same AWS Region as that of the exporting DB instance. The default is the AWS Region of the exporting DB instance.

To create an `aws_commons._s3_uri_1` composite structure, see the aws_commons.create_s3_uri function.

options

An optional text string containing arguments for the PostgreSQL `COPY` command. These arguments specify how the data is to be copied when exported. For more details, see the PostgreSQL COPY documentation.

kms_key text

An optional text string containing the customer managed KMS key of the S3 bucket to export the data to.

Alternate input parameters

To help with testing, you can use an expanded set of parameters instead of the `s3_info` parameter. Following are additional syntax variations for the `aws_s3.query_export_to_s3` function.

Instead of using the `s3_info` parameter to identify an Amazon S3 file, use the combination of the `bucket`, `file_path`, and `region` parameters.

```
aws_s3.query_export_to_s3(
    query text,
    bucket text,
    file_path text,
    region text,
    options text,
    kms_key text
)
```

query

A required text string containing an SQL query that the PostgreSQL engine runs. The results of this query are copied to an S3 bucket identified in the `s3_info` parameter.

bucket

A required text string containing the name of the Amazon S3 bucket that contains the file.

file_path

A required text string containing the Amazon S3 file name including the path of the file.

region

An optional text string containing the AWS Region that the bucket is in. For a listing of AWS Region names and associated values, see Regions, Availability Zones, and Local Zones.

Currently, this value must be the same AWS Region as that of the exporting DB instance. The default is the AWS Region of the exporting DB instance.

options

An optional text string containing arguments for the PostgreSQL `COPY` command. These arguments specify how the data is to be copied when exported. For more details, see the PostgreSQL COPY documentation.

kms_key text

An optional text string containing the customer managed KMS key of the S3 bucket to export the data to.

Output parameters

```
aws_s3.query_export_to_s3(
    OUT rows_uploaded bigint,
    OUT files_uploaded bigint,
    OUT bytes_uploaded bigint
)
```

rows_uploaded

The number of table rows that were successfully uploaded to Amazon S3 for the given query.

files_uploaded

The number of files uploaded to Amazon S3. Files are created in sizes of approximately 6 GB. Each additional file created has `_part`*XX* appended to the name. The *XX* represents 2, then 3, and so on as needed.

bytes_uploaded

The total number of bytes uploaded to Amazon S3.

Examples

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-
demo-bucket', 'sample-filepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-
demo-bucket', 'sample-filepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'amzn-s3-
demo-bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

Creates an `aws_commons._s3_uri_1` structure to hold Amazon S3 file information. You use the results of the `aws_commons.create_s3_uri` function in the `s3_info` parameter of the aws_s3.query_export_to_s3 function. For an example of using the `aws_commons.create_s3_uri` function, see Specifying the Amazon S3 file path to export to.

Syntax

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

Input parameters

bucket

A required text string containing the Amazon S3 bucket name for the file.

file_path

A required text string containing the Amazon S3 file name including the path of the file.

region

A required text string containing the AWS Region that the file is in. For a listing of AWS Region names and associated values, see Regions, Availability Zones, and Local Zones.

Troubleshooting access to Amazon S3

If you encounter connection problems when attempting to export data to Amazon S3, first confirm that the outbound access rules for the VPC security group associated with your DB instance permit network connectivity. Specifically, the security group must have a rule that allows the DB instance to send TCP traffic to port 443 and to any IPv4 addresses (0.0.0.0/0). For more information, see Provide access to your DB instance in your VPC by creating a security group.

See also the following for recommendations:

- Troubleshooting Amazon RDS identity and access
- Troubleshooting Amazon S3 in the *Amazon Simple Storage Service User Guide*
- Troubleshooting Amazon S3 and IAM in the *IAM User Guide*

Invoking an AWS Lambda function from an RDS for PostgreSQL DB instance

AWS Lambda is an event-driven compute service that lets you run code without provisioning or managing servers. It's available for use with many AWS services, including RDS for PostgreSQL. For example, you can use Lambda functions to process event notifications from a database, or to load data from files whenever a new file is uploaded to Amazon S3. To learn more about Lambda, see What is AWS Lambda? in the *AWS Lambda Developer Guide.*

Note

Invoking an AWS Lambda function is supported in these RDS for PostgreSQL versions:

- All PostgreSQL 18 versions
- All PostgreSQL 17 versions
- All PostgreSQL 16 versions
- All PostgreSQL 15 versions
- PostgreSQL 14.1 and higher minor versions
- PostgreSQL 13.2 and higher minor versions
- PostgreSQL 12.6 and higher minor versions

Setting up RDS for PostgreSQL to work with Lambda functions is a multi-step process involving AWS Lambda, IAM, your VPC, and your RDS for PostgreSQL DB instance. Following, you can find summaries of the necessary steps.

For more information about Lambda functions, see Getting started with Lambda and AWS Lambda foundations in the *AWS Lambda Developer Guide*.

Topics

- Step 1: Configure your RDS for PostgreSQL DB instance for outbound connections to AWS Lambda
- Step 2: Configure IAM for your RDS for PostgreSQL DB instance and AWS Lambda
- Step 3: Install the aws_lambda extension for an RDS for PostgreSQL DB instance
- Step 4: Use Lambda helper functions with your RDS for PostgreSQL DB instance (Optional)

- Step 5: Invoke a Lambda function from your RDS for PostgreSQL DB instance
- Step 6: Grant other users permission to invoke Lambda functions
- Examples: Invoking Lambda functions from your RDS for PostgreSQL DB instance
- Lambda function error messages
- AWS Lambda function and parameter reference

Step 1: Configure your RDS for PostgreSQL DB instance for outbound connections to AWS Lambda

Lambda functions always run inside an Amazon VPC that's owned by the AWS Lambda service. Lambda applies network access and security rules to this VPC and it maintains and monitors the VPC automatically. Your RDS for PostgreSQL DB instance sends network traffic to the Lambda service's VPC. How you configure this depends on whether your DB instance is public or private.

- **Public RDS for PostgreSQL DB instance** – A DB instance is public if it's located in a public subnet on your VPC, and if the instance's "PubliclyAccessible" property is `true`. To find the value of this property, you can use the describe-db-instances AWS CLI command. Or, you can use the AWS Management Console to open the **Connectivity & security** tab and check that **Publicly accessible** is **Yes**. To verify that the instance is in the public subnet of your VPC, you can use the AWS Management Console or the AWS CLI.

  To set up access to Lambda, you use the AWS Management Console or the AWS CLI to create an outbound rule on your VPC's security group. The outbound rule specifies that TCP can use port 443 to send packets to any IPv4 addresses (0.0.0.0/0).
- **Private RDS for PostgreSQL DB instance** – In this case, the instance's "PubliclyAccessible" property is `false` or it's in a private subnet. To allow the instance to work with Lambda, you can use a Network Address Translation) NAT gateway. For more information, see NAT gateways. Or, you can configure your VPC with a VPC endpoint for Lambda. For more information, see VPC endpoints in the *Amazon VPC User Guide*. The endpoint responds to calls made by your RDS for PostgreSQL DB instance to your Lambda functions. The VPC endpoint uses its own private DNS resolution. RDS for PostgreSQL can't use the Lambda VPC endpoint until you change the value of the `rds.custom_dns_resolution` from its default value of 0 (not enabled) to 1. To do so:
  - Create a custom DB parameter group.
  - Change the value of the `rds.custom_dns_resolution` parameter from its default of `0` to 1.
  - Modify your DB instance to use your custom DB parameter group.

- Reboot the instance to have the modified parameter take effect.

Your VPC can now interact with the AWS Lambda VPC at the network level. Next, you configure the permissions using IAM.

Step 2: Configure IAM for your RDS for PostgreSQL DB instance and AWS Lambda

Invoking Lambda functions from your RDS for PostgreSQL DB instance requires certain privileges. To configure the necessary privileges, we recommend that you create an IAM policy that allows invoking Lambda functions, assign that policy to a role, and then apply the role to your DB instance. This approach gives the DB instance privileges to invoke the specified Lambda function on your behalf. The following steps show you how to do this using the AWS CLI.

To configure IAM permissions for using your Amazon RDS instance with Lambda

1. Use the create-policy AWS CLI command to create an IAM policy that allows your RDS for PostgreSQL DB instance to invoke the specified Lambda function. (The statement ID (Sid) is an optional description for your policy statement and has no effect on usage.) This policy gives your DB instance the minimum permissions needed to invoke the specified Lambda function.

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

   Alternatively, you can use the predefined `AWSLambdaRole` policy that allows you to invoke any of your Lambda functions. For more information, see Identity-based IAM policies for Lambda

2. Use the create-role AWS CLI command to create an IAM role that the policy can assume at runtime.

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. Apply the policy to the role by using the attach-role-policy AWS CLI command.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. Apply the role to your RDS for PostgreSQL DB instance by using the add-role-to-db-instance AWS CLI command. This last step allows your DB instance's database users to invoke Lambda functions.

```
aws rds add-role-to-db-instance \
       --db-instance-identifier my-instance-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

With the VPC and the IAM configurations complete, you can now install the `aws_lambda` extension. (Note that you can install the extension at any time, but until you set up the correct VPC support and IAM privileges, the `aws_lambda` extension adds nothing to your RDS for PostgreSQL DB instance's capabilities.)

Step 3: Install the `aws_lambda` extension for an RDS for PostgreSQL DB instance

To use AWS Lambda with your RDS for PostgreSQL DB instance, add the `aws_lambda` PostgreSQL extension to your RDS for PostgreSQL DB instance. This extension provides your RDS for PostgreSQL DB instance with the ability to call Lambda functions from PostgreSQL.

To install the aws_lambda extension in your RDS for PostgreSQL DB instance

Use the PostgreSQL psql command-line or the pgAdmin tool to connect to your RDS for PostgreSQL DB instance.

1. Connect to your RDS for PostgreSQL DB instance as a user with rds_superuser privileges. The default postgres user is shown in the example.

```
psql -h instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. Install the aws_lambda extension. The aws_commons extension is also required. It provides helper functions to aws_lambda and many other Aurora extensions for PostgreSQL. If it's not already on your RDS for PostgreSQLDB instance, it's installed with aws_lambda as shown following.

```
CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

The aws_lambda extension is installed in your DB instance. You can now create convenience structures for invoking your Lambda functions.

Step 4: Use Lambda helper functions with your RDS for PostgreSQL DB instance (Optional)

You can use the helper functions in the aws_commons extension to prepare entities that you can more easily invoke from PostgreSQL. To do this, you need to have the following information about your Lambda functions:

- **Function name** – The name, Amazon Resource Name (ARN), version, or alias of the Lambda function. The IAM policy created in Step 2: Configure IAM for your instance and Lambda requires the ARN, so we recommend that you use your function's ARN.
- **AWS Region** – (Optional) The AWS Region where the Lambda function is located if it's not in the same Region as your RDS for PostgreSQL DB instance.

To hold the Lambda function name information, you use the aws_commons.create_lambda_function_arn function. This helper function creates an

`aws_commons._lambda_function_arn_1` composite structure with the details needed by the invoke function. Following, you can find three alternative approaches to setting up this composite structure.

```
SELECT aws_commons.create_lambda_function_arn(
   'my-function',
   'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   '111122223333:function:my-function',
   'aws-region'
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   'arn:aws:lambda:aws-region:111122223333:function:my-function'
) AS lambda_arn_1 \gset
```

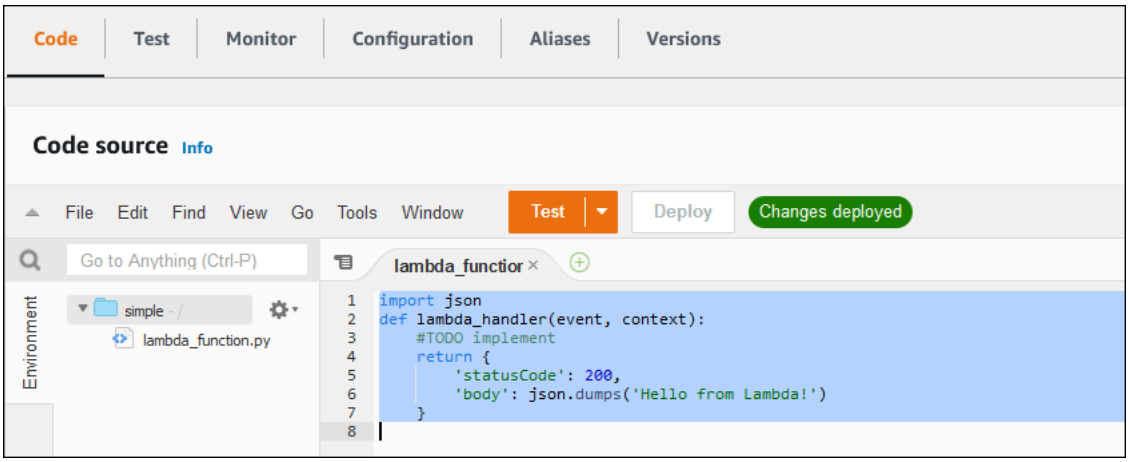
Any of these values can be used in calls to the aws_lambda.invoke function. For examples, see Step 5: Invoke a Lambda function from your RDS for PostgreSQL DB instance.

Step 5: Invoke a Lambda function from your RDS for PostgreSQL DB instance

The `aws_lambda.invoke` function behaves synchronously or asynchronously, depending on the `invocation_type`. The two alternatives for this parameter are `RequestResponse` (the default) and `Event`, as follows.

- **`RequestResponse`** – This invocation type is *synchronous*. It's the default behavior when the call is made without specifying an invocation type. The response payload includes the results of the `aws_lambda.invoke` function. Use this invocation type when your workflow requires receiving results from the Lambda function before proceeding.
- **`Event`** – This invocation type is *asynchronous*. The response doesn't include a payload containing results. Use this invocation type when your workflow doesn't need a result from the Lambda function to continue processing.

As a simple test of your setup, you can connect to your DB instance using `psql` and invoke an example function from the command line. Suppose that you have one of the basic functions set up on your Lambda service, such as the simple Python function shown in the following screenshot.



To invoke an example function

1. Connect to your DB instance using `psql` or pgAdmin.

   ```
   psql -h instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
   ```

2. Invoke the function using its ARN.

   ```
   SELECT * from
    aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
   region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
    Postgres!"}'::json );
   ```

   The response looks as follows.

   ```
   status_code |                         payload                         |
    executed_version | log_result
   -------------+---------------------------------------------------------
   +------------------+------------
            200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
        |
   (1 row)
   ```

If your invocation attempt doesn't succeed, see Lambda function error messages.

Step 6: Grant other users permission to invoke Lambda functions

At this point in the procedures, only you as `rds_superuser` can invoke your Lambda functions. To allow other users to invoke any functions that you create, you need to grant them permission.

To grant others permission to invoke Lambda functions

1. Connect to your DB instance using `psql` or pgAdmin.

```
psql -h instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. Run the following SQL commands:

```
postgres=>  GRANT USAGE ON SCHEMA aws_lambda TO db_username;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
```

Examples: Invoking Lambda functions from your RDS for PostgreSQL DB instance

Following, you can find several examples of calling the aws_lambda.invoke function. Most of the examples use the composite structure `aws_lambda_arn_1` that you create in Step 4: Use Lambda helper functions with your RDS for PostgreSQL DB instance (Optional) to simplify passing the function details. For an example of asynchronous invocation, see Example: Asynchronous (Event) invocation of Lambda functions. All the other examples listed use synchronous invocation.

To learn more about Lambda invocation types, see Invoking Lambda functions in the *AWS Lambda Developer Guide*. For more information about `aws_lambda_arn_1`, see aws_commons.create_lambda_function_arn.

Examples list

- Example: Synchronous (RequestResponse) invocation of Lambda functions
- Example: Asynchronous (Event) invocation of Lambda functions
- Example: Capturing the Lambda execution log in a function response
- Example: Including client context in a Lambda function
- Example: Invoking a specific version of a Lambda function

Example: Synchronous (RequestResponse) invocation of Lambda functions

Following are two examples of a synchronous Lambda function invocation. The results of these `aws_lambda.invoke` function calls are the same.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
```

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse');
```

The parameters are described as follows:

- `:'aws_lambda_arn_1'` – This parameter identifies the composite structure created in Step 4: Use Lambda helper functions with your RDS for PostgreSQL DB instance (Optional), with the `aws_commons.create_lambda_function_arn` helper function. You can also create this structure inline within your `aws_lambda.invoke` call as follows.

  ```
  SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',
   'aws-region'),
  '{"body": "Hello from Postgres!"}'::json
  );
  ```

- `'{"body": "Hello from PostgreSQL!"}'::json` – The JSON payload to pass to the Lambda function.
- `'RequestResponse'` – The Lambda invocation type.

Example: Asynchronous (Event) invocation of Lambda functions

Following is an example of an asynchronous Lambda function invocation. The `Event` invocation type schedules the Lambda function invocation with the specified input payload and returns immediately. Use the `Event` invocation type in certain workflows that don't depend on the results of the Lambda function.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'Event');
```

Example: Capturing the Lambda execution log in a function response

You can include the last 4 KB of the execution log in the function response by using the `log_type` parameter in your `aws_lambda.invoke` function call. By default, this parameter is set to None, but you can specify `Tail` to capture the results of the Lambda execution log in the response, as shown following.

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

Set the aws_lambda.invoke function's `log_type` parameter to `Tail` to include the execution log in the response. The default value for the `log_type` parameter is None.

The `log_result` that's returned is a `base64` encoded string. You can decode the contents using a combination of the `decode` and `convert_from` PostgreSQL functions.

For more information about `log_type`, see aws_lambda.invoke.

Example: Including client context in a Lambda function

The `aws_lambda.invoke` function has a `context` parameter that you can use to pass information separate from the payload, as shown following.

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

To include client context, use a JSON object for the aws_lambda.invoke function's `context` parameter.

For more information about the `context` parameter, see the aws_lambda.invoke reference.

Example: Invoking a specific version of a Lambda function

You can specify a particular version of a Lambda function by including the `qualifier` parameter with the `aws_lambda.invoke` call. Following, you can find an example that does this using '*`custom_version`*' as an alias for the version.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

You can also supply a Lambda function qualifier with the function name details instead, as follows.

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}'::json);
```

For more information about `qualifier` and other parameters, see the aws_lambda.invoke reference.

Lambda function error messages

In the following list you can find information about error messages, with possible causes and solutions.

- **VPC configuration issues**

  VPC configuration issues can raise the following error messages when trying to connect:

  ```
  ERROR:  invoke API failed
  DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
  CONTEXT:  SQL function "invoke" statement 1
  ```

  A common cause for this error is improperly configured VPC security group. Make sure you have an outbound rule for TCP open on port 443 of your VPC security group so that your VPC can connect to the Lambda VPC.

  If your DB instance is private, check the private DNS setup for your VPC. Make sure that you set the `rds.custom_dns_resolution` parameter to 1 and setup AWS PrivateLink as outlined in Step 1: Configure your RDS for PostgreSQL DB instance for outbound connections to AWS Lambda. For more information, see Interface VPC endpoints (AWS PrivateLink).

- **Lack of permissions needed to invoke Lambda functions**

  If you see either of the following error messages, the user (role) invoking the function doesn't have proper permissions.

  ```
  ERROR:  permission denied for schema aws_lambda
  ```

  ```
  ERROR:  permission denied for function invoke
  ```

A user (role) must be given specific grants to invoke Lambda functions. For more information, see Step 6: Grant other users permission to invoke Lambda functions.

- **Improper handling of errors in your Lambda functions**

  If a Lambda function throws an exception during request processing, `aws_lambda.invoke` fails with a PostgreSQL error such as the following.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
ERROR:  lambda invocation failed
DETAIL:  "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
 "Unhandled", details: "<Error details string>".
```

  Be sure to handle errors in your Lambda functions or in your PostgreSQL application.

AWS Lambda function and parameter reference

Following is the reference for the functions and parameters to use for invoking Lambda with RDS for PostgreSQL.

Functions and parameters

- aws_lambda.invoke
- aws_commons.create_lambda_function_arn
- aws_lambda parameters

aws_lambda.invoke

Runs a Lambda function for an RDS for PostgreSQL DB instance.

For more details about invoking Lambda functions, see also Invoke in the *AWS Lambda Developer Guide.*

Syntax

JSON

```
aws_lambda.invoke(
```

```
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
```

```
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

Input parameters

function_name

The identifying name of the Lambda function. The value can be the function name, an ARN, or a partial ARN. For a listing of possible formats, see Lambda function name formats in the *AWS Lambda Developer Guide.*

payload

The input for the Lambda function. The format can be JSON or JSONB. For more information, see JSON Types in the PostgreSQL documentation.

region

(Optional) The Lambda Region for the function. By default, RDS resolves the AWS Region from the full ARN in the `function_name` or it uses the RDS for PostgreSQL DB instance Region. If this Region value conflicts with the one provided in the `function_name` ARN, an error is raised.

invocation_type

The invocation type of the Lambda function. The value is case-sensitive. Possible values include the following:

- `RequestResponse` – The default. This type of invocation for a Lambda function is synchronous and returns a response payload in the result. Use the `RequestResponse` invocation type when your workflow depends on receiving the Lambda function result immediately.
- `Event` – This type of invocation for a Lambda function is asynchronous and returns immediately without a returned payload. Use the `Event` invocation type when you don't need results of the Lambda function before your workflow moves on.
- `DryRun` – This type of invocation tests access without running the Lambda function.

log_type

The type of Lambda log to return in the `log_result` output parameter. The value is case-sensitive. Possible values include the following:

- Tail – The returned `log_result` output parameter will include the last 4 KB of the execution log.
- None – No Lambda log information is returned.

context

Client context in JSON or JSONB format. Fields to use include than `custom` and `env`.

qualifier

A qualifier that identifies a Lambda function's version to be invoked. If this value conflicts with one provided in the `function_name` ARN, an error is raised.

Output parameters

status_code

An HTTP status response code. For more information, see Lambda Invoke response elements in the *AWS Lambda Developer Guide.*

payload

The information returned from the Lambda function that ran. The format is in JSON or JSONB.

executed_version

The version of the Lambda function that ran.

log_result

The execution log information returned if the `log_type` value is `Tail` when the Lambda function was invoked. The result contains the last 4 KB of the execution log encoded in Base64.

aws_commons.create_lambda_function_arn

Creates an `aws_commons._lambda_function_arn_1` structure to hold Lambda function name information. You can use the results of the `aws_commons.create_lambda_function_arn` function in the `function_name` parameter of the aws_lambda.invoke aws_lambda.invoke function.

Syntax

```
aws_commons.create_lambda_function_arn(
    function_name TEXT,
    region TEXT DEFAULT NULL
    )
    RETURNS aws_commons._lambda_function_arn_1
```

Input parameters

function_name

A required text string containing the Lambda function name. The value can be a function name, a partial ARN, or a full ARN.

region

An optional text string containing the AWS Region that the Lambda function is in. For a listing of Region names and associated values, see Regions, Availability Zones, and Local Zones.

aws_lambda parameters

In this table, you can find parameters associated with the `aws_lambda` function.

Parameter	Description
`aws_lambda.connect_timeout_ms`	This is a dynamic parameter and it sets the maximum wait time while connecting to AWS Lambda. The default values is `1000`. Allowed values for this parameter are 1 - 900000.
`aws_lambda.request_timeout_ms`	This is a dynamic parameter and it sets the maximum wait time while waiting for response from AWS Lambda. The default values is `3000`. Allowed values for this parameter are 1 - 900000.
`aws_lambda.endpoint_override`	Specifies the endpoint that can be used to connect to AWS Lambda. An empty string selects the default AWS Lambda endpoint for the region. You must restart the database for this static parameter change to take effect.

Common DBA tasks for Amazon RDS for PostgreSQL

Database administrators (DBAs) perform a variety of tasks when administering an Amazon RDS for PostgreSQL DB instance. If you're a DBA already familiar with PostgreSQL, you need to be aware of some of the important differences between running PostgreSQL on your hardware and RDS for PostgreSQL. For example, because it's a managed service, Amazon RDS doesn't allow shell access to your DB instances. That means that you don't have direct access to `pg_hba.conf` and other configuration files. For RDS for PostgreSQL, changes that are typically made to the PostgreSQL configuration file of an on-premises instance are made to a custom DB parameter group associated with the RDS for PostgreSQL DB instance. For more information, see Parameter groups for Amazon RDS.

You also can't access log files in the same way that you do with an on-premises PostgreSQL instance. To learn more about logging, see RDS for PostgreSQL database log files.

As another example, you don't have access to the PostgreSQL `superuser` account. On RDS for PostgreSQL, the `rds_superuser` role is the most highly privileged role, and it's granted to `postgres` at set up time. Whether you're familiar with using PostgreSQL on-premises or completely new to RDS for PostgreSQL, we recommend that you understand the `rds_superuser` role, and how to work with roles, users, groups, and permissions. For more information, see Understanding PostgreSQL roles and permissions.

Following are some common DBA tasks for RDS for PostgreSQL.

Topics

- Collations supported in RDS for PostgreSQL
- Understanding PostgreSQL roles and permissions
- Dead connection handling in PostgreSQL
- Working with PostgreSQL autovacuum on Amazon RDS for PostgreSQL
- Managing TOAST OID contention in Amazon RDS for PostgreSQL
- Working with logging mechanisms supported by RDS for PostgreSQL
- Managing temporary files with PostgreSQL
- Using pgBadger for log analysis with PostgreSQL
- Using PGSnapper for monitoring PostgreSQL
- Managing custom casts in RDS for PostgreSQL
- Working with parameters on your RDS for PostgreSQL DB instance

Collations supported in RDS for PostgreSQL

Collations are set of rules that determine how character strings stored in the database are sorted and compared. Collations play a fundamental role in the computer system and are included as part of the operating system. Collations change over time when new characters are added to languages or when ordering rules change.

Collation libraries define specific rules and algorithms for a collation. The most popular collation libraries used within PostgreSQL are GNU C (glibc) and Internationalization components for Unicode (ICU). By default, RDS for PostgreSQL uses the glibc collation that includes unicode character sort orders for multi-byte character sequences.

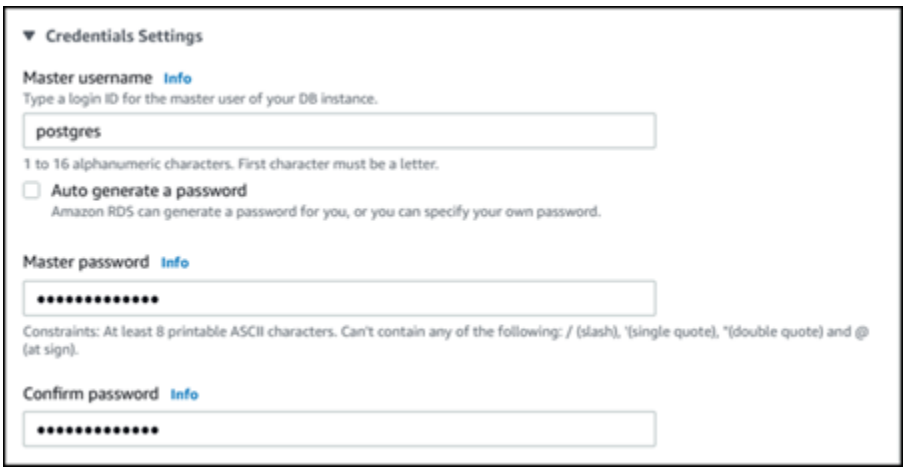
When you create a new DB instance in RDS for PostgreSQL , it checks the operating system for the available collation. The PostgreSQL parameters of the `CREATE DATABASE` command `LC_COLLATE` and `LC_CTYPE` are used to specify a collation, which stands as the default collation in that database. Alternatively, you can also use the `LOCALE` parameter in `CREATE DATABASE` to set these parameters. This determines the default collation for character strings in the database and the rules for classifying characters as letters, numbers, or symbols. You can also choose a collation to use on a column, index, or on a query.

RDS for PostgreSQL depends on the glibc library in the operating system for collation support. RDS for PostgreSQL instance is periodically updated with the latest versions of the operating system. These updates sometimes include a newer version of the glibc library. Rarely, newer versions of glibc change the sort order or collation of some characters, which can cause the data to sort differently or produce invalid index entries. If you discover sort order issues for collation during an update, you might need to rebuild the indexes.

To reduce the possible impacts of the glibc updates, RDS for PostgreSQL now includes an independent default collation library. This collation library is available in RDS for PostgreSQL 14.6, 13.9, 12.13, 11.18, 10.23 and newer minor version releases. It is compatible with glibc 2.26-59.amzn2, and provides sort order stability to prevent incorrect query results.

Understanding PostgreSQL roles and permissions

When you create an RDS for PostgreSQL DB instance using the AWS Management Console, an administrator account is created at the same time. By default, its name is `postgres`, as shown in the following screenshot:



You can choose another name rather than accept the default (`postgres`). If you do, the name you choose must start with a letter and be between 1 and 16 alphanumeric characters. For simplicity's sake, we refer to this main user account by its default value (`postgres`) throughout this guide.

If you use the `create-db-instance` AWS CLI rather than the AWS Management Console, you create the name by passing it with the `master-username` parameter in the command. For more information, see Creating an Amazon RDS DB instance.

Whether you use the AWS Management Console, the AWS CLI, or the Amazon RDS API, and whether you use the default `postgres` name or choose a different name, this first database user account is a member of the `rds_superuser` group and has `rds_superuser` privileges.

Topics

- Understanding the rds_superuser role
- Controlling user access to the PostgreSQL database
- Delegating and controlling user password management
- Using SCRAM for PostgreSQL password encryption

Understanding the rds_superuser role

In PostgreSQL, a *role* can define a user, a group, or a set of specific permissions granted to a group or user for various objects in the database. PostgreSQL commands to `CREATE USER` and `CREATE GROUP` have been replaced by the more general, `CREATE ROLE` with specific properties to distinguish database users. A database user can be thought of as a role with the LOGIN privilege.

> **Note**
>
> The `CREATE USER` and `CREATE GROUP` commands can still be used. For more information, see Database Roles in the PostgreSQL documentation.

The `postgres` user is the most highly privileged database user on your RDS for PostgreSQL DB instance. It has the characteristics defined by the following `CREATE ROLE` statement.

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION
 VALID UNTIL 'infinity'
```

The properties `NOSUPERUSER`, `NOREPLICATION`, `INHERIT`, and `VALID UNTIL 'infinity'` are the default options for CREATE ROLE, unless otherwise specified.

By default, `postgres` has privileges granted to the `rds_superuser` role, and permissions to create roles and databases. The `rds_superuser` role allows the `postgres` user to do the following:

- Add extensions that are available for use with Amazon RDS. For more information, see Working with PostgreSQL features supported by Amazon RDS for PostgreSQL
- Create roles for users and grant privileges to users. For more information, see CREATE ROLE and GRANT in the PostgreSQL documentation.
- Create databases. For more information, see CREATE DATABASE in the PostgreSQL documentation.
- Grant `rds_superuser` privileges to user roles that don't have these privileges, and revoke privileges as needed. We recommend that you grant this role only to those users who perform superuser tasks. In other words, you can grant this role to database administrators (DBAs) or system administrators.
- Grant (and revoke) the `rds_replication` role to database users that don't have the `rds_superuser` role.
- Grant (and revoke) the `rds_password` role to database users that don't have the `rds_superuser` role.
- Obtain status information about all database connections by using the `pg_stat_activity` view. When needed, `rds_superuser` can stop any connections by using `pg_terminate_backend` or `pg_cancel_backend`.

In the `CREATE ROLE postgres...` statement, you can see that the `postgres` user role specifically disallows PostgreSQL `superuser` permissions. RDS for PostgreSQL is a managed service, so you can't access the host OS, and you can't connect using the PostgreSQL `superuser` account. Many of the tasks that require `superuser` access on a stand-alone PostgreSQL are managed automatically by Amazon RDS.

For more information about granting privileges, see GRANT in the PostgreSQL documentation.

The `rds_superuser` role is one of several *predefined* roles in an RDS for PostgreSQL DB instance.

> **Note**
>
> In PostgreSQL 13 and earlier releases, *predefined* roles are known as *default* roles.

In the following list, you find some of the other predefined roles that are created automatically for a new RDS for PostgreSQL DB instance. Predefined roles and their privileges can't be changed. You can't drop, rename, or modify privileges for these predefined roles. Attempting to do so results in an error.

- **rds_password** – A role that can change passwords and set up password constraints for database users. The `rds_superuser` role is granted with this role by default, and can grant the role to database users. For more information, see Controlling user access to the PostgreSQL database.
  - For RDS for PostgreSQL versions older than 14, `rds_password` role can change passwords and set up password constraints for database users and users with `rds_superuser` role. From RDS for PostgreSQL version 14 and later, `rds_password` role can change passwords and set up password constraints only for database users. Only users with `rds_superuser` role can perform these actions on other users with `rds_superuser` role.
- **rdsadmin** – A role that's created to handle many of the management tasks that the administrator with `superuser` privileges would perform on a standalone PostgreSQL database. This role is used internally by RDS for PostgreSQL for many management tasks.
- **rdstopmgr** – A role that's used internally by Amazon RDS to support Multi-AZ deployments.
- **rds_reserved** – A role that's used internally by Amazon RDS to reserve database connections.

Viewing roles and their privileges

You can view predefined roles and their privileges in your RDS for PostgreSQL DB instance using different commands depending on your PostgreSQL version. To see all predefined roles, you can connect to your RDS for PostgreSQL DB instance and run following commands using the `psql`.

For `psql` 15 and earlier versions

Connect to your RDS for PostgreSQL DB instance and use the \du command in psql:

```
postgres=> \du
                                                          List of roles
    Role name    |                         Attributes                          |
                 Member of
-----------------+-----------------------------------------------------------------
+-----------------------------------------------------
 postgres        | Create role, Create DB                                      +|
 {rds_superuser}
                 | Password valid until infinity                               |
 rds_ad          | Cannot login                                                | {}
 rds_iam         | Cannot login                                                | {}
 rds_password    | Cannot login                                                | {}
 rds_replication | Cannot login                                                | {}
 rds_superuser   | Cannot login                                                |
 {pg_monitor,pg_signal_backend,rds_password,rds_replication}
 rdsadmin        | Superuser, Create role, Create DB, Replication, Bypass RLS+| {}
                 | Password valid until infinity                               |
```

For `psql` 16 and later versions

```
postgres=> \drg+
                            List of role grants
   Role name   |          Member of          |       Options       | Grantor
---------------+-----------------------------+---------------------+----------
 postgres      | rds_superuser               | INHERIT, SET        | rdsadmin
 rds_superuser | pg_checkpoint               | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_monitor                  | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_signal_backend           | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | pg_use_reserved_connections | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_password                | ADMIN, INHERIT, SET | rdsadmin
 rds_superuser | rds_replication             | ADMIN, INHERIT, SET | rdsadmin
```

To check role membership without version dependency, you can use the following SQL query:

```
SELECT m.rolname AS "Role name", r.rolname AS "Member of"
FROM pg_catalog.pg_roles m
JOIN pg_catalog.pg_auth_members pam ON (pam.member = m.oid)
LEFT JOIN pg_catalog.pg_roles r ON (pam.roleid = r.oid)
LEFT JOIN pg_catalog.pg_roles g ON (pam.grantor = g.oid)
WHERE m.rolname !~ '^pg_'
ORDER BY 1, 2;
```

In the output, you can see that `rds_superuser` isn't a database user role (it can't login), but it has the privileges of many other roles. You can also see that database user `postgres` is a member of the `rds_superuser` role. As mentioned previously, `postgres` is the default value in the Amazon RDS console's **Create database** page. If you chose another name, that name is shown in the list of roles instead.

Controlling user access to the PostgreSQL database

New databases in PostgreSQL are always created with a default set of privileges in the database's `public` schema that allow all database users and roles to create objects. These privileges allow database users to connect to the database, for example, and create temporary tables while connected.

To better control user access to the databases instances that you create on your RDS for PostgreSQL DB instance, we recommend that you revoke these default `public` privileges. After doing so, you then grant specific privileges for database users on a more granular basis, as shown in the following procedure.

To set up roles and privileges for a new database instance

Suppose you're setting up a database on a newly created RDS for PostgreSQL DB instance for use by several researchers, all of whom need read-write access to the database.

1. Use `psql` (or pgAdmin) to connect to your RDS for PostgreSQL DB instance:

   ```
   psql --host=your-db-instance.666666666666.aws-region.rds.amazonaws.com --port=5432
    --username=postgres --password
   ```

   When prompted, enter your password. The `psql` client connects and displays the default administrative connection database, `postgres=>`, as the prompt.
2. To prevent database users from creating objects in the `public` schema, do the following:

```
postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE
```

3. Next, you create a new database instance:

```
postgres=> CREATE DATABASE lab_db;
CREATE DATABASE
```

4. Revoke all privileges from the PUBLIC schema on this new database.

```
postgres=> REVOKE ALL ON DATABASE lab_db FROM public;
REVOKE
```

5. Create a role for database users.

```
postgres=> CREATE ROLE lab_tech;
CREATE ROLE
```

6. Give database users that have this role the ability to connect to the database.

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;
GRANT
```

7. Grant all users with the `lab_tech` role all privileges on this database.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;
GRANT
```

8. Create database users, as follows:

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. Grant these two users the privileges associated with the lab_tech role:

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

At this point, `lab_user1` and `lab_user2` can connect to the `lab_db` database. This example doesn't follow best practices for enterprise usage, which might include creating multiple database instances, different schemas, and granting limited permissions. For more complete information and additional scenarios, see Managing PostgreSQL Users and Roles.

For more information about privileges in PostgreSQL databases, see the GRANT command in the PostgreSQL documentation.

Delegating and controlling user password management

As a DBA, you might want to delegate the management of user passwords. Or, you might want to prevent database users from changing their passwords or reconfiguring password constraints, such as password lifetime. To ensure that only the database users that you choose can change password settings, you can turn on the restricted password management feature. When you activate this feature, only those database users that have been granted the `rds_password` role can manage passwords.

> **Note**
>
> To use restricted password management, your RDS for PostgreSQL DB instance must be running PostgreSQL 10.6 or higher.

By default, this feature is `off`, as shown in the following:

```
postgres=> SHOW rds.restrict_password_commands;
  rds.restrict_password_commands
--------------------------------
 off
(1 row)
```

To turn on this feature, you use a custom parameter group and change the setting for `rds.restrict_password_commands` to 1. Be sure to reboot your RDS for PostgreSQL DB instance so that the setting takes effect.

With this feature active, `rds_password` privileges are needed for the following SQL commands:

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
```

```
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

Renaming a role (`ALTER ROLE myrole RENAME TO newname`) is also restricted if the password uses the MD5 hashing algorithm.

With this feature active, attempting any of these SQL commands without the `rds_password` role permissions generates the following error:

```
ERROR: must be a member of rds_password to alter passwords
```

We recommend that you grant the `rds_password` to only a few roles that you use solely for password management. If you grant `rds_password` privileges to database users that don't have `rds_superuser` privileges, you need to also grant them the `CREATEROLE` attribute.

Make sure that you verify password requirements such as expiration and needed complexity on the client side. If you use your own client-side utility for password related changes, the utility needs to be a member of `rds_password` and have `CREATE ROLE` privileges.

Using SCRAM for PostgreSQL password encryption

The *Salted Challenge Response Authentication Mechanism (SCRAM)* is an alternative to PostgreSQL's default message digest (MD5) algorithm for encrypting passwords. The SCRAM authentication mechanism is considered more secure than MD5. To learn more about these two different approaches to securing passwords, see Password Authentication in the PostgreSQL documentation.

We recommend that you use SCRAM rather than MD5 as the password encryption scheme for your RDS for PostgreSQL DB instance. It's a cryptographic challenge-response mechanism that uses the scram-sha-256 algorithm for password authentication and encryption.

You might need to update libraries for your client applications to support SCRAM. For example, JDBC versions before 42.2.0 don't support SCRAM. For more information, see PostgreSQL JDBC Driver in the PostgreSQL JDBC Driver documentation. For a list of other PostgreSQL drivers and SCRAM support, see List of drivers in the PostgreSQL documentation.

RDS for PostgreSQL version 13.1 and higher support scram-sha-256. These versions also let you configure your DB instance to require SCRAM, as discussed in the following procedures.

Setting up RDS for PostgreSQL DB instance to require SCRAM

you can require the RDS for PostgreSQL DB instance to accept only passwords that use the scram-sha-256 algorithm.

> **Important**
>
> For existing RDS Proxies with PostgreSQL databases, if you modify the database authentication to use SCRAM only, the proxy becomes unavailable for up to 60 seconds. To avoid the issue, do one of the following:
>
> - Ensure that the database allows both SCRAM and MD5 authentication.
> - To use only SCRAM authentication, create a new proxy, migrate your application traffic to the new proxy, then delete the proxy previously associated with the database.

Before making changes to your system, be sure you understand the complete process, as follows:

- Get information about all roles and password encryption for all database users.
- Double-check the parameter settings for your RDS for PostgreSQL DB instance for the parameters that control password encryption.
- If your RDS for PostgreSQL DB instance uses a default parameter group, you need to create a custom DB parameter group and apply it to your RDS for PostgreSQL DB instance so that you can modify parameters when needed. If your RDS for PostgreSQL DB instance uses a custom parameter group, you can modify the necessary parameters later in the process, as needed.
- Change the `password_encryption` parameter to `scram-sha-256`.
- Notify all database users that they need to update their passwords. Do the same for your `postgres` account. The new passwords are encrypted and stored using the scram-sha-256 algorithm.
- Verify that all passwords are encrypted using as the type of encryption.
- If all passwords use scram-sha-256, you can change the `rds.accepted_password_auth_method` parameter from `md5+scram` to `scram-sha-256`.

Warning

After you change `rds.accepted_password_auth_method` to scram-sha-256 alone, any users (roles) with md5–encrypted passwords can't connect.

Getting ready to require SCRAM for your RDS for PostgreSQL DB instance

Before making any changes to your RDS for PostgreSQL DB instance, check all existing database user accounts. Also, check the type of encryption used for passwords. You can do these tasks by using the `rds_tools` extension. To see which PostgreSQL versions support `rds_tools`, see Extension versions for Amazon RDS for PostgreSQL.

To get a list of database users (roles) and password encryption methods

1. Use `psql` to connect to your RDS for PostgreSQL DB instance, as shown in the following.

```
psql --host=db-name.111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. Install the `rds_tools` extension.

```
postgres=> CREATE EXTENSION rds_tools;
CREATE EXTENSION
```

3. Get a listing of roles and encryption.

```
postgres=> SELECT * FROM
      rds_tools.role_password_encryption_type();
```

You see output similar to the following.

```
       rolname        | encryption_type
----------------------+-----------------
 pg_monitor           |
 pg_read_all_settings |
 pg_read_all_stats    |
 pg_stat_scan_tables  |
 pg_signal_backend    |
 lab_tester           | md5
 user_465             | md5
```

```
 postgres             | md5
(8 rows)
```

Creating a custom DB parameter group

> **Note**
>
> If your RDS for PostgreSQL DB instance already uses a custom parameter group, you don't need to create a new one.

For an overview of parameter groups for Amazon RDS, see Working with parameters on your RDS for PostgreSQL DB instance.

The password encryption type used for passwords is set in one parameter, `password_encryption`. The encryption that the RDS for PostgreSQL DB instance allows is set in another parameter, `rds.accepted_password_auth_method`. Changing either of these from the default values requires that you create a custom DB parameter group and apply it to your instance.

You can also use the AWS Management Console or the RDS API to create a custom DB parameter group. For more information, see

You can now associate the custom parameter group with your DB instance.

To create a custom DB parameter group

1. Use the `create-db-parameter-group` CLI command to create the custom DB parameter group. This example uses `postgres13` as the source for this custom parameter group.

   For Linux, macOS, or Unix:

   ```
   aws rds create-db-parameter-group --db-parameter-group-name 'docs-lab-scram-passwords' \
     --db-parameter-group-family postgres13  --description 'Custom parameter group for SCRAM'
   ```

   For Windows:

   ```
   aws rds create-db-parameter-group --db-parameter-group-name "docs-lab-scram-passwords" ^
   ```

```
  --db-parameter-group-family postgres13  --description "Custom DB parameter group
for SCRAM"
```

2. Use the modify-db-instance CLI command to apply this custom parameter group to your RDS for PostgreSQL DB cluster.

   For Linux, macOS, or Unix:

   ```
   aws rds modify-db-instance --db-instance-identifier 'your-instance-name' \
           --db-parameter-group-name "docs-lab-scram-passwords
   ```

   For Windows:

   ```
   aws rds modify-db-instance --db-instance-identifier "your-instance-name" ^
           --db-parameter-group-name "docs-lab-scram-passwords
   ```

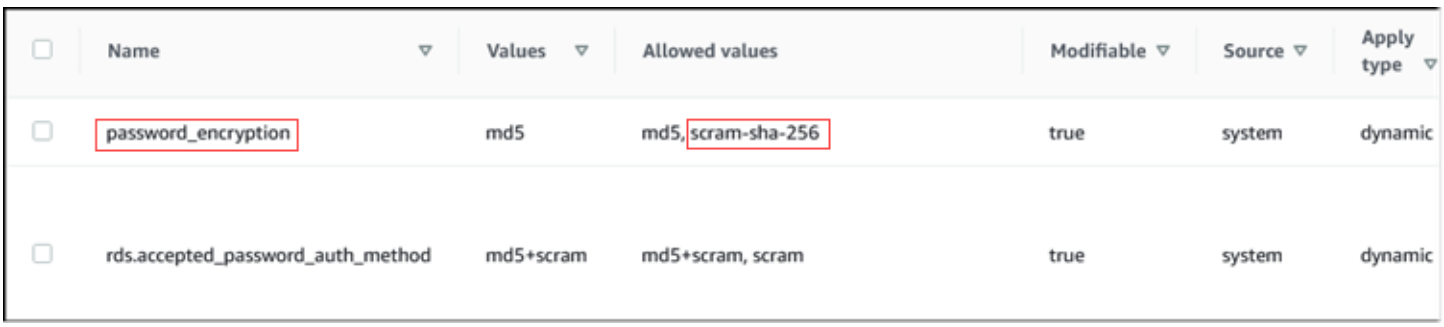
   To resynchronize your RDS for PostgreSQL DB instance with your custom DB parameter group, you need to reboot the primary and all other instances of the cluster. To minimize impact to your users, schedule this to occur during your regular maintenance window.

Configuring password encryption to use SCRAM

The password encryption mechanism used by an RDS for PostgreSQL DB instance is set in the DB parameter group in the `password_encryption` parameter. Allowed values are unset, `md5`, or `scram-sha-256`. The default value depends on the RDS for PostgreSQL version, as follows:

- RDS for PostgreSQL 14 and above – Default is `scram-sha-256`
- RDS for PostgreSQL 13 – Default is `md5`

With a custom DB parameter group attached to your RDS for PostgreSQL DB instance, you can modify values for the password encryption parameter.



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	md5	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

To change password encryption setting to scram-sha-256

- Change the value of password encryption to scram-sha-256, as shown following. The change can be applied immediately because the parameter is dynamic, so a restart isn't required for the change to take effect.

  For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group --db-parameter-group-name \
  'docs-lab-scram-passwords' --parameters
 'ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate'
```

  For Windows:

```
aws rds modify-db-parameter-group --db-parameter-group-name ^
  "docs-lab-scram-passwords" --parameters
 "ParameterName=password_encryption,ParameterValue=scram-
sha-256,ApplyMethod=immediate"
```

Migrating passwords for user roles to SCRAM

You can migrate passwords for user roles to SCRAM as described following.

To migrate database user (role) passwords from MD5 to SCRAM

1. Log in as the administrator user (default user name, `postgres`) as shown following.

```
psql --host=db-name.111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. Check the setting of the `password_encryption` parameter on your RDS for PostgreSQL DB instance by using the following command.

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 md5
 (1 row)
```

3. Change the value of this parameter to scram-sha-256. For more information, see [Configuring password encryption to use SCRAM].
4. Check the value again to make sure that it's now set to `scram-sha-256`, as follows.

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 scram-sha-256
 (1 row)
```

5. Notify all database users to change their passwords. Be sure to also change your own password for account `postgres` (the database user with `rds_superuser` privileges).

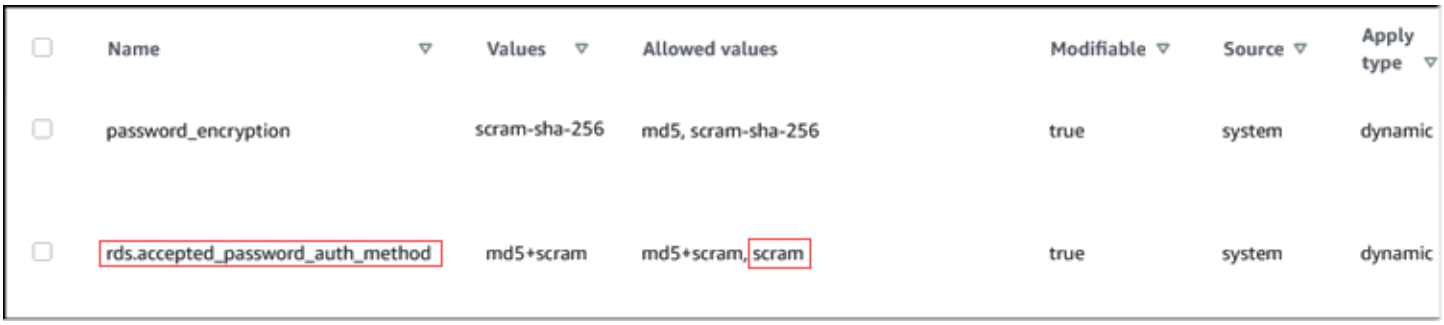
```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

6. Repeat the process for all databases on your RDS for PostgreSQL DB instance.

Changing parameter to require SCRAM

This is the final step in the process. After you make the change in the following procedure, any user accounts (roles) that still use `md5` encryption for passwords can't log in to the RDS for PostgreSQL DB instance.

The `rds.accepted_password_auth_method` specifies the encryption method that the RDS for PostgreSQL DB instance accepts for a user password during the login process. The default value is `md5+scram`, meaning that either method is accepted. In the following image, you can find the default setting for this parameter.



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

The allowed values for this parameter are `md5+scram` or `scram` alone. Changing this parameter value to `scram` makes this a requirement.

To change the parameter value to require SCRAM authentication for passwords

1. Verify that all database user passwords for all databases on your RDS for PostgreSQL DB instance use `scram-sha-256` for password encryption. To do so, query `rds_tools` for the role (user) and encryption type, as follows.

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
  rolname        | encryption_type
  ---------------------+----------------
  pg_monitor           |
  pg_read_all_settings |
  pg_read_all_stats    |
  pg_stat_scan_tables  |
  pg_signal_backend    |
  lab_tester           | scram-sha-256
  user_465             | scram-sha-256
  postgres             | scram-sha-256
  ( rows)
```

2. Repeat the query across all DB instances in your RDS for PostgreSQL DB instance.

   If all passwords use scram-sha-256, you can proceed.

3. Change the value of the accepted password authentication to scram-sha-256, as follows.

   For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group --db-parameter-group-name 'docs-lab-scram-
passwords' \
  --parameters
 'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

   For Windows:

```
aws rds modify-db-parameter-group --db-parameter-group-name "docs-lab-scram-
passwords" ^
  --parameters
 "ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

Dead connection handling in PostgreSQL

Dead connections occur when a database session remains active on the server despite the client application having abandoned or terminated abnormally. This situation typically arises when client processes crash or terminate unexpectedly without properly closing their database connections or canceling ongoing requests.

PostgreSQL efficiently identifies and cleans up dead connections when server processes are idle or attempt to send data to clients. However, detection is challenging for sessions that are idle, waiting for client input, or actively running queries. To handle these scenarios, PostgreSQL provides `tcp_keepalives_*`, `tcp_user_timeout`, and `client_connection_check_interval` parameters.

Topics

- Understanding TCP keepalive
- Key TCP keepalive parameters in RDS for PostgreSQL
- Use cases for TCP keepalive settings
- Best practices

Understanding TCP keepalive

TCP Keepalive is a protocol-level mechanism that helps maintain and verify connection integrity. Each TCP connection maintains kernel-level settings that govern keepalive behavior. When the keepalive timer expires, the system does the following:

- Sends a probe packet with no data and the ACK flag set.
- Expects a response from the remote endpoint according to TCP/IP specifications.
- Manages connection state based on the response or lack thereof.

Key TCP keepalive parameters in RDS for PostgreSQL

Parameter	Description	Default values
`tcp_keepalives_idle`	Specifies number of seconds of inactivity before sending keepalive message.	300

Parameter	Description	Default values
tcp_keepalives_interval	Specifies number of seconds between retransmissions of unacknowledged keepalive messages.	30
tcp_keepalives_count	Maximum lost keepalive messages before declaring connection dead	2
tcp_user_timeout	Specifies how long (in Milliseconds) unacknowledged data can remain before forcibly closing the connection.	0
client_connection_check_interval	Sets the interval (in Milliseconds) for checking client connection status during long-running queries. This ensures quicker detection of closed connections.	0

Use cases for TCP keepalive settings

Keeping idle sessions alive

To prevent idle connections from being terminated by firewalls or routers due to inactivity:

- Configure `tcp_keepalives_idle` to send keepalive packets at regular intervals.

Detecting dead connections

To detect dead connections promptly:

- Adjust `tcp_keepalives_idle`, `tcp_keepalives_interval`, and `tcp_keepalives_count`. For example, with Aurora PostgreSQL defaults, it takes about a minute (2 probes × 30 seconds) to detect a dead connection. Lowering these values can speed up detection.
- Use `tcp_user_timeout` to specify the maximum wait time for an acknowledgment.

TCP keepalive settings help the kernel detect dead connections, but PostgreSQL may not act until the socket is used. If a session is running a long query, dead connections might only be detected after query completion. In PostgreSQL 14 and higher versions, `client_connection_check_interval` can expedite dead connection detection by periodically polling the socket during query execution.

Best practices

- **Set reasonable keepalive intervals:** Tune `tcp_user_timeout`, `tcp_keepalives_idle`, `tcp_keepalives_count` and `tcp_keepalives_interval` to balance detection speed and resource use.
- **Optimize for your environment:** Align settings with network behavior, firewall policies, and session needs.
- **Leverage PostgreSQL features:** Use `client_connection_check_interval` in PostgreSQL 14 and higher versions for efficient connection checks.

Working with PostgreSQL autovacuum on Amazon RDS for PostgreSQL

We strongly recommend that you use the autovacuum feature to maintain the health of your PostgreSQL DB instance. Autovacuum automates the start of the VACUUM and the ANALYZE commands. It checks for tables with a large number of inserted, updated, or deleted tuples. After this check, it reclaims storage by removing obsolete data or tuples from the PostgreSQL database.

By default, autovacuum is turned on for the RDS for PostgreSQL DB instances that you create using any of the default PostgreSQL DB parameter groups. Other configuration parameters associated with the autovacuum feature are also set by default. Because these defaults are somewhat generic, you can benefit from tuning some of the parameters associated with the autovacuum feature for your specific workload.

Following, you can find more information about the autovacuum and how to tune some of its parameters on your RDS for PostgreSQL DB instance. For high-level information, see Best practices for working with PostgreSQL.

Topics

- Allocating memory for autovacuum
- Reducing the likelihood of transaction ID wraparound
- Determining if the tables in your database need vacuuming
- Determining which tables are currently eligible for autovacuum
- Determining if autovacuum is currently running and for how long
- Performing a manual vacuum freeze
- Reindexing a table when autovacuum is running
- Managing autovacuum with large indexes
- Other parameters that affect autovacuum
- Setting table-level autovacuum parameters
- Logging autovacuum and vacuum activities
- Understanding the behavior of autovacuum with invalid databases
- Identify and resolve aggressive vacuum blockers in RDS for PostgreSQL

Allocating memory for autovacuum

One of the most important parameters influencing autovacuum performance is the `autovacuum_work_mem` parameter. In RDS for PostgreSQL versions 14 and prior, the `autovacuum_work_mem` parameter is set to -1, indicating that the setting of `maintenance_work_mem` is used instead. For all other versions, `autovacuum_work_mem` is determined by GREATEST({DBInstanceClassMemory/32768}, 65536).

Manual vacuum operations always use the `maintenance_work_mem` setting, with a default setting of GREATEST({DBInstanceClassMemory/63963136*1024}, 65536), and it can also be adjusted at the session level using the `SET` command for more targeted manual `VACUUM` operations.

The `autovacuum_work_mem` determines memory for autovacuum to hold identifiers of dead tuples (`pg_stat_all_tables.n_dead_tup`) for vacuuming indexes.

When doing calculations to determine the `autovacuum_work_mem` parameter's value, be aware of the following:

- If you set the parameter too low, the vacuum process might have to scan the table multiple times to complete its work. Such multiple scans can have a negative impact on performance. For larger instances, setting `maintenance_work_mem` or `autovacuum_work_mem` to at least 1 GB can improve the performance of vacuuming tables with a high number of dead tuples. However, in PostgreSQL versions 16 and prior, vacuum's memory usage is capped at 1 GB, which is sufficient to process approximately 179 million dead tuples in a single pass. If a table has more dead tuples than this, vacuum will need to make multiple passes through the table's indexes, significantly increasing the time required. Starting with PostgreSQL version 17, there isn't a limit of 1 GB, and autovacuum can process more than 179 million tuples by using radix trees.

  A tuple identifier is 6 bytes in size. To estimate the memory needed for vacuuming an index of a table, query `pg_stat_all_tables.n_dead_tup` to find the number of dead tuples, then multiply this number by 6 to determine the memory required for vacuuming the index in a single pass. You may use the following query:

  ```
  SELECT
      relname AS table_name,
      n_dead_tup,
      pg_size_pretty(n_dead_tup * 6) AS estimated_memory
  FROM
      pg_stat_all_tables
  WHERE
      relname = 'name_of_the_table';
  ```

- The `autovacuum_work_mem` parameter works in conjunction with the `autovacuum_max_workers` parameter. Each worker among `autovacuum_max_workers` can use the memory that you allocate. If you have many small tables, allocate more `autovacuum_max_workers` and less `autovacuum_work_mem`. If you have large tables (larger than 100 GB), allocate more memory and fewer worker processes. You need to have enough memory allocated to succeed on your biggest table. Thus, make sure that the combination of worker processes and memory equals the total memory that you want to allocate.

Reducing the likelihood of transaction ID wraparound

In some cases, parameter group settings related to autovacuum might not be aggressive enough to prevent transaction ID wraparound. To address this, RDS for PostgreSQL provides a mechanism that adapts the autovacuum parameter values automatically. *Adaptive autovacuum* is a feature

for RDS for PostgreSQL. A detailed explanation of TransactionID wraparound is found in the PostgreSQL documentation.

Adaptive autovacuum is turned on by default for RDS for PostgreSQL instances with the dynamic parameter `rds.adaptive_autovacuum` set to ON. We strongly recommend that you keep this turned on. However, to turn off adaptive autovacuum parameter tuning, set the `rds.adaptive_autovacuum` parameter to 0 or OFF.

Transaction ID wraparound is still possible even when Amazon RDS Amazon RDS tunes the autovacuum parameters. We encourage you to implement an Amazon CloudWatch alarm for transaction ID wraparound. For more information, see the post Implement an early warning system for transaction ID wraparound in RDS for PostgreSQL on the AWS Database Blog.

With adaptive autovacuum parameter tuning turned on, Amazon RDS begins adjusting autovacuum parameters when the CloudWatch metric `MaximumUsedTransactionIDs` reaches the value of the `autovacuum_freeze_max_age` parameter or 500,000,000, whichever is greater.

Amazon RDS continues to adjust parameters for autovacuum if a table continues to trend toward transaction ID wraparound. Each of these adjustments dedicates more resources to autovacuum to avoid wraparound. Amazon RDS updates the following autovacuum-related parameters:

- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit
- `autovacuum_work_mem`
- autovacuum_naptime

RDS modifies these parameters only if the new value makes autovacuum more aggressive. The parameters are modified in memory on the DB instance. The values in the parameter group aren't changed. To view the current in-memory settings, use the PostgreSQL SHOW SQL command.

When Amazon RDS modifies any of these autovacuum parameters, it generates an event for the affected DB instance. This event is visible on the AWS Management Console and through the Amazon RDS API. After the `MaximumUsedTransactionIDs` CloudWatch metric returns below the threshold, Amazon RDS resets the autovacuum-related parameters in memory back to the values specified in the parameter group. It then generates another event corresponding to this change.

Determining if the tables in your database need vacuuming

You can use the following query to show the number of unfrozen transactions in a database. The `datfrozenxid` column of a database's `pg_database` row is a lower bound on the normal transaction IDs appearing in that database. This column is the minimum of the per-table `relfrozenxid` values within the database.

```
SELECT datname, age(datfrozenxid) FROM pg_database ORDER BY age(datfrozenxid) desc
 limit 20;
```

For example, the results of running the preceding query might be the following.

```
datname    | age
mydb       | 1771757888
template0  | 1721757888
template1  | 1721757888
rdsadmin   | 1694008527
postgres   | 1693881061
(5 rows)
```

When the age of a database reaches 2 billion transaction IDs, transaction ID (XID) wraparound occurs and the database becomes read-only. You can use this query to produce a metric and run a few times a day. By default, autovacuum is set to keep the age of transactions to no more than 200,000,000 (`autovacuum_freeze_max_age`).

A sample monitoring strategy might look like this:

- Set the `autovacuum_freeze_max_age` value to 200 million transactions.
- If a table reaches 500 million unfrozen transactions, that triggers a low-severity alarm. This isn't an unreasonable value, but it can indicate that autovacuum isn't keeping up.
- If a table ages to 1 billion, this should be treated as an alarm to take action on. In general, you want to keep ages closer to `autovacuum_freeze_max_age` for performance reasons. We recommend that you investigate using the recommendations that follow.
- If a table reaches 1.5 billion unvacuumed transactions, that triggers a high-severity alarm. Depending on how quickly your database uses transaction IDs, this alarm can indicate that the system is running out of time to run autovacuum. In this case, we recommend that you resolve this immediately.

If a table is constantly breaching these thresholds, modify your autovacuum parameters further. By default, using VACUUM manually (which has cost-based delays disabled) is more aggressive than using the default autovacuum, but it is also more intrusive to the system as a whole.

We recommend the following:

- Be aware and turn on a monitoring mechanism so that you are aware of the age of your oldest transactions.

  For information on creating a process that warns you about transaction ID wraparound, see the AWS Database Blog post Implement an early warning system for transaction ID wraparound in Amazon RDS for PostgreSQL.
- For busier tables, perform a manual vacuum freeze regularly during a maintenance window, in addition to relying on autovacuum. For information on performing a manual vacuum freeze, see Performing a manual vacuum freeze.

Determining which tables are currently eligible for autovacuum

Often, it is one or two tables in need of vacuuming. Tables whose `relfrozenxid` value is greater than the number of transactions in `autovacuum_freeze_max_age` are always targeted by autovacuum. Otherwise, if the number of tuples made obsolete since the last VACUUM exceeds the vacuum threshold, the table is vacuumed.

The autovacuum threshold is defined as:

```
Vacuum-threshold = vacuum-base-threshold + vacuum-scale-factor * number-of-tuples
```

where the `vacuum base threshold` is `autovacuum_vacuum_threshold`, the `vacuum scale factor` is `autovacuum_vacuum_scale_factor`, and the `number of tuples` is `pg_class.reltuples`.

While you are connected to your database, run the following query to see a list of tables that autovacuum sees as eligible for vacuuming.

```
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold FROM
pg_settings WHERE name = 'autovacuum_vacuum_threshold'),
vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor FROM
pg_settings WHERE name = 'autovacuum_vacuum_scale_factor'),
```

```
fma AS (SELECT setting AS autovacuum_freeze_max_age FROM pg_settings WHERE name =
 'autovacuum_freeze_max_age'),
sto AS (select opt_oid, split_part(setting, '=', 1) as param,
split_part(setting, '=', 2) as value from (select oid opt_oid, unnest(reloptions)
 setting from pg_class) opt)
SELECT '"'||ns.nspname||'"."'||c.relname||'"' as relation,
pg_size_pretty(pg_table_size(c.oid)) as table_size,
age(relfrozenxid) as xid_age,
coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age,
(coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples)
AS autovacuum_vacuum_tuples, n_dead_tup as dead_tuples FROM
pg_class c join pg_namespace ns on ns.oid = c.relnamespace
join pg_stat_all_tables stat on stat.relid = c.oid join vbt on (1=1) join vsf on (1=1)
 join fma on (1=1)
left join sto cvbt on cvbt.param = 'autovacuum_vacuum_threshold' and c.oid =
 cvbt.opt_oid
left join sto cvsf on cvsf.param = 'autovacuum_vacuum_scale_factor' and c.oid =
 cvsf.opt_oid
left join sto cfma on cfma.param = 'autovacuum_freeze_max_age' and c.oid = cfma.opt_oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog'
AND (age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
OR coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
c.reltuples <= n_dead_tup)
ORDER BY age(relfrozenxid) DESC LIMIT 50;
```

Determining if autovacuum is currently running and for how long

If you need to manually vacuum a table, make sure to determine if autovacuum is currently running. If it is, you might need to adjust parameters to make it run more efficiently, or turn off autovacuum temporarily so that you can manually run VACUUM.

Use the following query to determine if autovacuum is running, how long it has been running, and if it is waiting on another session.

```
SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query
FROM pg_stat_activity
WHERE upper(query) LIKE '%VACUUM%'
ORDER BY xact_start;
```

After running the query, you should see output similar to the following.

```
 datname | usename  |  pid  | state  | wait_event |      xact_runtime       | query
 --------+----------+-------+--------+------------+-------------------------
+-------------------------------------------------------------------------------------------------
 mydb    | rdsadmin | 16473 | active |            | 33 days 16:32:11.600656 |
 autovacuum: VACUUM ANALYZE public.mytable1 (to prevent wraparound)
 mydb    | rdsadmin | 22553 | active |            | 14 days 09:15:34.073141 |
 autovacuum: VACUUM ANALYZE public.mytable2 (to prevent wraparound)
 mydb    | rdsadmin | 41909 | active |            | 3 days 02:43:54.203349  |
 autovacuum: VACUUM ANALYZE public.mytable3
 mydb    | rdsadmin |   618 | active |            | 00:00:00                |
 SELECT datname, usename, pid, state, wait_event, current_timestamp - xact_start AS
 xact_runtime, query+
         |          |       |        |            |                         | FROM
 pg_stat_activity
           +
         |          |       |        |            |                         | WHERE
 query like '%VACUUM%'
          +
         |          |       |        |            |                         | ORDER BY
 xact_start;
       +
```

Several issues can cause a long-running autovacuum session (that is, multiple days long). The most common issue is that your `maintenance_work_mem` parameter value is set too low for the size of the table or rate of updates.

We recommend that you use the following formula to set the `maintenance_work_mem` parameter value.

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

Short running autovacuum sessions can also indicate problems:

- It can indicate that there aren't enough `autovacuum_max_workers` for your workload. In this case, you need to indicate the number of workers.
- It can indicate that there is an index corruption (autovacuum crashes and restarts on the same relation but makes no progress). In this case, run a manual `vacuum freeze verbose` *`table`* to see the exact cause.

Performing a manual vacuum freeze

You might want to perform a manual vacuum on a table that has a vacuum process already running. This is useful if you have identified a table with an age approaching 2 billion transactions (or above any threshold you are monitoring).

The following steps are guidelines, with several variations to the process. For example, during testing, suppose that you find that the `maintenance_work_mem` parameter value is set too small and that you need to take immediate action on a table. However, perhaps you don't want to bounce the instance at the moment. Using the queries in previous sections, you determine which table is the problem and notice a long running autovacuum session. You know that you need to change the `maintenance_work_mem` parameter setting, but you also need to take immediate action and vacuum the table in question. The following procedure shows what to do in this situation.

To manually perform a vacuum freeze

1. Open two sessions to the database containing the table you want to vacuum. For the second session, use "screen" or another utility that maintains the session if your connection is dropped.
2. In session one, get the process ID (PID) of the autovacuum session running on the table.

   Run the following query to get the PID of the autovacuum session.

   ```
   SELECT datname, usename, pid, current_timestamp - xact_start
   AS xact_runtime, query
   FROM pg_stat_activity WHERE upper(query) LIKE '%VACUUM%' ORDER BY
   xact_start;
   ```

3. In session two, calculate the amount of memory that you need for this operation. In this example, we determine that we can afford to use up to 2 GB of memory for this operation, so we set `maintenance_work_mem` for the current session to 2 GB.

   ```
   SET maintenance_work_mem='2 GB';
   SET
   ```

4. In session two, issue a `vacuum freeze verbose` command for the table. The verbose setting is useful because, although there is no progress report for this in PostgreSQL currently, you can see activity.

```
\timing on
Timing is on.
vacuum freeze verbose pgbench_branches;
```

```
INFO:  vacuuming "public.pgbench_branches"
INFO:  index "pgbench_branches_pkey" now contains 50 row versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  index "pgbench_branches_test_index" now contains 50 row versions in 2 pages
DETAIL:  0 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
INFO:  "pgbench_branches": found 0 removable, 50 nonremovable row versions
     in 43 out of 43 pages
DETAIL:  0 dead row versions cannot be removed yet.
There were 9347 unused item pointers.
0 pages are entirely empty.
CPU 0.00s/0.00u sec elapsed 0.00 sec.
VACUUM
Time: 2.765 ms
```

5. In session one, if autovacuum was blocking the vacuum session, `pg_stat_activity` shows that waiting is `T` for your vacuum session. In this case, end the autovacuum process as follows.

```
SELECT pg_terminate_backend('the_pid');
```

> **Note**
>
> Some lower versions of Amazon RDS can't terminate an autovacuum process using the preceding command and fail with the following error: `ERROR: 42501: must be a superuser to terminate superuser process LOCATION: pg_terminate_backend, signalfuncs.c:227`.

At this point, your session begins. Autovacuum restarts immediately because this table is probably the highest on its list of work.

6. Initiate your `vacuum freeze verbose` command in session two, and then end the autovacuum process in session one.

Reindexing a table when autovacuum is running

If an index has become corrupt, autovacuum continues to process the table and fails. If you attempt a manual vacuum in this situation, you receive an error message like the following.

```
postgres=>  vacuum freeze pgbench_branches;
ERROR: index "pgbench_branches_test_index" contains unexpected
   zero page at block 30521
HINT: Please REINDEX it.
```

When the index is corrupted and autovacuum is attempting to run on the table, you contend with an already running autovacuum session. When you issue a REINDEX command, you take out an exclusive lock on the table. Write operations are blocked, and also read operations that use that specific index.

To reindex a table when autovacuum is running on the table

1. Open two sessions to the database containing the table that you want to vacuum. For the second session, use "screen" or another utility that maintains the session if your connection is dropped.
2. In session one, get the PID of the autovacuum session running on the table.

   Run the following query to get the PID of the autovacuum session.

   ```
   SELECT datname, usename, pid, current_timestamp - xact_start
   AS xact_runtime, query
   FROM pg_stat_activity WHERE upper(query) like '%VACUUM%' ORDER BY
   xact_start;
   ```

3. In session two, issue the reindex command.

   ```
   \timing on
   Timing is on.
   reindex index pgbench_branches_test_index;
   REINDEX
     Time: 9.966 ms
   ```

4. In session one, if autovacuum was blocking the process, you see in `pg_stat_activity` that waiting is "T" for your vacuum session. In this case, you end the autovacuum process.

   ```
   SELECT pg_terminate_backend('the_pid');
   ```

   At this point, your session begins. It's important to note that autovacuum restarts immediately because this table is probably the highest on its list of work.
5. Initiate your command in session two, and then end the autovacuum process in session 1.

Managing autovacuum with large indexes

As part of its operation, *autovacuum* performs several vacuum phases while running on a table. Before the table is cleaned up, all of its indexes are first vacuumed. When removing multiple large indexes, this phase consumes a significant amount of time and resources. Therefore, as a best practice, be sure to control the number of indexes on a table and eliminate unused indexes.

For this process, first check the overall index size. Then, determine if there are potentially unused indexes that can be removed as shown in the following examples.

To check the size of the table and its indexes

```
postgres=> select pg_size_pretty(pg_relation_size('pgbench_accounts'));
pg_size_pretty
6404 MB
(1 row)
```

```
postgres=> select pg_size_pretty(pg_indexes_size('pgbench_accounts'));
pg_size_pretty
11 GB
(1 row)
```

In this example, the size of indexes is larger than the table. This difference can cause performance issues as the indexes are bloated or unused, which impacts the autovacuum as well as insert operations.

To check for unused indexes

Using the `pg_stat_user_indexes` view, you can check how frequently an index is used with the `idx_scan` column. In the following example, the unused indexes have the `idx_scan` value of `0`.

```
postgres=> select * from pg_stat_user_indexes where relname = 'pgbench_accounts' order
 by idx_scan desc;

relid  | indexrelid | schemaname | relname          | indexrelname          | idx_scan
 | idx_tup_read | idx_tup_fetch
-------+------------+------------+------------------+-----------------------+----------
+--------------+--------------
16433  | 16454      | public     | pgbench_accounts | index_f               | 6
 | 6            | 0
16433  | 16450      | public     | pgbench_accounts | index_b               | 3
 | 199999       | 0
16433  | 16447      | public     | pgbench_accounts | pgbench_accounts_pkey | 0
 | 0            | 0
16433  | 16452      | public     | pgbench_accounts | index_d               | 0
 | 0            | 0
16433  | 16453      | public     | pgbench_accounts | index_e               | 0
 | 0            | 0
16433  | 16451      | public     | pgbench_accounts | index_c               | 0
 | 0            | 0
16433  | 16449      | public     | pgbench_accounts | index_a               | 0
 | 0            | 0
(7 rows)
```

```
postgres=> select schemaname, relname, indexrelname, idx_scan from pg_stat_user_indexes
 where relname = 'pgbench_accounts' order by idx_scan desc;

schemaname  | relname          | indexrelname          | idx_scan
------------+------------------+-----------------------+----------
public      | pgbench_accounts | index_f               | 6
public      | pgbench_accounts | index_b               | 3
public      | pgbench_accounts | pgbench_accounts_pkey | 0
public      | pgbench_accounts | index_d               | 0
public      | pgbench_accounts | index_e               | 0
public      | pgbench_accounts | index_c               | 0
public      | pgbench_accounts | index_a               | 0
(7 rows)
```

> **Note**
>
> These statistics are incremental from the time that the statistics are reset. Suppose you have an index that is only used at the end of a business quarter or just for a specific report. It's possible that this index hasn't been used since the statistics were reset. For more information, see Statistics Functions. Indexes that are used to enforce uniqueness won't have scans performed and shouldn't be identified as unused indexes. To identify the unused indexes, you should have in-depth knowledge of the application and its queries.

To check when the stats were last reset for a database, use `pg_stat_database`

```
postgres=> select datname, stats_reset from pg_stat_database where datname =
 'postgres';

datname   | stats_reset
----------+-------------------------------
postgres  | 2022-11-17 08:58:11.427224+00
(1 row)
```

Vacuuming a table as quickly as possible

RDS for PostgreSQL 12 and higher

If you have too many indexes in a large table, your DB instance could be nearing transaction ID wraparound (XID), which is when the XID counter wraps around to zero. Left unchecked, this situation could result in data loss. However, you can quickly vacuum the table without cleaning up the indexes. In RDS for PostgreSQL 12 and higher, you can use VACUUM with the `INDEX_CLEANUP` clause.

```
postgres=> VACUUM (INDEX_CLEANUP FALSE, VERBOSE TRUE) pgbench_accounts;

INFO: vacuuming "public.pgbench_accounts"
INFO: table "pgbench_accounts": found 0 removable, 8 nonremovable row versions in 1 out
 of 819673 pages
DETAIL: 0 dead row versions cannot be removed yet, oldest xmin: 7517
Skipped 0 pages due to buffer pins, 0 frozen pages.
CPU: user: 0.01 s, system: 0.00 s, elapsed: 0.01 s.
```

If an autovacuum session is already running, you must terminate it to begin the manual VACUUM. For information on performing a manual vacuum freeze, see Performing a manual vacuum freeze

> **Note**
>
> Skipping index cleanup regularly causes index bloat, which degrades scan performance. The index retains dead rows, and the table retains dead line pointers. As a result, `pg_stat_all_tables.n_dead_tup` increases until autovacuum or a manual VACUUM with index cleanup runs. As a best practice, use this procedure only to prevent transaction ID wraparound.

RDS for PostgreSQL 11 and older

However, in RDS for PostgreSQL 11 and lower versions, the only way to allow vacuum to complete faster is to reduce the number of indexes on a table. Dropping an index can affect query plans. We recommend that you drop unused indexes first, then drop the indexes when XID wraparound is very near. After the vacuum process completes, you can recreate these indexes.

Other parameters that affect autovacuum

The following query shows the values of some of the parameters that directly affect autovacuum and its behavior. The autovacuum parameters are described fully in the PostgreSQL documentation.

```
SELECT name, setting, unit, short_desc
FROM pg_settings
WHERE name IN (
'autovacuum_max_workers',
'autovacuum_analyze_scale_factor',
'autovacuum_naptime',
'autovacuum_analyze_threshold',
'autovacuum_analyze_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_scale_factor',
'autovacuum_vacuum_threshold',
'autovacuum_vacuum_cost_delay',
'autovacuum_vacuum_cost_limit',
'vacuum_cost_limit',
'autovacuum_freeze_max_age',
```

```
'maintenance_work_mem',
'vacuum_freeze_min_age');
```

While these all affect autovacuum, some of the most important ones are:

- maintenance_work_mem
- autovacuum_freeze_max_age
- autovacuum_max_workers
- autovacuum_vacuum_cost_delay
- autovacuum_vacuum_cost_limit

Setting table-level autovacuum parameters

You can set autovacuum-related storage parameters at a table level, which can be better than altering the behavior of the entire database. For large tables, you might need to set aggressive settings and you might not want to make autovacuum behave that way for all tables.

The following query shows which tables currently have table-level options in place.

```
SELECT relname, reloptions
FROM pg_class
WHERE reloptions IS NOT null;
```

An example where this might be useful is on tables that are much larger than the rest of your tables. Suppose that you have one 300-GB table and 30 other tables less than 1 GB. In this case, you might set some specific parameters for your large table so you don't alter the behavior of your entire system.

```
ALTER TABLE mytable set (autovacuum_vacuum_cost_delay=0);
```

Doing this turns off the cost-based autovacuum delay for this table at the expense of more resource usage on your system. Normally, autovacuum pauses for `autovacuum_vacuum_cost_delay` each time `autovacuum_cost_limit` is reached. For more details, see the PostgreSQL documentation about cost-based vacuuming.

Logging autovacuum and vacuum activities

Information about autovacuum activities is sent to the `postgresql.log` based on the level specified in the `rds.force_autovacuum_logging_level` parameter. Following are the values allowed for this parameter and the PostgreSQL versions for which that value is the default setting:

- `disabled` (PostgreSQL 10, PostgreSQL 9.6)
- `debug5`, `debug4`, `debug3`, `debug2`, `debug1`
- `info` (PostgreSQL 12, PostgreSQL 11)
- `notice`
- `warning` (PostgreSQL 13 and above)
- `error`, log, `fatal`, `panic`

The `rds.force_autovacuum_logging_level` works with the `log_autovacuum_min_duration` parameter. The `log_autovacuum_min_duration` parameter's value is the threshold (in milliseconds) above which autovacuum actions get logged. A setting of `-1` logs nothing, while a setting of 0 logs all actions. As with `rds.force_autovacuum_logging_level`, default values for `log_autovacuum_min_duration` are version dependent, as follows:

- `10000 ms` – PostgreSQL 14, PostgreSQL 13, PostgreSQL 12, and PostgreSQL 11
- `(empty)` – No default value for PostgreSQL 10 and PostgreSQL 9.6

We recommend that you set `rds.force_autovacuum_logging_level` to `WARNING`. We also recommend that you set `log_autovacuum_min_duration` to a value from 1000 to 5000. A setting of 5000 logs activity that takes longer than 5,000 milliseconds. Any setting other than -1 also logs messages if the autovacuum action is skipped because of a conflicting lock or concurrently dropped relations. For more information, see Automatic Vacuuming in the PostgreSQL documentation.

To troubleshoot issues, you can change the `rds.force_autovacuum_logging_level` parameter to one of the debug levels, from `debug1` up to `debug5` for the most verbose information. We recommend that you use debug settings for short periods of time and for troubleshooting purposes only. To learn more, see When to log in the PostgreSQL documentation.

Note

PostgreSQL allows the `rds_superuser` account to view autovacuum sessions in `pg_stat_activity`. For example, you can identify and end an autovacuum session that is blocking a command from running, or running slower than a manually issued vacuum command.

Understanding the behavior of autovacuum with invalid databases

A new value `-2` is introduced into the `datconnlimit` column in the `pg_database` catalog to indicate databases that have been interrupted in the middle of the DROP DATABASE operation as invalid.

This new value is available from the following RDS for PostgreSQL versions:

- 15.4 and all higher versions
- 14.9 and higher versions
- 13.12 and higher versions
- 12.16 and higher versions
- 11.21 and higher versions

Invalid databases do not affect autovacuum's ability to freeze functionality for valid databases. Autovacuum ignores invalid databases. Consequently, regular vacuum operations will continue to function properly and efficiently for all valid databases in your PostgreSQL environment.

Topics

- Monitoring transaction ID
- Adjusting the monitoring query
- Resolving invalid database issue

Monitoring transaction ID

The `age(datfrozenxid)` function is commonly used to monitor the transaction ID (XID) age of databases to prevent transaction ID wraparound.

Since invalid databases are excluded from autovacuum, their transaction ID (XID) counter can reach the maximum value of `2 billion`, wrap around to `- 2 billion`, and continue this cycle indefinitely. A typical query to monitor Transaction ID wraparound might look like:

```
SELECT max(age(datfrozenxid)) FROM pg_database;
```

However, with the introduction of the -2 value for `datconnlimit`, invalid databases can skew the results of this query. Since these databases are not valid and should not be part of regular maintenance checks, they can cause false positives, leading you to believe that the `age(datfrozenxid)` is higher than it actually is.

Adjusting the monitoring query

To ensure accurate monitoring, you should adjust your monitoring query to exclude invalid databases. Follow this recommended query:

```
SELECT
    max(age(datfrozenxid))
FROM
    pg_database
WHERE
    datconnlimit <> -2;
```

This query ensures that only valid databases are considered in the `age(datfrozenxid)` calculation, providing a true reflection of the transaction ID age across your PostgreSQL environment.

Resolving invalid database issue

When attempting to connect to an invalid database, you may encounter an error message similar to the following:

```
postgres=> \c db1
connection to server at "mydb.xxxxxxxxxx.us-west-2.rds.amazonaws.com" (xx.xx.xx.xxx),
 port xxxx failed: FATAL:  cannot connect to invalid database "db1"
HINT:  Use DROP DATABASE to drop invalid databases.
Previous connection kept
```

Additionally, if the `log_min_messages` parameter is set to `DEBUG2` or higher, you may notice the following log entries indicating that the autovacuum process is skipping the invalid database:

```
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db6"
2024-07-30 05:59:00 UTC::@:[32000]:DEBUG:  autovacuum: skipping invalid database "db1"
```

To resolve the issue, follow the `HINT` provided during the connection attempt. Connect to any valid database using your RDS master account or a database account with the `rds_superuser` role, and drop invalid database(s).

```
SELECT
    'DROP DATABASE ' || quote_ident(datname) || ';'
FROM
    pg_database
WHERE
    datconnlimit = -2 \gexec
```

Identify and resolve aggressive vacuum blockers in RDS for PostgreSQL

In PostgreSQL, vacuuming is vital for ensuring database health as it reclaims storage and prevents transaction ID wraparound issues. However, there are times when vacuuming can be prevented from operating as desired, which can result in performance degradation, storage bloat, and even impact availability of your DB instance by transaction ID wraparound. Therefore, identifying and resolving these issues are essential for optimal database performance and availability. Read Understanding autovacuum in Amazon RDS for PostgreSQL environments to learn more about autovacuum.

The `postgres_get_av_diag()` function helps identify issues that either prevent or delay the aggressive vacuum progress. Suggestions are provided, which may include commands to resolve the issue where it is identifiable or guidance for further diagnostics where the issue is not identifiable. Aggressive vacuum blockers are reported when the age exceeds RDS' adaptive autovacuum threshold of 500 million transaction IDs.

What is the age of the transaction ID?

The `age()` function for transaction IDs calculates the number of transactions that have occurred since the oldest unfrozen transaction ID for a database (`pg_database.datfrozenxid`) or table (`pg_class.relfrozenxid`). This value indicates database activity since the last aggressive vacuum operation and highlights the likely workload for upcoming VACUUM processes.

What is an aggressive vacuum?

An aggressive VACUUM operation conducts a comprehensive scan of all pages within a table, including those typically skipped during regular VACUUMs. This thorough scan aims to "freeze" transaction IDs approaching their maximum age, effectively preventing a situation known as transaction ID wraparound.

For `postgres_get_av_diag()` to report blockers, the blocker must be at least 500 million transactions old.

Topics

- Installing autovacuum monitoring and diagnostic tools in RDS for PostgreSQL
- Functions of postgres_get_av_diag() in RDS for PostgreSQL
- Resolving identifiable vacuum blockers in RDS for PostgreSQL
- Resolving unidentifiable vacuum blockers in RDS for PostgreSQL
- Resolving vacuum performance issues in RDS for PostgreSQL
- Explanation of the NOTICE messages in RDS for PostgreSQL

Installing autovacuum monitoring and diagnostic tools in RDS for PostgreSQL

The `postgres_get_av_diag()` function is currently available in the following RDS for PostgreSQL versions:

- 17.2 and higher 17 versions
- 16.7 and higher 16 versions
- 15.11 and higher 15 versions
- 14.16 and higher 14 versions
- 13.19 and higher 13 versions

In order to use `postgres_get_av_diag()`, create the `rds_tools` extension.

```
postgres=> CREATE EXTENSION rds_tools ;
CREATE EXTENSION
```

Verify that the extension is installed.

```
postgres=> \dx rds_tools
             List of installed extensions
   Name    | Version |  Schema   |                    Description
 ----------+---------+-----------
+-------------------------------------------------------------
 rds_tools |   1.8   | rds_tools | miscellaneous administrative functions for RDS
 PostgreSQL
 1 row
```

Verify that the function is created.

```
postgres=> SELECT
    proname function_name,
    pronamespace::regnamespace function_schema,
    proowner::regrole function_owner
FROM
    pg_proc
WHERE
    proname = 'postgres_get_av_diag';
    function_name     | function_schema | function_owner
----------------------+-----------------+----------------
 postgres_get_av_diag | rds_tools       | rds_superuser
(1 row)
```

Functions of postgres_get_av_diag() in RDS for PostgreSQL

The `postgres_get_av_diag()` function retrieves diagnostic information about autovacuum processes that are blocking or lagging behind in a RDS for PostgreSQL database. The query needs to be executed in the database with the oldest transaction ID for accurate results. For more information about using the database with the oldest transaction ID, see Not connected to the database with the age of oldest transaction ID

```
SELECT
```

```
    blocker,
    DATABASE,
    blocker_identifier,
    wait_event,
    TO_CHAR(autovacuum_lagging_by, 'FM9,999,999,999') AS autovacuum_lagging_by,
    suggestion,
    suggested_action
FROM (
    SELECT
        *
    FROM
        rds_tools.postgres_get_av_diag ()
    ORDER BY
        autovacuum_lagging_by DESC) q;
```

The `postgres_get_av_diag()` function returns a table with the following information:

blocker

Specifies the category of database activity that is blocking the vacuum.

- Active statement
- Idle in transaction
- Prepared transaction
- Logical replication slot
- Read replica with physical replication slot
- Read replica with streaming replication
- Temporary tables

database

Specifies the name of the database where applicable and supported. This is the database in which the activity is ongoing and blocking or will block the autovacuum. This is the database you are required to connect to and take action.

blocker_identifier

Specifies the identifier of the activity that is blocking or will block the autovacuum. The identifier can be a process ID along with a SQL statement, a prepared transaction, an IP address of a read replica, and the name of the replication slot, either logical or physical.

wait_event

Specifies the wait event of the blocking session and is applicable for the following blockers:

- Active statement
- Idle in transaction

autovacum_lagging_by

Specifies the number of transactions that autovacuum is lagging behind in its backlog work per category.

suggestion

Specifies suggestions to resolve the blocker. These instructions include the name of the database in which the activity exists where applicable, the Process ID (PID) of the session where applicable, and the action to be taken.

suggested_action

Suggests the action that needs to be taken to resolve the blocker.

Resolving identifiable vacuum blockers in RDS for PostgreSQL

Autovacuum performs aggressive vacuums and lowers the age of transaction IDs to below the threshold specified by the `autovacuum_freeze_max_age` parameter of your RDS instance. You can track this age using the Amazon CloudWatch metric `MaximumUsedTransactionIDs`.

To find the setting of `autovacuum_freeze_max_age` (which has a default of 200 million transaction IDs) for your Amazon RDS instance, you can use the following query:

```
SELECT
    TO_CHAR(setting::bigint, 'FM9,999,999,999') autovacuum_freeze_max_age
FROM
    pg_settings
WHERE
    name = 'autovacuum_freeze_max_age';
```

Note that `postgres_get_av_diag()` only checks for aggressive vacuum blockers when the age exceeds Amazon RDS' adaptive autovacuum threshold of 500 million transaction IDs. For `postgres_get_av_diag()` to detect blockers, the blocker must be at least 500 million transactions old.

The `postgres_get_av_diag()` function identifies the following types of blockers:

Topics

- Active statement
- Idle in transaction
- Prepared transaction
- Logical replication slot
- Read replicas
- Temporary tables

Active statement

In PostgreSQL, an active statement is an SQL statement that is currently being executed by the database. This includes queries, transactions, or any operations in progress. When monitoring via `pg_stat_activity`, the state column indicates that the process with the corresponding PID is active.

The `postgres_get_av_diag()` function displays output similar to the following when it identifies a statement that is an active statement.

```
blocker               | Active statement
database              | my_database
blocker_identifier    | SELECT pg_sleep(20000);
wait_event            | Timeout:PgSleep
autovacuum_lagging_by | 568,600,871
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (29621);"}
```

Suggested action

Following the guidance in the `suggestion` column, the user can connect to the database where the active statement is present and, as specified in the `suggested_action` column, it's advisable to carefully review the option to terminate the session. If termination is safe, you may use the `pg_terminate_backend()` function to terminate the session. This action can be performed by an administrator (such as the RDS master account) or a user with the required `pg_terminate_backend()` privilege.

> **Warning**
>
> A terminated session will undo (ROLLBACK) changes it made. Depending on your requirements, you may want to rerun the statement. However, it is recommended to do so only after the autovacuum process has finished its aggressive vacuum operation.

Idle in transaction

An idle in transaction statement refers to any session that has opened an explicit transaction (such as by issuing a BEGIN statement), performed some work, and is now waiting for the client to either pass more work or signal the end of the transaction by issuing a COMMIT, ROLLBACK, or END (which would result in an implicit COMMIT).

The `postgres_get_av_diag()` function displays output similar to the following when it identifies an `idle in transaction` statement as a blocker.

```
blocker               | idle in transaction
database              | my_database
blocker_identifier    | INSERT INTO tt SELECT * FROM tt;
wait_event            | Client:ClientRead
autovacuum_lagging_by | 1,237,201,759
suggestion            | Connect to database "my_database", review carefully and you
 may consider terminating the process using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"SELECT pg_terminate_backend (28438);"}
```

Suggested action

As indicated in the `suggestion` column, you can connect to the database where the idle in transaction session is present and terminate the session using the `pg_terminate_backend()` function. The user can be your admin (RDS master account) user or a user with the `pg_terminate_backend()` privilege.

> **Warning**
>
> A terminated session will undo (ROLLBACK) changes it made. Depending on your requirements, you may want to rerun the statement. However, it is recommended to do so only after the autovacuum process has finished its aggressive vacuum operation.

Prepared transaction

PostgreSQL allows transactions that are part of a two-phase commit strategy called prepared transactions. These are enabled by setting the `max_prepared_transactions` parameter to a non-zero value. Prepared transactions are designed to ensure that a transaction is durable and remains available even after database crashes, restarts, or client disconnections. Like regular transactions, they are assigned a transaction ID and can affect the autovacuum. If left in a prepared state, autovacuum cannot perform freeezing and it can lead to transaction ID wraparound.

When transactions are left prepared indefinitely without being resolved by a transaction manager, they become orphaned prepared transactions. The only way to fix this is to either commit or rollback the transaction using the `COMMIT PREPARED` or `ROLLBACK PREPARED` commands, respectively.

> **Note**
>
> Be aware that a backup taken during a prepared transaction will still contain that transaction after restoration. Refer to the following information about how to locate and close such transactions.

The `postgres_get_av_diag()` function displays the following output when it identifies a blocker that is a prepared transaction.

```
blocker               | Prepared transaction
database              | my_database
blocker_identifier    | myptx
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database" and consider either COMMIT
 or ROLLBACK the prepared transaction using suggested_action. For more information, see
 Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"COMMIT PREPARED 'myptx';",[OR],"ROLLBACK PREPARED 'myptx';"}
```

Suggested action

As mentioned in the suggestion column, connect to the database where the prepared transaction is located. Based on the `suggested_action` column, carefully review whether to perform either `COMMIT` or `ROLLBACK`, and the the appropiate the action.

To monitor prepared transactions in general, PostgreSQL offers a catalog view called `pg_prepared_xacts`. You can use the following query to find prepared transactions.

```
SELECT
    gid,
    prepared,
    owner,
    database,
    transaction AS oldest_xmin
FROM
    pg_prepared_xacts
ORDER BY
    age(transaction) DESC;
```

Logical replication slot

The purpose of a replication slot is to hold unconsumed changes until they are replicated to a target server. For more information, see PostgreSQL's Logical replication.

There are two types of logical replication slots.

Inactive logical replication slots

When replication is terminated, unconsumed transaction logs can't be removed, and the replication slot becomes inactive. Although an inactive logical replication slot isn't currently used by a subscriber, it remains on the server, leading to the retention of WAL files and preventing the removal of old transaction logs. This can increase disk usage and specifically block autovacuum from cleaning up internal catalog tables, as the system must preserve LSN information from being overwritten. If not addressed, this can result in catalog bloat, performance degradation, and an increased risk of wraparound vacuum, potentially causing transaction downtime.

Active but slow logical replication slots

Sometimes removal of dead tuples of catalog is delayed due to the performance degradation of logical replication. This delay in replication slows down updating the `catalog_xmin` and can lead to catalog bloat and wraparound vacuum.

The `postgres_get_av_diag()` function displays output similar to the following when it finds a logical replication slot as a blocker.

```
blocker              | Logical replication slot
```

```
database              | my_database
blocker_identifier    | slot1
wait_event            | Not applicable
autovacuum_lagging_by | 1,940,103,068
suggestion            | Ensure replication is active and resolve any lag for the slot
 if active. If inactive, consider dropping it using the command in suggested_action.
 For more information, see Working with PostgreSQL autovacuum in the Amazon RDS User
 Guide.
suggested_action      | {"SELECT pg_drop_replication_slot('slot1') FROM
 pg_replication_slots WHERE active = 'f';"}
```

Suggested action

To resolve this problem, check the replication configuration for issues with the target schema or data that might be terminating the apply process. The most common reasons are the following:

- Missing columns
- Incompatible data type
- Data mismatch
- Missing table

If the problem is related to infrastructure issues:

- Network issues - How do I resolve issues with an Amazon RDS DB in an incompatible network state?.
- Database or DB instance is not available due to the following reasons:
  - Replica instance is out of storage - Review Amazon RDS DB instances run out of storage for information about adding storage.
  - Incompatible-parameters - Review How can I fix an Amazon RDS DB instance that is stuck in the incompatible-parameters status? for more information about how you can resolve the issue.

If your instance is outside the AWS network or on AWS EC2, consult your administrator on how to resolve the availability or infrastructure-related issues.

Dropping the inactive slot

> **Warning**
>
> Caution: Before dropping a replication slot, carefully ensure that it has no ongoing replication, is inactive, and is in an unrecoverable state. Dropping a slot prematurely could disrupt replication or cause data loss.

After confirming that the replication slot is no longer needed, drop it to allow autovacuum to continue. The condition `active = 'f'` ensures that only an inactive slot is dropped.

```
SELECT pg_drop_replication_slot('slot1') WHERE active ='f'
```

Read replicas

When the `hot_standby_feedback` setting is enabled for Amazon RDS read replicas, it prevents autovacuum on the primary database from removing dead rows that might still be needed by queries running on the read replica. This affects all types of physical read replicas including those managed with or without replication slots. This behavior is necessary because queries running on the standby replica require those rows to remain available on the primary preventing query conflicts and cancellations.

Read replica with physical replication slot

Read replicas with physical replication slots significantly enhance the reliability and stability of replication in RDS for PostgreSQL. These slots ensure the primary database retains essential Write-Ahead Log files until the replica processes them, maintaining data consistency even during network disruptions.

Beginning with RDS for PostgreSQL version 14, all replicas utilize replication slots. In earlier versions, only cross-Region replicas used replication slots.

The `postgres_get_av_diag()` function displays output similar to the following when it finds a read replica with physical replication slot as the blocker.

```
blocker               | Read replica with physical replication slot
database              |
blocker_identifier    | rds_us_west_2_db_xxxxxxxxxxxxxxxxxxxx
wait_event            | Not applicable
autovacuum_lagging_by | 554,080,689
```

```
suggestion            | Run the following query on the replica
 "rds_us_west_2_db_xxxxxxxxxxxxxxxxxxx" to find the long running query:

                      | SELECT * FROM pg_catalog.pg_stat_activity WHERE
 backend_xmin::text::bigint = 757989377;

                      | Review carefully and you may consdier terminating the query on
 read replica using suggested_action. For more information, see Working with PostgreSQL
 autovacuum in the Amazon RDS User Guide.                                    +
          |
suggested_action      | {"SELECT pg_terminate_backend(pid) FROM
 pg_catalog.pg_stat_activity WHERE backend_xmin::text::bigint = 757989377;","
                                                                             +
                      | [OR]

                                                +
                      | ","Disable hot_standby_feedback","

                                                +
                      | [OR]

                                                +
                      | ","Delete the read replica if not needed"}
```

Read replica with streaming replication

Amazon RDS allows setting up read replicas without a physical replication slot in older versions, up to version 13. This approach reduces overhead by allowing the primary to recycle WAL files more aggressively, which is advantageous in environments with limited disk space and can tolerate occasional ReplicaLag. However, without a slot, the standby must remain in sync to avoid missing WAL files. Amazon RDS uses archived WAL files to help the replica catch up if it falls behind, but this process requires careful monitoring and can be slow.

The `postgres_get_av_diag()` function displays output similar to the following when it finds a streaming read replica as the blocker.

```
blocker               | Read replica with streaming replication slot
database              | Not applicable
blocker_identifier    | xx.x.x.xxx/xx
wait_event            | Not applicable
autovacuum_lagging_by | 610,146,760
```

```
suggestion            | Run the following query on the replica "xx.x.x.xxx" to find the
 long running query:
                                                                                +
                      | SELECT * FROM pg_catalog.pg_stat_activity WHERE
 backend_xmin::text::bigint = 348319343;

               +
                      | Review carefully and you may consdier terminating the query on
 read replica using suggested_action. For more information, see Working with PostgreSQL
 autovacuum in the Amazon RDS User Guide.                                        +
                      |
suggested_action      | {"SELECT pg_terminate_backend(pid) FROM
 pg_catalog.pg_stat_activity WHERE backend_xmin::text::bigint = 348319343;","

                       +
                      | [OR]

                                                                                +
                      | ","Disable hot_standby_feedback","

                                                                                +
                      | [OR]

                                                                                +
                      | ","Delete the read replica if not needed"}
```

Suggested action

As recommended in the `suggested_action` column, carefully review these options to unblock autovacuum.

- **Terminate the query** – Following the guidance in the suggestion column, you can connect to the read replica, as specified in the suggested_action column, it's advisable to carefully review the option to terminate the session. If termination is deemed safe, you may use the `pg_terminate_backend()` function to terminate the session. This action can be performed by an administrator (such as the RDS master account) or a user with the required pg_terminate_backend() privilege.

  You may run the following SQL command on the read replica to terminate the query that is preventing the vacuum on the primary from cleaning up old rows. The value of `backend_xmin` is reported in the function's output:

```
SELECT
    pg_terminate_backend(pid)
FROM
    pg_catalog.pg_stat_activity
WHERE
    backend_xmin::text::bigint = backend_xmin;
```

- **Disable hot standby feedback** – Consider disabling the `hot_standby_feedback` parameter if it's causing significant vacuum delays.

  The `hot_standby_feedback` parameter allows a read replica to inform the primary about its query activity, preventing the primary from vacuuming tables or rows that are in use on the standby. While this ensures query stability on the standby, it can significantly delay vacuuming on the primary. Disabling this feature allows the primary to proceed with vacuuming without waiting for the standby to catch up. However, this can lead to query cancellations or failures on the standby if it attempts to access rows that have been vacuumed by the primary.
- **Delete the read replica if not needed** – If the read replica is no longer necessary, you can delete it. This will remove the associated replication overhead and allow the primary to recycle transaction logs without being held back by the replica.

Temporary tables

Temporary tables, created using the `TEMPORARY` keyword, reside in the temp schema, for example pg_temp_xxx, and are only accessible to the session that created them. Temporary tables are dropped when the session ends. However, these tables are invisible to PostgreSQL's autovacuum process, and must be manually vacuumed by the session that created them. Trying to vacuum the temp table from another session has no effect.

In unusual circumstances, a temporary table exists without an active session owning it. If the owning session ends unexpectedly due to a fatal crash, network issue, or similar event, the temporary table might not be cleaned up, leaving it behind as an "orphaned" table. When the PostgreSQL autovacuum process detects an orphaned temporary table, it logs the following message:

```
LOG: autovacuum: found orphan temp table \"%s\".\"%s\" in database \"%s\"
```

The `postgres_get_av_diag()` function displays output similar to the following when it identifies a temporary table as a blocker. For the function to correctly show the output related to temporary tables, it needs to be executed within the same database where those tables exist.

```
blocker               | Temporary table
database              | my_database
blocker_identifier    | pg_temp_14.ttemp
wait_event            | Not applicable
autovacuum_lagging_by | 1,805,802,632
suggestion            | Connect to database "my_database". Review carefully, you
 may consider dropping temporary table using command in suggested_action. For more
 information, see Working with PostgreSQL autovacuum in the Amazon RDS User Guide.
suggested_action      | {"DROP TABLE ttemp;"}
```

Suggested action

Follow the instructions provided in the `suggestion` column of the output to identify and remove the temporary table that is preventing autovacuum from running. Use the following command to drop the temporary table reported by `postgres_get_av_diag()`. Replace the table name based on the output provided by the `postgres_get_av_diag()` function.

```
DROP TABLE my_temp_schema.my_temp_table;
```

The following query can be used to identify temporary tables:

```
SELECT
    oid,
    relname,
    relnamespace::regnamespace,
    age(relfrozenxid)
FROM
    pg_class
WHERE
relpersistence = 't'
ORDER BY
    age(relfrozenxid) DESC;
```

Resolving unidentifiable vacuum blockers in RDS for PostgreSQL

This section explores additional reasons that can prevent vacuuming from making progress. These issues are currently not directly identifiable by the `postgres_get_av_diag()` function.

Topics

- Invalid pages
- Index inconsistency
- Exceptionally high transaction rate

Invalid pages

An invalid page error occurs when PostgreSQL detects a mismatch in a page's checksum while accessing that page. The contents are unreadable, preventing autovacuum from freezing tuples. This effectively stops the cleanup process. The following error is written into PostgreSQL's log:

```
WARNING:  page verification failed, calculated checksum YYYYY but expected XXXX
ERROR:  invalid page in block ZZZZZ of relation base/XXXXX/XXXXX
CONTEXT:  automatic vacuum of table myschema.mytable
```

Determine the object type

```
ERROR: invalid page in block 4305910 of relation base/16403/186752608
WARNING: page verification failed, calculated checksum 50065 but expected 60033
```

From the error message, the path `base/16403/186752608` provides the following information:

- "base" is the directory name under the PostgreSQL data directory.
- "16403" is the database OID, which you can look up in the `pg_database` system catalog.
- "186752608" is the `relfilenode`, which you can use to look up the schema and object name in the `pg_class` system catalog.

By checking the output of the following query in the impacted database, you can determine the object type. The following query retrieves object information for oid: 186752608. Replace the OID with the one relevant to the error you encountered.

```
SELECT
    relname AS object_name,
    relkind AS object_type,
    nspname AS schema_name
FROM
    pg_class c
    JOIN pg_namespace n ON c.relnamespace = n.oid
```

```
WHERE
    c.oid = 186752608;
```

For more information, see the PostgreSQL documentation pg_class for all the supported object types, noted by the relkind column in pg_class.

Guidance

The most effective solution for this issue depends on the configuration of your specific Amazon RDS instance and the type of data impacted by the inconsistent page.

If the object type is an index:

Rebuilding the index is recommended.

- **Using the CONCURRENTLY option** – Prior to PostgreSQL version 12, rebuilding an index required an exclusive table lock, restricting access to the table. With PostgreSQL version 12, and later versions, the CONCURRENTLY option allows for row-level locking, significantly improving the table's availability. Following is the command:

  ```
  REINDEX INDEX ix_name CONCURRENTLY;
  ```

  While CONCURRENTLY is less disruptive, it can be slower on busy tables. Consider building the index during low-traffic periods if possible.

  For more information, see the PostgreSQL REINDEX documentation.
- **Using the INDEX_CLEANUP FALSE option** – If the indexes are large and estimated to require a significant amount of time to finish, you can unblock autovacuum by executing a manual VACUUM FREEZE while excluding indexes. This functionality is available in PostgreSQL version 12 and later versions.

  Bypassing indexes will allow you to skip the vacuum process of the inconsistent index and mitigate the wraparound issue. However, this will not resolve the underlying invalid page problem. To fully address and resolve the invalid page issue, you will still need to rebuild the index.

If the object type is a materialized view:

If an invalid page error occurs on a materialized view, login to the impacted database and refresh it to resolve the invalid page:

Refresh the materialized view:

```
REFRESH MATERIALIZED VIEW schema_name.materialized_view_name;
```

If refreshing fails, try recreating:

```
DROP MATERIALIZED VIEW schema_name.materialized_view_name;
CREATE MATERIALIZED VIEW schema_name.materialized_view_name AS query;
```

Refreshing or recreating the materialized view restores it without impacting the underlying table data.

For all other object types:

For all other object types, reach out to AWS support.

Index inconsistency

A logically inconsistent index can prevent autovacuum from making progress. The following errors or similar errors are logged during either the vacuum phase of the index or when the index is accessed by SQL statements.

```
ERROR: right sibling's left-link doesn't match:block 5 links to 10 instead of expected
 2 in index ix_name
```

```
ERROR: failed to re-find parent key in index "XXXXXXXXXX" for deletion target page XXX
CONTEXT:  while vacuuming index index_name of relation schema.table
```

Guidance

Rebuild the index or skip indexes using `INDEX_CLEANUP` on manual `VACUUM FREEZE`. For information about how to rebuild the index, see If the object type is an index.

- **Using the CONCURRENTLY option** – Prior to PostgreSQL version 12, rebuilding an index required an exclusive table lock, restricting access to the table. With PostgreSQL version 12, and later versions, the CONCURRENTLY option allows for row-level locking, significantly improving the table's availability. Following is the command:

  ```
  REINDEX INDEX ix_name CONCURRENTLY;
  ```

While CONCURRENTLY is less disruptive, it can be slower on busy tables. Consider building the index during low-traffic periods if possible. For more information, see REINDEX in *PostgreSQL* documentation.

- **Using the INDEX_CLEANUP FALSE option** – If the indexes are large and estimated to require a significant amount of time to finish, you can unblock autovacuum by executing a manual VACUUM FREEZE while excluding indexes. This functionality is available in PostgreSQL version 12 and later versions.

  Bypassing indexes will allow you to skip the vacuum process of the inconsistent index and mitigate the wraparound issue. However, this will not resolve the underlying invalid page problem. To fully address and resolve the invalid page issue, you will still need to rebuild the index.

Exceptionally high transaction rate

In PostgreSQL, high transaction rates can significantly impact autovacuum's performance, leading to slower cleanup of dead tuples and increased risk of transaction ID wraparound. You can monitor the transaction rate by measuring the difference in `max(age(datfrozenxid))` between two time periods, typically per second. Additionally, you can use the following counter metrics from RDS Performance Insights to measure the transaction rate (the sum of xact_commit and xact_rollback) which is the total number of transactions.

Counter	Type	Unit	Metric
xact_commit	Transactions	Commits per second	db.Transactions.xact_commit
xact_rollback	Transactions	Rollbacks per second	db.Transactions.xact_rollback

A rapid increase indicates a high transaction load, which can overwhelm autovacuum, causing bloat, lock contention, and potential performance issues. This can negatively impact the autovacuum process in a couple of ways:

- **Table Activity:** The specific table being vacuumed could be experiencing a high volume of transactions, causing delays.

- **System Resources** The overall system might be overloaded, making it difficult for autovacuum to access the necessary resources to function efficiently.

Consider the following strategies for allowing autovacuum to operate more effectively and keep up with its tasks:

1. Reduce the transaction rate if possible. Consider to batch or group similar transactions where feasible.
2. Target frequently updated tables with manual `VACUUM FREEZE` operation nightly, weekly, or biweekly during off-peak hours.
3. Consider scaling up your instance class to allocate more system resources to handle the high transaction volume and autovacuum.

Resolving vacuum performance issues in RDS for PostgreSQL

This section discusses factors that often contribute to slower vacuum performance and how to address these issues.

Topics

- Vacuum large indexes
- Too many tables or databases to vacuum
- Aggressive vacuum (to prevent wraparound) is running

Vacuum large indexes

VACUUM operates through sequential phases: initialization, heap scanning, index and heap vacuuming, index cleanup, heap truncation, and final cleanup. During the heap scan, the process prunes pages, defragments and freezes them. After completing the heap scan, VACUUM cleans indexes, returns empty pages to the operating system, and performs final cleanup tasks like vacuuming the free space map and updating statistics.

Index vacuuming may require multiple passes when `maintenance_work_mem` (or `autovacuum_work_mem`) is insufficient to process the index. In PostgreSQL 16 and earlier, a 1 GB memory limit for storing dead tuple IDs often forced multiple passes on large indexes. PostgreSQL 17 introduces `TidStore`, which dynamically allocates memory instead of using a single-allocation

array. This removes the 1 GB constraint, uses memory more efficiently, and reduces the need for multiple index scans per each index.

Large indexes may still require multiple passes in PostgreSQL 17 if available memory can't accommodate the entire index processing at once. Typically, larger indexes contain more dead tuples that require multiple passes.

Detecting slow vacuum operations

The `postgres_get_av_diag()` function can detect when vacuum operations are running slowly due to insufficient memory. For more information on this function, see Installing autovacuum monitoring and diagnostic tools in RDS for PostgreSQL.

The `postgres_get_av_diag()` function issues the following notices when the available memory is not enough to complete the index vacuuming in a single pass.

`rds_tools` 1.8

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_work_mem is "XXX" and might not be
 sufficient. Consider increasing the setting, and if necessary, scaling up the Amazon
 RDS instance class for more memory.
       Additionally, review the possibility of manual vacuum with exclusion of indexes
 using (VACUUM (INDEX_CLEANUP FALSE, VERBOSE TRUE) table_name;).
```

`rds_tools` 1.9

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_work_mem is XX might not be sufficient.
 Consider increasing the setting to XXX, and if necessary, scaling up the RDS instance
 class for more
       memory. The suggested value is an estimate based on the current number of dead
 tuples for the table being vacuumed, which might not fully reflect the latest state.
 Additionally, review the possibility of manual
       vacuum with exclusion of indexes using (VACUUM (INDEX_CLEANUP FALSE, VERBOSE
 TRUE) table_name;). For more information, see
```

```
        Working with PostgreSQL autovacuum in the Amazon Amazon RDS User Guide
        .
```

> **Note**
>
> The `postgres_get_av_diag()` function relies on `pg_stat_all_tables.n_dead_tup` for estimating the amount of memory required for index vacuuming.

When the `postgres_get_av_diag()` function identifies a slow vacuum operation that requires multiple index scans due to insufficient `autovacuum_work_mem`, it will generate the following message:

```
NOTICE: Your vacuum is performing multiple index scans due to insufficient
 autovacuum_work_mem:XXX for index vacuuming.
        For more information, see Working with PostgreSQL autovacuum in the Amazon
 Amazon RDS User Guide.
```

Guidance

You can apply the following workarounds using manual `VACUUM FREEZE` to speed up freezing the table.

Increase the memory for vacuuming

As suggested by the `postgres_get_av_diag()` function, it's advisable to increase the `autovacuum_work_mem` parameter to address potential memory constraints at the instance level. While `autovacuum_work_mem` is a dynamic parameter, it's important to note that for the new memory setting to take effect, the autovacuum daemon needs to restart its workers. To accomplish this:

1. Confirm that the new setting is in place.
2. Terminate the processes currently running autovacuum.

This approach ensures that the adjusted memory allocation is applied to new autovacuum operations.

For more immediate results, consider manually performing a `VACUUM FREEZE` operation with an increased `maintenance_work_mem` setting within your session:

```
SET maintenance_work_mem TO '1GB';
VACUUM FREEZE VERBOSE table_name;
```

If you're using Amazon RDS and find that you need additional memory to support higher values for `maintenance_work_mem` or `autovacuum_work_mem`, consider upgrading to an instance class with more memory. This can provide the necessary resources to enhance both manual and automatic vacuum operations, leading to improved overall vacuum and database performance.

Disable INDEX_CLEANUP

Manual `VACUUM` in PostgreSQL version 12 and later allows skipping the index cleanup phase, while emergency autovacuum in PostgreSQL version 14 and later does this automatically based on the `vacuum_failsafe_age` parameter.

> **Warning**
>
> Skipping index cleanup can lead to index bloat and negatively impact query performance. To mitigate this, consider reindexing or vacuuming affected indexes during a maintenance window.

For additional guidance on handling large indexes, refer to the documentation on Managing autovacuum with large indexes.

Parallel index vacuuming

Starting with PostgreSQL 13, indexes can be vacuumed and cleaned in parallel by default using manual `VACUUM`, with one vacuum worker process assigned to each index. However, for PostgreSQL to determine if a vacuum operation qualifies for parallel execution, specific criteria must be met:

- There must be at least two indexes.
- The `max_parallel_maintenance_workers` parameter should be set to at least 2.
- The index size must exceed the `min_parallel_index_scan_size` limit, which defaults to 512KB.

You can adjust the `max_parallel_maintenance_workers` setting based on the number of vCPUs available on your Amazon RDS instance and the number of indexes on the table to optimize vacuuming turnaround time.

For more information, see Parallel vacuuming in Amazon RDS for PostgreSQL and Amazon Aurora PostgreSQL.

Too many tables or databases to vacuum

As mentioned in PostgreSQL's The Autovacuum Daemon documentation, the autovacuum daemon operates through multiple processes. This includes a persistent autovacuum launcher responsible for starting autovacuum worker processes for each database within the system. The launcher schedules these workers to initiate approximately every `autovacuum_naptime` seconds per database.

With 'N' databases, a new worker begins roughly every [`autovacuum_naptime`/N seconds]. However, the total number of concurrent workers is limited by the `autovacuum_max_workers` setting. If the number of databases or tables requiring vacuuming exceeds this limit, the next database or table will be processed as soon as a worker becomes available.

When many large tables or databases require vacuuming concurrently, all available autovacuum workers can become occupied for an extended duration, delaying maintenance on other tables and databases. In environments with high transaction rates, this bottleneck can quickly escalate and potentially lead to wraparound vacuum issues within your Amazon RDS instance.

When `postgres_get_av_diag()` detects a high number of tables or databases, it provides the following recommendation:

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound and
 it might be slow.
```

```
NOTICE: The current setting of autovacuum_max_workers:3 might not be sufficient.
 Consider increasing the setting and, if necessary, consider scaling up the Amazon RDS
 instance class for more workers.
```

Guidance

Increase autovacuum_max_workers

To expedite the vacuuming, we recommend adjusting the `autovacuum_max_workers` parameter to allow more concurrent autovacuum workers. If performance bottlenecks persist, consider scaling up your Amazon RDS instance to a class with more vCPUs, which can further improve the parallel processing capabilities.

Aggressive vacuum (to prevent wraparound) is running

The age of the database (MaximumUsedTransactionIDs) in PostgreSQL only decreases when an aggressive vacuum (to prevent wraparound) is successfully completed. Until this vacuum finishes, the age will continue to increase depending on the transaction rate.

The `postgres_get_av_diag()` function generates the following `NOTICE` when it detects an aggressive vacuum. However, it only triggers this output after the vacuum has been active for at least two minutes.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

For more information about aggressive vacuum, see [When an aggressive vacuum is already running].

You can verify if an aggressive vacuum is in progress with the following query:

```
SELECT
    a.xact_start AS start_time,
    v.datname "database",
    a.query,
    a.wait_event,
    v.pid,
    v.phase,
    v.relid::regclass,
    pg_size_pretty(pg_relation_size(v.relid)) AS heap_size,
    (
        SELECT
            string_agg(pg_size_pretty(pg_relation_size(i.indexrelid)) || ':' ||
 i.indexrelid::regclass || chr(10), ', ')
        FROM
            pg_index i
        WHERE
            i.indrelid = v.relid
    ) AS index_sizes,
    trunc(v.heap_blks_scanned * 100 / NULLIF(v.heap_blks_total, 0)) AS step1_scan_pct,
    v.index_vacuum_count || '/' || (
        SELECT
            count(*)
        FROM
            pg_index i
```

```
        WHERE
            i.indrelid = v.relid
    ) AS step2_vacuum_indexes,
    trunc(v.heap_blks_vacuumed * 100 / NULLIF(v.heap_blks_total, 0)) AS
 step3_vacuum_pct,
    age(CURRENT_TIMESTAMP, a.xact_start) AS total_time_spent_sofar
FROM
    pg_stat_activity a
    INNER JOIN pg_stat_progress_vacuum v ON v.pid = a.pid;
```

You can determine if it's an aggressive vacuum (to prevent wraparound) by checking the query column in the output. The phrase "to prevent wraparound" indicates that it is an aggressive vacuum.

```
query                 | autovacuum: VACUUM public.t3 (to prevent wraparound)
```

For example, suppose you have a blocker at transaction age 1 billion and a table requiring an aggressive vacuum to prevent wraparound at the same transaction age. Additionally, there's another blocker at transaction age 750 million. After clearing the blocker at transaction age 1 billion, the transaction age won't immediately drop to 750 million. It will remain high until the table needing the aggressive vacuum or any transaction with an age over 750 million is completed. During this period, the transaction age of your PostgreSQL cluster will continue to rise. Once the vacuum process is completed, the transaction age will drop to 750 million but will start increasing again until further vacuuming is finished. This cycle will continue as long as these conditions persist, until the transaction age eventually drops to the level configured for your Amazon RDS instance, specified by `autovacuum_freeze_max_age`.

Explanation of the NOTICE messages in RDS for PostgreSQL

The `postgres_get_av_diag()` function provides the following NOTICE messages:

When the age has not reached the monitoring threshold yet

The monitoring threshold for `postgres_get_av_diag()` to identify blockers is 500 million transactions by default. If `postgres_get_av_diag()` generates the following NOTICE, it indicates that the transaction age has not yet reached this threshold.

```
NOTICE: postgres_get_av_diag() checks for blockers that prevent aggressive vacuums
 only, it does so only after exceeding dvb_threshold which is 500,000,000 and age of
 this PostgreSQL cluster is currently at 2.
```

Not connected to the database with the age of oldest transaction ID

The `postgres_get_av_diag()` function provides the most accurate output when connected to the database with the oldest transaction ID age. The database with the oldest transaction ID age reported by `postgres_get_av_diag()` will be different than “my_database” in your case. If you are not connected to the correct database, the following NOTICE is generated:

```
NOTICE: You are not connected to the database with the age of oldest transaction
 ID. Connect to my_database database and run postgres_get_av_diag() for accurate
 reporting.
```

Connecting to the database with the oldest transaction age is important for the following reasons:

- **Identifying temporary table blockers:** Because the metadata for temporary tables is specific to each database, they are typically found in the database where they are created. However, if a temporary table happens to be the top blocker and resides in the database with the oldest transaction, this could be misleading. Connecting to the correct database ensures the accurate identification of the temporary table blocker.
- **Diagnosing slow vacuums:** The index metadata and table count information are database-specific and necessary for diagnosing slow vacuum issues.

Database with oldest transaction by age is on an rdsadmin or template0 database

In certain cases, the `rdsadmin` or `template0` databases may be identified as the database with the oldest transaction ID age. If this happens, `postgres_get_av_diag()` will issue the following NOTICE:

```
NOTICE: The database with the age of oldest transaction ID is rdsadmin or template0,
 reach out to support if the reported blocker is in rdsadmin or template0.
```

Verify that the listed blocker is not originating from either of these two databases. If the blocker is reported to be present in either `rdsadmin` or `template0`, contact support as these databases are not user-accessible and require intervention.

It is highly unlikely for either the `rdsadmin` or `template0` database to contain a top blocker.

When an aggressive vacuum is already running

The `postgres_get_av_diag()` function is designed to report when an aggressive vacuum process is running, but it only triggers this output after the vacuum has been active for at least

1 minute. This intentional delay helps reduce the chances of false positives. By waiting, the function ensures that only effective, significant vacuums are reported, leading to more accurate and reliable monitoring of vacuum activity.

The `postgres_get_av_diag()` function generates the following NOTICE when it detects one or more aggressive vacuums in progress.

```
NOTICE: Your database is currently running aggressive vacuum to prevent wraparound,
 monitor autovacuum performance.
```

As indicated in the NOTICE, continue to monitor the performance of vacuum. For more information about aggressive vacuum see Aggressive vacuum (to prevent wraparound) is running

When autovacuum is off

The `postgres_get_av_diag()` function generates the following NOTICE if autovacuum is disabled on your database instance:

```
NOTICE: Autovacuum is OFF, we strongly recommend to enable it, no restart is
 necessary.
```

Autovacuum is a critical feature of your RDS for PostgreSQL DB instance that ensures smooth database operation. It automatically removes old row versions, reclaims storage space, and prevents table bloat, helping to keep tables and indexes efficient for optimal performance. Additionally, it protects against transaction ID wraparound, which can halt transactions on your Amazon RDS instance. Disabling autovacuum can lead to long-term declines in database performance and stability. We suggest you to keep it on all the times. For more information, see Understanding autovacuum in RDS for PostgreSQL environments.

> **Note**
>
> Turning off autovacuum doesn't stop aggressive vacuums. These will still occur once your tables hit the `autovacuum_freeze_max_age` threshold.

The number of transactions remaining is critically low

The `postgres_get_av_diag()` function generates the following NOTICE when a wraparound vacuum is imminent. This NOTICE is issued when your Amazon RDS instance is 100 million transactions away from potentially rejecting new transactions.

```
WARNING: Number of transactions remaining is critically low, resolve issues with
 autovacuum or perform manual VACUUM FREEZE before your instance stops accepting
 transactions.
```

Your immediate action is required to avoid database downtime. You should closely monitor your vacuuming operations and consider manually initiating a `VACUUM FREEZE` on the affected database to prevent transaction failures.

Managing TOAST OID contention in Amazon RDS for PostgreSQL

TOAST (The Oversized-Attribute Storage Technique) is a PostgreSQL feature designed to handle large data values that exceed the typical 8KB database block size. PostgreSQL doesn't allow physical rows to span multiple blocks. The block size acts as an upper limit on row size. TOAST overcomes this restriction by splitting large field values into smaller chunks. It stores them separately in a dedicated TOAST table linked to the main table. For more information, see the PostgreSQL TOAST storage mechanism and implementation documentation.

Topics

- Understanding TOAST operations
- Identifying performance challenges
- Recommendations
- Monitoring

Understanding TOAST operations

TOAST performs compression and stores large field values out of line. TOAST assigns a unique OID (Object Identifier) to each chunk of oversized data stored in the TOAST table. The main table stores the TOAST value ID and relation ID on the page to reference the corresponding row in the TOAST table. This allows PostgreSQL to efficiently locate and manage these TOAST chunks. However, as the TOAST table grows, the system risks exhausting available OIDs, leading to both performance degradation and potential downtime due to OID depletion.

Object identifiers in TOAST

An Object Identifier (OID) is a system-wide unique identifier used by PostgreSQL to reference database objects like tables, indexes, and functions. These identifiers play a vital role in PostgreSQL's internal operations, allowing the database to efficiently locate and manage objects.

For tables with eligible data sets for toasting, PostgreSQL assigns OIDs to uniquely identify each chunk of oversized data stored in the associated TOAST table. The system associates each chunk with a `chunk_id`, which helps PostgreSQL organize and locate these chunks efficiently within the TOAST table.

Identifying performance challenges

PostgreSQL's OID management relies on a global 32-bit counter so that it wraps around after generating 4 billion unique values. While the database cluster shares this counter, OID allocation involves two steps during TOAST operations:

- **Global counter for allocation** – The global counter assigns a new OID across the cluster.
- **Local search for conflicts** – The TOAST table ensures the new OID does not conflict with existing OIDs already used in that specific table.

Performance degradation can occur when:

- The TOAST table has high fragmentation or dense OID usage, leading to delays in assigning the OID.
- The system frequently allocates and reuses OIDs in environments with high data churn or wide tables that use TOAST extensively.

For more information, see the PostgreSQL TOAST table size limits and OID allocation documentation:

A global counter generates the OIDs and wraps around every 4 billion values, so that from time to time, the system generates an already-used value again. PostgreSQL detects that and tries again with the next OID. A slow INSERT could occur if there is a very long run of used OID values with no gaps in the TOAST table. These challenges become more pronounced as the OID space fills, leading to slower inserts and updates.

Identifying the problem

- Simple `INSERT` statements take significantly longer than usual in an inconsistent and random manner.
- Delays occur only for `INSERT` and `UPDATE` statements involving TOAST operations.
- The following log entries appear in PostgreSQL logs when the system struggles to find available OIDs in TOAST tables:

```
LOG: still searching for an unused OID in relation "pg_toast_20815"
DETAIL: OID candidates have been checked 1000000 times, but no unused OID has been
 found yet.
```

- Performance Insights indicates a high number of average active sessions (AAS) associated with `LWLock:buffer_io` and `LWLock:OidGenLock` wait events.

  You can run the following SQL query to identify long-running INSERT transactions with wait events:

```
SELECT
    datname AS database_name,
    usename AS database_user,
    pid,
    now() - pg_stat_activity.xact_start AS transaction_duration,
    concat(wait_event_type, ':', wait_event) AS wait_event,
    substr(query, 1, 30) AS TRANSACTION,
    state
FROM
    pg_stat_activity
WHERE (now() - pg_stat_activity.xact_start) > INTERVAL '60 seconds'
    AND state IN ('active', 'idle in transaction', 'idle in transaction (aborted)',
 'fastpath function call', 'disabled')
    AND pid <> pg_backend_pid()
AND lower(query) LIKE '%insert%'
ORDER BY
    transaction_duration DESC;
```

  Example query results displaying INSERT operations with extended wait times:

```
 database_name |  database_user  |  pid  | transaction_duration |     wait_event
 |          transaction           | state
```

```
---------------+-----------------+-------+---------------------
+--------------------+--------------------------------+--------
 postgres       | db_admin_user| 70965 | 00:10:19.484061      | LWLock:buffer_io    |
 INSERT INTO "products" (......... | active
 postgres       | db_admin_user| 69878 | 00:06:14.976037      | LWLock:buffer_io    |
 INSERT INTO "products" (......... | active
 postgres       | db_admin_user| 68937 | 00:05:13.942847      | :                   |
 INSERT INTO "products" (......... | active
```

Isolating the problem

- **Test small insert** – Insert a record smaller than the `toast_tuple_target` threshold. Remember that compression is applied before TOAST storage. If this operates without performance issues, the problem is related to TOAST operations.
- **Test new table** – Create a new table with the same structure and insert a record larger than `toast_tuple_target`. If this works without issues, the problem is localized to the original table's OID allocation.

Recommendations

The following approaches can help resolve TOAST OID contention issues.

- **Data cleanup and archive** – Review and delete any obsolete or unnecessary data to free up OIDs for future use, or archive the data. Consider the following limitations:
  - Limited scalability, as future cleanup might not always be possible.
  - Possible long-running VACUUM operation to remove the resulting dead tuples.
- **Write to a new table** – Create a new table for future inserts and use a `UNION ALL` view to combine old and new data for queries. This view presents the combined data from both old and new tables, allowing queries to access them as a single table. Consider the following limitations:
  - Updates on the old table might still cause OID exhaustion.
- **Partition or Shard** – Partition the table or shard data for better scalability and performance. Consider the following limitations:
  - Increased complexity in query logic and maintenance, potential need for application changes to handle partitioned data correctly.

Monitoring

Using system tables

You can use PostgreSQL's system tables to monitor growth of OID usage.

> **Warning**
>
> Depending on the number of OIDs in the TOAST table, it may take time to complete. We recommend that you schedule monitoring during off-business hours to minimize impact.

The following anonymous block counts the number of distinct OIDs used in each TOAST table and displays the parent table information:

```
DO $$
DECLARE
    r record;
    o bigint;
    parent_table text;
    parent_schema text;
BEGIN
    SET LOCAL client_min_messages TO notice;
    FOR r IN
    SELECT
        c.oid,
        c.oid::regclass AS toast_table
    FROM
        pg_class c
    WHERE
        c.relkind = 't'
        AND c.relowner != 10 LOOP
            -- Fetch the number of distinct used OIDs (chunk IDs) from the TOAST table
            EXECUTE 'SELECT COUNT(DISTINCT chunk_id) FROM ' || r.toast_table INTO o;
            -- If there are used OIDs, find the associated parent table and its schema
            IF o <> 0 THEN
                SELECT
                    n.nspname,
                    c.relname INTO parent_schema,
                    parent_table
                FROM
                    pg_class c
```

```
                    JOIN pg_namespace n ON c.relnamespace = n.oid
                WHERE
                    c.reltoastrelid = r.oid;
                -- Raise a concise NOTICE message
                RAISE NOTICE 'Parent schema: % | Parent table: % | Toast table: %
 | Number of used OIDs: %', parent_schema, parent_table, r.toast_table, TO_CHAR(o,
 'FM9,999,999,999,999');
            END IF;
        END LOOP;
END
$$;
```

Example output displaying OID usage statistics by TOAST table:

```
NOTICE:  Parent schema: public | Parent table: my_table | Toast table:
 pg_toast.pg_toast_16559 | Number of used OIDs: 45,623,317
NOTICE:  Parent schema: public | Parent table: my_table1 | Toast table:
 pg_toast.pg_toast_45639925 | Number of used OIDs: 10,000
NOTICE:  Parent schema: public | Parent table: my_table2 | Toast table:
 pg_toast.pg_toast_45649931 | Number of used OIDs: 1,000,000
DO
```

The following anonymous block retrieves the maximum assigned OID for each non-empty TOAST table:

```
DO $$
DECLARE
    r record;
    o bigint;
    parent_table text;
    parent_schema text;
BEGIN
    SET LOCAL client_min_messages TO notice;
    FOR r IN
    SELECT
        c.oid,
        c.oid::regclass AS toast_table
    FROM
        pg_class c
    WHERE
        c.relkind = 't'
        AND c.relowner != 10 LOOP
            -- Fetch the max(chunk_id) from the TOAST table
```

```
            EXECUTE 'SELECT max(chunk_id) FROM ' || r.toast_table INTO o;
            -- If there's at least one TOASTed chunk, find the associated parent table
 and its schema
            IF o IS NOT NULL THEN
                SELECT
                    n.nspname,
                    c.relname INTO parent_schema,
                    parent_table
                FROM
                    pg_class c
                    JOIN pg_namespace n ON c.relnamespace = n.oid
                WHERE
                    c.reltoastrelid = r.oid;
                -- Raise a concise NOTICE message
                RAISE NOTICE 'Parent schema: % | Parent table: % | Toast table:
 % | Max chunk_id: %', parent_schema, parent_table, r.toast_table, TO_CHAR(o,
 'FM9,999,999,999,999');
            END IF;
        END LOOP;
END
$$;
```

Example output displaying maximum chunk IDs for TOAST tables:

```
NOTICE:  Parent schema: public | Parent table: my_table | Toast table:
 pg_toast.pg_toast_16559 | Max chunk_id: 45,639,907
NOTICE:  Parent schema: public | Parent table: my_table1 | Toast table:
 pg_toast.pg_toast_45639925 | Max chunk_id: 45,649,929
NOTICE:  Parent schema: public | Parent table: my_table2 | Toast table:
 pg_toast.pg_toast_45649931 | Max chunk_id: 46,649,935
DO
```

Using Performance Insights

The wait events `LWLock:buffer_io` and `LWLock:OidGenLock` appear in Performance Insights during operations that require assigning new Object Identifiers (OIDs). High Average Active Sessions (AAS) for these events typically point to contention during OID assignment and resource management. This is particularly common in environments with high data churn, extensive large data usage, or frequent object creation.

LWLock:buffer_io

`LWLock:buffer_io` is a wait event that occurs when a PostgreSQL session is waiting for I/O operations on a shared buffer to complete. This typically happens when the database reads data from disk into memory or writes modified pages from memory to disk. The `BufferIO` wait event ensures consistency by preventing multiple processes from accessing or modifying the same buffer while I/O operations are in progress. High occurrences of this wait event may indicate disk bottlenecks or excessive I/O activity in the database workload.

During TOAST operations:

- PostgreSQL allocates OIDs for large objects and ensures their uniqueness by scanning the TOAST table's index.
- Large TOAST indexes may require accessing multiple pages to verify OID uniqueness. This results in increased disk I/O, especially when the buffer pool cannot cache all required pages.

The size of the index directly affects the number of buffer pages that need to be accessed during these operations. Even if the index is not bloated, its sheer size can increase buffer I/O, particularly in high-concurrency or high-churn environments. For more information, see LWLock:BufferIO wait event troubleshooting guide.

LWLock:OidGenLock

`OidGenLock` is a wait event that occurs when a PostgreSQL session is waiting to allocate a new object identifier (OID). This lock ensures that OIDs are generated sequentially and safely, allowing only one process to generate OIDs at a time.

During TOAST operations:

- **OID allocation for chunks in TOAST table** – PostgreSQL assigns OIDs to chunks in TOAST tables when managing large data records. Each OID must be unique to prevent conflicts in the system catalog.
- **High concurrency** – Since access to OID generator is sequential, when multiple sessions are concurrently creating objects that require OIDs, contention for `OidGenLock` can occur. This increases the likelihood of sessions waiting for OID allocation to complete.
- **Dependency on system catalog access** – Allocating OIDs requires updates to shared system catalog tables like `pg_class` and `pg_type`. If these tables experience heavy activity (due to frequent DDL operations), it can increase lock contention for `OidGenLock`.

- **High OID allocation demand** – TOAST heavy workloads with large data records require constant OID allocation, increasing contention.

Additional factors that increase OID contention:

- **Frequent object creation** – Workloads that frequently create and drop objects, such as temporary tables, amplify contention on the global OID counter.
- **Global counter locking** – The global OID counter is accessed serially to ensure uniqueness, creating a single point of contention in high-concurrency environments.

Working with logging mechanisms supported by RDS for PostgreSQL

There are several parameters, extensions, and other configurable items that you can set to log activities that occur on your PostgreSQL DB instance. These include the following:

- The `log_statement` parameter can be used to log user activity in your PostgreSQL database. To learn more about RDS for PostgreSQL logging and how to monitor the logs, see RDS for PostgreSQL database log files.
- The `rds.force_admin_logging_level` parameter logs actions by the Amazon RDS internal user (rdsadmin) in the databases on the DB instance. It writes the output to the PostgreSQL error log. Allowed values are `disabled`, `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, log, `fatal`, and `panic`. The default value is `disabled`.
- The `rds.force_autovacuum_logging_level` parameter can be set to capture various autovacuum operations in the PostgreSQL error log. For more information, see Logging autovacuum and vacuum activities.
- The PostgreSQL Audit (pgAudit) extension can be installed and configured to capture activities at the session level or at the object level. For more information, see Using pgAudit to log database activity.
- The `log_fdw` extension makes it possible for you to access the database engine log using SQL. For more information, see Using the log_fdw extension to access the DB log using SQL.
- The `pg_stat_statements` library is specified as the default for the `shared_preload_libraries` parameter in RDS for PostgreSQL version 10 and higher. It's this library that you can use to analyze running queries. Be sure that `pg_stat_statements` is set in your DB parameter group. For more information about monitoring your RDS for PostgreSQL DB instance using the information that this library provides, see SQL statistics for RDS PostgreSQL.

- The `log_hostname` parameter captures to the log the hostname of each client connection. For RDS for PostgreSQL version 12 and higher versions, this parameter is set to `off` by default. If you turn it on, be sure to monitor session connection times. When turned on, the service uses the domain name system (DNS) reverse lookup request to get the hostname of the client that's making the connection and add it to the PostgreSQL log. This has a noticeable impact during session connection. We recommend that you turn on this parameter for troubleshooting purposes only.

In general terms, the point of logging is so that the DBA can monitor, tune performance, and troubleshoot. Many of the logs are uploaded automatically to Amazon CloudWatch or Performance Insights. Here, they're sorted and grouped to provide complete metrics for your DB instance. To learn more about Amazon RDS monitoring and metrics, see Monitoring metrics in an Amazon RDS instance.

Managing temporary files with PostgreSQL

In PostgreSQL, a complex query might perform several sort or hash operations at the same time, with each operation using instance memory to store results up to the value specified in the `work_mem` parameter. When the instance memory is not sufficient, temporary files are created to store the results. These are written to disk to complete the query execution. Later, these files are automatically removed after the query completes. In RDS for PostgreSQL, these files are stored in Amazon EBS on the data volume. For more information, see Amazon RDS DB instance storage. You can monitor the `FreeStorageSpace` metric published in CloudWatch to make sure that your DB instance has enough free storage space. For more information, see `FreeStorageSpace` .

We recommend using Amazon RDS Optimized Read instances for workloads involving multiple concurrent queries that increase the usage of temporary files. These instances use local Non-Volatile Memory Express (NVMe) based solid state drive (SSD) block-level storage to place the temporary files. For more information, see Improving query performance for RDS for PostgreSQL with Amazon RDS Optimized Reads.

You can use the following parameters and functions to manage the temporary files in your instance.

- **`temp_file_limit`** – This parameter cancels any query exceeding the size of temp_files in KB. This limit prevents any query from running endlessly and consuming disk space with temporary files. You can estimate the value using the results from the `log_temp_files` parameter. As a

best practice, examine the workload behavior and set the limit according to the estimation. The following example shows how a query is canceled when it exceeds the limit.

```
postgres=>select * from pgbench_accounts, pg_class, big_table;
```

```
ERROR: temporary file size exceeds temp_file_limit (64kB)
```

- **log_temp_files** – This parameter sends messages to the postgresql.log when the temporary files of a session are removed. This parameter produces logs after a query successfully completes. Therefore, it might not help in troubleshooting active, long-running queries.

  The following example shows that when the query successfully completes, the entries are logged in the postgresql.log file while the temporary files are cleaned up.

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
```

- **pg_ls_tmpdir** – This function that is available from RDS for PostgreSQL 13 and above provides visibility into the current temporary file usage. The completed query doesn't appear in the results of the function. In the following example, you can view the results of this function.

```
postgres=>select * from pg_ls_tmpdir();
```

```
      name       |    size    |      modification
-----------------+------------+------------------------
```

```
 pgsql_tmp8355.1 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.0 | 1072250880 | 2023-02-06 22:54:43+00
 pgsql_tmp8327.0 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.1 |  703168512 | 2023-02-06 22:54:56+00
 pgsql_tmp8355.0 | 1072250880 | 2023-02-06 22:54:00+00
 pgsql_tmp8328.1 |  835031040 | 2023-02-06 22:54:56+00
 pgsql_tmp8328.0 | 1072250880 | 2023-02-06 22:54:40+00
(7 rows)
```

```
postgres=>select query from pg_stat_activity where pid = 8355;

query
-------------------------------------------------------------------------------------
select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid
(1 row)
```

The file name includes the processing ID (PID) of the session that generated the temporary file. A more advanced query, such as in the following example, performs a sum of the temporary files for each PID.

```
postgres=>select replace(left(name, strpos(name, '.')-1),'pgsql_tmp','') as pid,
 count(*), sum(size) from pg_ls_tmpdir() group by pid;
```

```

 pid  | count |   sum
------+-------------------
 8355 |     2 | 2144501760
 8351 |     2 | 2090770432
 8327 |     1 | 1072250880
 8328 |     2 | 2144501760
(4 rows)
```

- **pg_stat_statements** – If you activate the pg_stat_statements parameter, then you can view the average temporary file usage per call. You can identify the query_id of the query and use it to examine the temporary file usage as shown in the following example.

```
postgres=>select queryid from pg_stat_statements where query like 'select a.aid from
 pgbench%';
```

```
       queryid
----------------------
 -7170349228837045701
(1 row)
```

```
postgres=>select queryid, substr(query,1,25), calls, temp_blks_read/calls
 temp_blks_read_per_call, temp_blks_written/calls temp_blks_written_per_call from
 pg_stat_statements where queryid = -7170349228837045701;
```

```
       queryid        |          substr           | calls | temp_blks_read_per_call |
 temp_blks_written_per_call
----------------------+---------------------------+-------+-------------------------
+---------------------------
 -7170349228837045701 | select a.aid from pgbench |    50 |                  239226 |
                    388678
(1 row)
```

- **`Performance Insights`** – In the Performance Insights dashboard, you can view temporary file usage by turning on the metrics **temp_bytes** and **temp_files**. Then, you can see the average of both of these metrics and see how they correspond to the query workload. The view within Performance Insights doesn't show specifically the queries that are generating the temporary files. However, when you combine Performance Insights with the query shown for `pg_ls_tmpdir`, you can troubleshoot, analyze, and determine the changes in your query workload.

  For more information about how to analyze metrics and queries with Performance Insights, see Analyzing metrics with the Performance Insights dashboard.

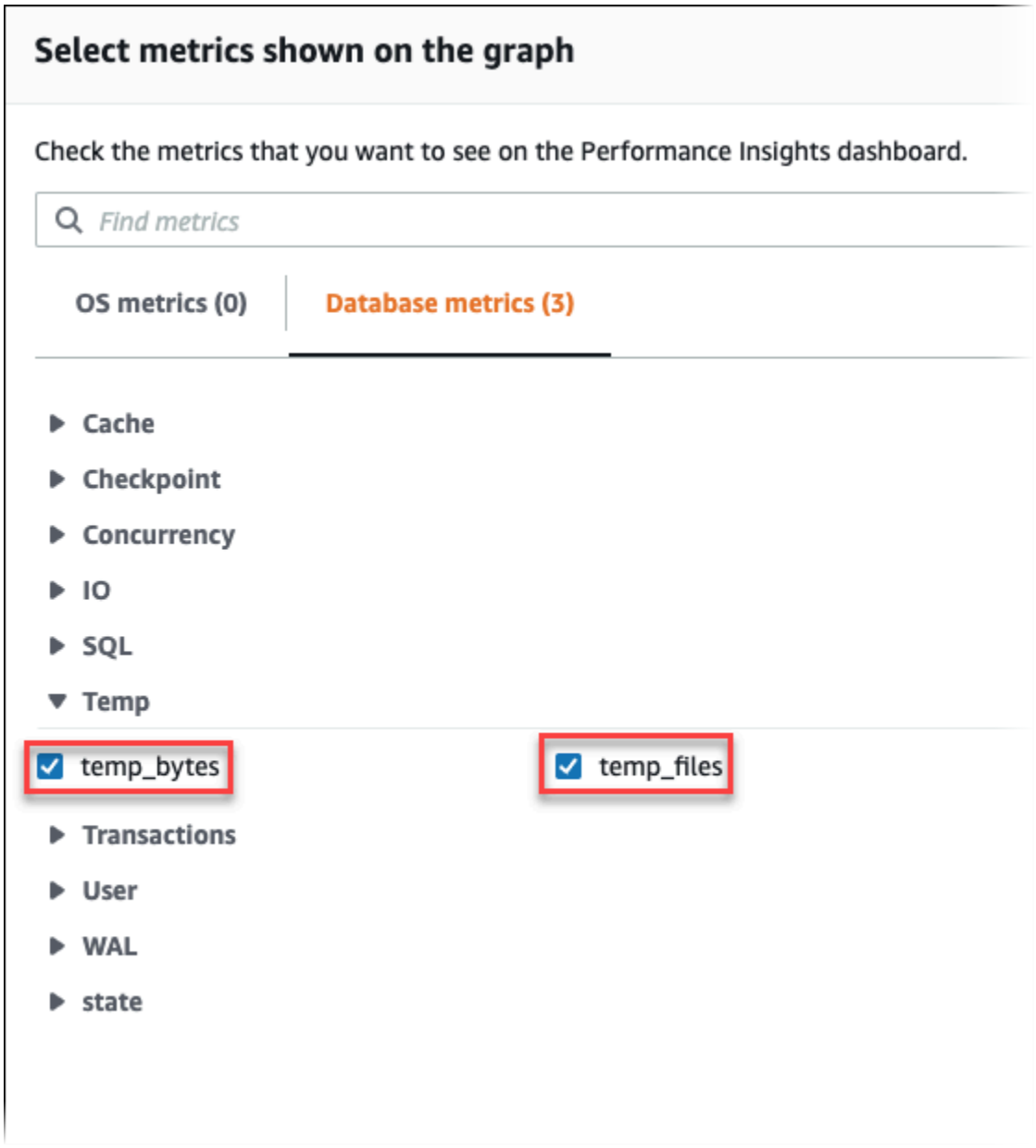
  For an example of viewing temporary file usage with Performance Insights, see Viewing temporary file usage with Performance Insights

Viewing temporary file usage with Performance Insights

You can use Performance Insights to view temporary file usage by turning on the metrics **temp_bytes** and **temp_files**. The view in Performance Insights doesn't show the specific queries

that generate temporary files, however, when you combine Performance Insights with the query shown for `pg_ls_tmpdir`, you can troubleshoot, analyze, and determine the changes in your query workload.

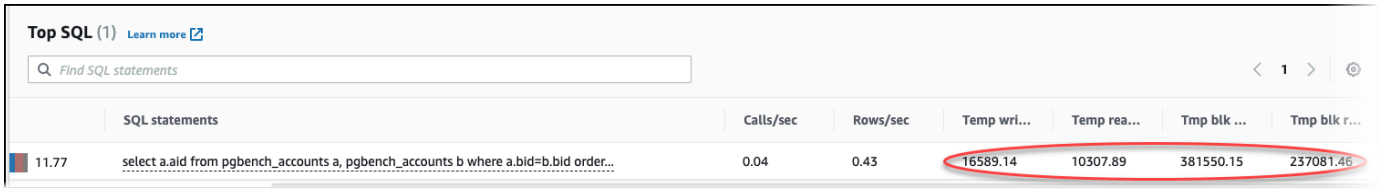
1. In the Performance Insights dashboard, choose **Manage Metrics**.
2. Choose **Database metrics**, and select the **temp_bytes** and **temp_files** metrics as shown in the following image.



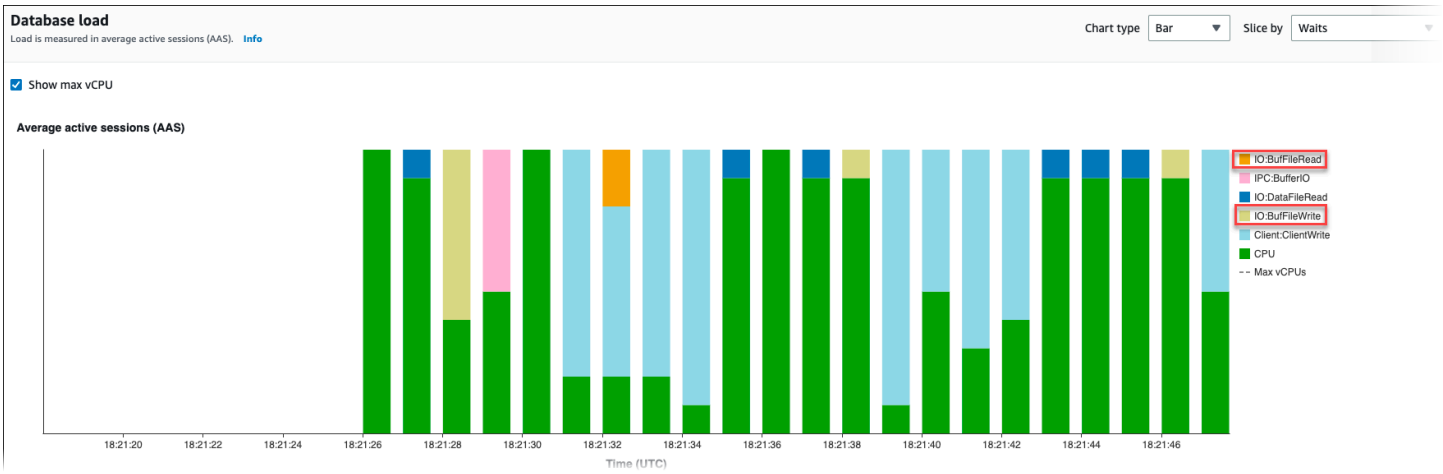
3. In the **Top SQL** tab, choose the **Preferences** icon.
4. In the **Preferences** window, turn on the following statistics to appear in the **Top SQL**tab and choose **Continue**.

- Temp writes/sec
- Temp reads/sec
- Tmp blk write/call
- Tmp blk read/call

5. The temporary file is broken out when combined with the query shown for `pg_ls_tmpdir`, as shown in the following example.



The `IO:BufFileRead` and `IO:BufFileWrite` events occur when the top queries in your workload often create temporary files. You can use Performance Insights to identify top queries waiting on `IO:BufFileRead` and `IO:BufFileWrite` by reviewing Average Active Session (AAS) in Database Load and Top SQL sections.



For more information on how to analyze top queries and load by wait event with Performance Insights, see Overview of the Top SQL tab. You should identify and tune the queries that cause increase in temporary file usage and related wait events. For more information on these wait events and remediation, see IO:BufFileRead and IO:BufFileWrite.

> **Note**
>
> The `work_mem` parameter controls when the sort operation runs out of memory and results are written into temporary files. We recommend that you don't change the setting of this

parameter higher than the default value because it would permit every database session to consume more memory. Also, a single session that performs complex joins and sorts can perform parallel operations in which each operation consumes memory.
As a best practice, when you have a large report with multiple joins and sorts, set this parameter at the session level by using the `SET work_mem` command. Then the change is only applied to the current session and doesn't change the value globally.

Using pgBadger for log analysis with PostgreSQL

You can use a log analyzer such as pgBadger to analyze PostgreSQL logs. The pgBadger documentation states that the %l pattern (the log line for the session or process) should be a part of the prefix. However, if you provide the current RDS `log_line_prefix` as a parameter to pgBadger it should still produce a report.

For example, the following command correctly formats an Amazon RDS for PostgreSQL log file dated 2014-02-04 using pgBadger.

```
./pgbadger -f stderr -p '%t:%r:%u@%d:[%p]:' postgresql.log.2014-02-04-00
```

Using PGSnapper for monitoring PostgreSQL

You can use PGSnapper to assist with periodic collection of Amazon RDS for PostgreSQL performance-related statistics and metrics. For more information, see Monitor Amazon RDS for PostgreSQL performance using PGSnapper.

Managing custom casts in RDS for PostgreSQL

Type casting in PostgreSQL is the process of converting a value from one data type to another. PostgreSQL provides built-in casts for many common conversions, but you can also create custom casts to define how specific type conversions should behave.

A cast specifies how to perform a conversion from one data type to another. For example, converting text `'123'` to integer 123, or numeric `45.67` to text `'45.67'`.

For comprehensive information about PostgreSQL casting concepts and syntax, refer to the PostgreSQL CREATE CAST Documentation.

Starting with RDS for PostgreSQL versions 13.23, 14.20, 15.15, 16.11, 17.7, and 18.1, you can use the rds_casts extension to install additional casts for built-in types, while still being able to create your own casts for custom types.

Topics

- Installing and using the rds_casts extension
- Supported casts
- Creating or dropping casts
- Creating custom casts with proper context strategy

Installing and using the rds_casts extension

To create the `rds_casts` extension, connect to your RDS for PostgreSQL DB instance as an `rds_superuser` and run the following command:

```
CREATE EXTENSION IF NOT EXISTS rds_casts;
```

Supported casts

Create the extension in each database where you want to use custom casts. After creating the extension, use the following command to view all available casts:

```
SELECT * FROM rds_casts.list_supported_casts();
```

This function lists the available cast combinations (source type, target type, coercion context, and cast function). For example, if you want to create `text` to `numeric` as an `implicit` cast. You can use the following query to find if the cast is available to create:

```
SELECT * FROM rds_casts.list_supported_casts()
WHERE source_type = 'text' AND target_type = 'numeric';
 id | source_type | target_type |          qualified_function          |
 coercion_context
----+-------------+-------------+--------------------------------------
+------------------
 10 | text        | numeric     | rds_casts.rds_text_to_numeric_custom | implicit
 11 | text        | numeric     | rds_casts.rds_text_to_numeric_custom | assignment
 13 | text        | numeric     | rds_casts.rds_text_to_numeric_custom | explicit
 20 | text        | numeric     | rds_casts.rds_text_to_numeric_inout  | implicit
```

```
 21 | text        | numeric     | rds_casts.rds_text_to_numeric_inout  | assignment
 23 | text        | numeric     | rds_casts.rds_text_to_numeric_inout  | explicit
```

The rds_casts extension provides two types of conversion functions for each cast:

- *_inout functions* - Use PostgreSQL's standard I/O conversion mechanism, behaving identically to casts created with the INOUT method
- *_custom functions* - Provide enhanced conversion logic that handles edge cases, such as converting empty strings to NULL values to avoid conversion errors

The `inout` functions replicate PostgreSQL's native casting behavior, while `custom` functions extend this functionality by handling scenarios that standard INOUT casts cannot accommodate, such as converting empty strings to integers.

Creating or dropping casts

You can create and drop supported casts using two methods:

Cast creation

Method 1: Using native CREATE CAST command

```
CREATE CAST (text AS numeric)
WITH FUNCTION rds_casts.rds_text_to_numeric_custom
AS IMPLICIT;
```

Method 2: Using the rds_casts.create_cast function

```
SELECT rds_casts.create_cast(10);
```

The `create_cast` function takes the ID from the `list_supported_casts()` output. This method is simpler and ensures you're using the correct function and context combination. This id is guaranteed to remain the same across different postgres versions.

To verify the cast was created successfully, query the pg_cast system catalog:

```
SELECT oid, castsource::regtype, casttarget::regtype, castfunc::regproc, castcontext,
 castmethod
FROM pg_cast
WHERE castsource = 'text'::regtype AND casttarget = 'numeric'::regtype;
```

```
  oid   | castsource | casttarget |               castfunc               | castcontext
 | castmethod
--------+------------+------------+--------------------------------------+-------------
+------------
 356372 | text       | numeric    | rds_casts.rds_text_to_numeric_custom | i
 | f
```

The `castcontext` column shows: `e` for EXPLICIT, `a` for ASSIGNMENT, or `i` for IMPLICIT.

Dropping casts

Method 1: Using DROP CAST command

```
DROP CAST IF EXISTS (text AS numeric);
```

Method 2: Using the rds_casts.drop_cast function

```
SELECT rds_casts.drop_cast(10);
```

The `drop_cast` function takes the same ID used when creating the cast. This method ensures you're dropping the exact cast that was created with the corresponding ID.

Creating custom casts with proper context strategy

When creating multiple casts for integer types, operator ambiguity errors can occur if all casts are created as IMPLICIT. The following example demonstrates this issue by creating two implicit casts from text to different integer widths:

```
-- Creating multiple IMPLICIT casts causes ambiguity
postgres=> CREATE CAST (text AS int4) WITH FUNCTION
 rds_casts.rds_text_to_int4_custom(text) AS IMPLICIT;
CREATE CAST
postgres=> CREATE CAST (text AS int8) WITH FUNCTION
 rds_casts.rds_text_to_int8_custom(text) AS IMPLICIT;
CREATE CAST

postgres=> CREATE TABLE test_cast(col int);
CREATE TABLE
postgres=> INSERT INTO test_cast VALUES ('123'::text);
INSERT 0 1
postgres=> SELECT * FROM test_cast WHERE col='123'::text;
```

```
ERROR:  operator is not unique: integer = text
LINE 1: SELECT * FROM test_cast WHERE col='123'::text;
                                            ^
HINT:  Could not choose a best candidate operator. You might need to add explicit type
 casts.
```

The error occurs because PostgreSQL cannot determine which implicit cast to use when comparing an integer column with a text value. Both the int4 and int8 implicit casts are valid candidates, creating ambiguity.

To avoid this operator ambiguity, use ASSIGNMENT context for smaller integer widths and IMPLICIT context for larger integer widths:

```
-- Use ASSIGNMENT for smaller integer widths
CREATE CAST (text AS int2)
WITH FUNCTION rds_casts.rds_text_to_int2_custom(text)
AS ASSIGNMENT;

CREATE CAST (text AS int4)
WITH FUNCTION rds_casts.rds_text_to_int4_custom(text)
AS ASSIGNMENT;

-- Use IMPLICIT for larger integer widths
CREATE CAST (text AS int8)
WITH FUNCTION rds_casts.rds_text_to_int8_custom(text)
AS IMPLICIT;

postgres=> INSERT INTO test_cast VALUES ('123'::text);
INSERT 0 1
postgres=> SELECT * FROM test_cast WHERE col='123'::text;
 col
-----
 123
(1 row)
```

With this strategy, only the int8 cast is implicit, so PostgreSQL can unambiguously determine which cast to use.

Working with parameters on your RDS for PostgreSQL DB instance

In some cases, you might create an RDS for PostgreSQL DB instance without specifying a custom parameter group. If so, your DB instance is created using the default parameter group for

the version of PostgreSQL that you choose. For example, suppose that you create an RDS for PostgreSQL DB instance using PostgreSQL 13.3. In this case, the DB instance is created using the values in the parameter group for PostgreSQL 13 releases, `default.postgres13`.

You can also create your own custom DB parameter group. You need to do this if you want to modify any settings for the RDS for PostgreSQL DB instance from their default values. To learn how, see Parameter groups for Amazon RDS.

You can track the settings on your RDS for PostgreSQL DB instance in several different ways. You can use the AWS Management Console, the AWS CLI, or the Amazon RDS API. You can also query the values from the PostgreSQL `pg_settings` table of your instance, as shown following.

```
SELECT name, setting, boot_val, reset_val, unit
 FROM pg_settings
 ORDER BY name;
```

To learn more about the values returned from this query, see `pg_settings` in the PostgreSQL documentation.

Be especially careful when changing the settings for `max_connections` and `shared_buffers` on your RDS for PostgreSQL DB instance. For example, suppose that you modify settings for `max_connections` or `shared_buffers` and you use values that are too high for your actual workload. In this case, your RDS for PostgreSQL DB instance won't start. If this happens, you see an error such as the following in the `postgres.log`.

```
2018-09-18 21:13:15 UTC::@:[8097]:FATAL:  could not map anonymous shared memory: Cannot
 allocate memory
2018-09-18 21:13:15 UTC::@:[8097]:HINT:  This error usually means that PostgreSQL's
 request for a shared memory segment
exceeded available memory or swap space. To reduce the request size (currently
 3514134274048 bytes), reduce
PostgreSQL's shared memory usage, perhaps by reducing shared_buffers or
 max_connections.
```

However, you can't change any values of the settings contained in the default RDS for PostgreSQL DB parameter groups. To change settings for any parameters, first create a custom DB parameter group. Then change the settings in that custom group, and then apply the custom parameter group to your RDS for PostgreSQL DB instance. To learn more, see Parameter groups for Amazon RDS.

There are two types of parameters in RDS for PostgreSQL.

- **Static parameters** – Static parameters require that the RDS for PostgreSQL DB instance be rebooted after a change so that the new value can take effect.
- **Dynamic parameters** – Dynamic parameters don't require a reboot after changing their settings.

> **Note**
>
> If your RDS for PostgreSQL DB instance is using your own custom DB parameter group, you can change the values of dynamic parameters on the running DB instance. You can do this by using the AWS Management Console, the AWS CLI, or the Amazon RDS API.

If you have privileges to do so, you can also change parameter values by using the `ALTER DATABASE`, `ALTER ROLE`, and `SET` commands.

RDS for PostgreSQL DB instance parameter list

The following table lists some (but not all) parameters available in an RDS for PostgreSQL DB instance. To view all available parameters, you use the describe-db-parameters AWS CLI command. For example, to get the list of all parameters available in the default parameter group for RDS for PostgreSQL version 13, run the following.

```
aws rds describe-db-parameters --db-parameter-group-name default.postgres13
```

You can also use the Console. Choose **Parameter groups** from the Amazon RDS menu, and then choose the parameter group from those available in your AWS Region.

Parameter name	Apply_Type	Description
application_name	Dynamic	Sets the application name to be reported in statistics and logs.
archive_command	Dynamic	Sets the shell command that will be called to archive a WAL file.
array_nulls	Dynamic	Enables input of NULL elements in arrays.
authentication_timeout	Dynamic	Sets the maximum allowed time to complete client authentication.
autovacuum	Dynamic	Starts the autovacuum subprocess.
autovacuum_analyze_scale_factor	Dynamic	Number of tuple inserts, updates, or deletes before analyze as a fraction of reltuples.
autovacuum_analyze_threshold	Dynamic	Minimum number of tuple inserts, updates, or deletes before analyze.
autovacuum_freeze_max_age	Static	Age at which to autovacuum a table to prevent transaction ID wraparound.
autovacuum_naptime	Dynamic	Time to sleep between autovacuum runs.

Parameter name	Apply_Type	Description
autovacuum_max_workers	Static	Sets the maximum number of simultaneously running autovacuum worker processes.
autovacuum_vacuum_cost_delay	Dynamic	Vacuum cost delay, in milliseconds, for autovacuum.
autovacuum_vacuum_cost_limit	Dynamic	Vacuum cost amount available before napping, for autovacuum.
autovacuum_vacuum_scale_factor	Dynamic	Number of tuple updates or deletes before vacuum as a fraction of reltuples.
autovacuum_vacuum_threshold	Dynamic	Minimum number of tuple updates or deletes before vacuum.
backslash_quote	Dynamic	Sets whether a backslash (\) is allowed in string literals.
bgwriter_delay	Dynamic	Background writer sleep time between rounds.
bgwriter_lru_maxpages	Dynamic	Background writer maximum number of LRU pages to flush per round.
bgwriter_lru_multiplier	Dynamic	Multiple of the average buffer usage to free per round.
bytea_output	Dynamic	Sets the output format for bytes.
check_function_bodies	Dynamic	Checks function bodies during CREATE FUNCTION.
checkpoint_completion_target	Dynamic	Time spent flushing dirty buffers during checkpoint, as a fraction of the checkpoint interval.

Parameter name	Apply_Type	Description
checkpoint_segments	Dynamic	Sets the maximum distance in log segments between automatic write-ahead log (WAL) checkpoints.
checkpoint_timeout	Dynamic	Sets the maximum time between automatic WAL checkpoints.
checkpoint_warning	Dynamic	Enables warnings if checkpoint segments are filled more frequently than this.
client_connection_check_interval	Dynamic	Sets the time interval between checks for disconnection while running queries.
client_encoding	Dynamic	Sets the client's character set encoding.
client_min_messages	Dynamic	Sets the message levels that are sent to the client.
commit_delay	Dynamic	Sets the delay in microseconds between transaction commit and flushing WAL to disk.
commit_siblings	Dynamic	Sets the minimum concurrent open transactions before performing commit_delay.
constraint_exclusion	Dynamic	Enables the planner to use constraints to optimize queries.
cpu_index_tuple_cost	Dynamic	Sets the planner's estimate of the cost of processing each index entry during an index scan.
cpu_operator_cost	Dynamic	Sets the planner's estimate of the cost of processing each operator or function call.
cpu_tuple_cost	Dynamic	Sets the planner's estimate of the cost of processing each tuple (row).
cursor_tuple_fraction	Dynamic	Sets the planner's estimate of the fraction of a cursor's rows that will be retrieved.

Parameter name	Apply_Type	Description
datestyle	Dynamic	Sets the display format for date and time values.
deadlock_timeout	Dynamic	Sets the time to wait on a lock before checking for deadlock.
debug_pretty_print	Dynamic	Indents parse and plan tree displays.
debug_print_parse	Dynamic	Logs each query's parse tree.
debug_print_plan	Dynamic	Logs each query's execution plan.
debug_print_rewritten	Dynamic	Logs each query's rewritten parse tree.
default_statistics_target	Dynamic	Sets the default statistics target.
default_tablespace	Dynamic	Sets the default tablespace to create tables and indexes in.
default_transaction_deferrable	Dynamic	Sets the default deferrable status of new transactions.
default_transaction_isolation	Dynamic	Sets the transaction isolation level of each new transaction.
default_transaction_read_only	Dynamic	Sets the default read-only status of new transactions.
default_with_oids	Dynamic	Creates new tables with object IDs (OIDs) by default.
effective_cache_size	Dynamic	Sets the planner's assumption about the size of the disk cache.
effective_io_concurrency	Dynamic	Number of simultaneous requests that can be handled efficiently by the disk subsystem.

Parameter name	Apply_Type	Description
enable_bitmapscan	Dynamic	Enables the planner's use of bitmap-scan plans.
enable_hashagg	Dynamic	Enables the planner's use of hashed aggregation plans.
enable_hashjoin	Dynamic	Enables the planner's use of hash join plans.
enable_indexscan	Dynamic	Enables the planner's use of index-scan plans.
enable_material	Dynamic	Enables the planner's use of materialization.
enable_mergejoin	Dynamic	Enables the planner's use of merge join plans.
enable_nestloop	Dynamic	Enables the planner's use of nested-loop join plans.
enable_seqscan	Dynamic	Enables the planner's use of sequential-scan plans.
enable_sort	Dynamic	Enables the planner's use of explicit sort steps.
enable_tidscan	Dynamic	Enables the planner's use of TID scan plans.
escape_string_warning	Dynamic	Warns about backslash (\) escapes in ordinary string literals.
extra_float_digits	Dynamic	Sets the number of digits displayed for floating-point values.
from_collapse_limit	Dynamic	Sets the FROM-list size beyond which subqueries are not collapsed.
fsync	Dynamic	Forces synchronization of updates to disk.
full_page_writes	Dynamic	Writes full pages to WAL when first modified after a checkpoint.
geqo	Dynamic	Enables genetic query optimization.

Parameter name	Apply_Type	Description
geqo_effort	Dynamic	GEQO: effort is used to set the default for other GEQO parameters.
geqo_generations	Dynamic	GEQO: number of iterations of the algorithm.
geqo_pool_size	Dynamic	GEQO: number of individuals in the population.
geqo_seed	Dynamic	GEQO: seed for random path selection.
geqo_selection_bias	Dynamic	GEQO: selective pressure within the population.
geqo_threshold	Dynamic	Sets the threshold of FROM items beyond which GEQO is used.
gin_fuzzy_search_limit	Dynamic	Sets the maximum allowed result for exact search by GIN.
hot_standby_feedback	Dynamic	Determines whether a hot standby sends feedback messages to the primary or upstream standby.
intervalstyle	Dynamic	Sets the display format for interval values.
join_collapse_limit	Dynamic	Sets the FROM-list size beyond which JOIN constructs are not flattened.
lc_messages	Dynamic	Sets the language in which messages are displayed.
lc_monetary	Dynamic	Sets the locale for formatting monetary amounts.
lc_numeric	Dynamic	Sets the locale for formatting numbers.
lc_time	Dynamic	Sets the locale for formatting date and time values.

Parameter name	Apply_Type	Description
log_autovacuum_min_duration	Dynamic	Sets the minimum running time above which autovacuum actions will be logged.
log_checkpoints	Dynamic	Logs each checkpoint.
log_connections	Dynamic	Logs each successful connection.
log_disconnections	Dynamic	Logs end of a session, including duration.
log_duration	Dynamic	Logs the duration of each completed SQL statement.
log_error_verbosity	Dynamic	Sets the verbosity of logged messages.
log_executor_stats	Dynamic	Writes executor performance statistics to the server log.
log_filename	Dynamic	Sets the file name pattern for log files.
log_file_mode	Dynamic	Sets file permissions for log files. Default value is 0644.
log_hostname	Dynamic	Logs the host name in the connection logs. As of PostgreSQL 12 and later versions, this parameter is 'off' by default. When turned on, the connection uses DNS reverse-lookup to get the hostname that gets captured to the connection logs. If you turn on this parameter, you should monitor the impact that it has on the time it takes to establish connections.
log_line_prefix	Dynamic	Controls information prefixed to each log line.
log_lock_waits	Dynamic	Logs long lock waits.
log_min_duration_statement	Dynamic	Sets the minimum running time above which statements will be logged.

Parameter name	Apply_Type	Description
log_min_error_statement	Dynamic	Causes all statements generating an error at or above this level to be logged.
log_min_messages	Dynamic	Sets the message levels that are logged.
log_parser_stats	Dynamic	Writes parser performance statistics to the server log.
log_planner_stats	Dynamic	Writes planner performance statistics to the server log.
log_rotation_age	Dynamic	Automatic log file rotation will occur after N minutes.
log_rotation_size	Dynamic	Automatic log file rotation will occur after N kilobytes.
log_statement	Dynamic	Sets the type of statements logged.
log_statement_stats	Dynamic	Writes cumulative performance statistics to the server log.
log_temp_files	Dynamic	Logs the use of temporary files larger than this number of kilobytes.
log_timezone	Dynamic	Sets the time zone to use in log messages.
log_truncate_on_rotation	Dynamic	Truncate existing log files of same name during log rotation.
logging_collector	Static	Start a subprocess to capture stderr output and/or csvlogs into log files.
maintenance_work_mem	Dynamic	Sets the maximum memory to be used for maintenance operations.

Parameter name	Apply_Type	Description
max_connections	Static	Sets the maximum number of concurrent connections.
max_files_per_process	Static	Sets the maximum number of simultaneously open files for each server process.
max_locks_per_transaction	Static	Sets the maximum number of locks per transaction.
max_pred_locks_per_transaction	Static	Sets the maximum number of predicate locks per transaction.
max_prepared_transactions	Static	Sets the maximum number of simultaneously prepared transactions.
max_stack_depth	Dynamic	Sets the maximum stack depth, in kilobytes.
max_standby_archive_delay	Dynamic	Sets the maximum delay before canceling queries when a hot standby server is processing archived WAL data.
max_standby_streaming_delay	Dynamic	Sets the maximum delay before canceling queries when a hot standby server is processing streamed WAL data.

Parameter name	Apply_Type	Description

max_wal_size	Dynamic	Sets the WAL size (MB) that triggers a checkpoint.
		• For RDS for PostgreSQL 15 and earlier versions, it is 2 GB.
		• For RDS for PostgreSQL 16 and later versions, RDS for PostgreSQL automatically configures based on the allocated storage size of your instance:
		  • 6 GB for instances with allocated storage 100 GB and more.
		  • 2 GB for instances with allocated storage less than 100 GB.
		
		Use the following command on your Amazon RDS for PostgreSQL DB instance to see its current value:
		`SHOW max_wal_size;`

min_wal_size	Dynamic	Sets the minimum size to shrink the WAL to. For PostgreSQL version 9.6 and earlier, min_wal_size is in units of 16 MB. For PostgreSQL version 10 and later, min_wal_size is in units of 1 MB.
quote_all_identifiers	Dynamic	Adds quotes (") to all identifiers when generating SQL fragments.
random_page_cost	Dynamic	Sets the planner's estimate of the cost of a non-sequentially fetched disk page. This parameter has no value unless query plan management (QPM) is turned on. When QPM is on, the default value for this parameter 4.

Parameter name	Apply_Type	Description
rds.adaptive_autovacuum	Dynamic	Automatically tunes the autovacuum parameters whenever the transaction ID thresholds are exceeded.
rds.force_ssl	Dynamic	Requires the use of SSL connections. The default value is set to 1 (on) for RDS for PostgreSQL version 15. All other RDS for PostgreSQL major version 14 and older have the default value set to 0 (off).
rds.local_volume_spill_enabled	Static	Enables writing logical spill files to the local volume.
rds.log_retention_period	Dynamic	Sets log retention such that Amazon RDS deletes PostgreSQL logs that are older than *n* minutes.
rds.rds_superuser_reserved_connections	Static	Sets the number of connection slots reserved for rds_superusers. This parameter is only available in versions 15 and earlier. For more information, see the PostgreSQL documentation reserved_connections.
rds.replica_identity_full	Dynamic	When you set this parameter to on, it overrides the replica identity setting to FULL for all database tables. This means all column values are written to the write ahead log (WAL), regardless of your REPLICA IDENTITY FULL settings. **Note** Turning on this parameter may increase your database instance IOPS due to the additional WAL logging.

Parameter name	Apply_Type	Description
rds.restrict_password_commands	Static	Restricts who can manage passwords to users with the rds_password role. Set this parameter to 1 to enable password restriction. The default is 0.
search_path	Dynamic	Sets the schema search order for names that are not schema-qualified.
seq_page_cost	Dynamic	Sets the planner's estimate of the cost of a sequentially fetched disk page.
session_replication_role	Dynamic	Sets the sessions behavior for triggers and rewrite rules.
shared_buffers	Static	Sets the number of shared memory buffers used by the server.
shared_preload_libraries	Static	Lists the shared libraries to preload into the RDS for PostgreSQL DB instance. Supported values include auto_explain, orafce, pgaudit, pglogical, pg_bigm, pg_cron, pg_hint_plan, pg_prewarm, pg_similarity, pg_stat_statements, pg_tle, pg_transport, plprofiler, and plrust.
ssl	Dynamic	Enables SSL connections.
sql_inheritance	Dynamic	Causes subtables to be included by default in various commands.
ssl_renegotiation_limit	Dynamic	Sets the amount of traffic to send and receive before renegotiating the encryption keys.
standard_conforming_strings	Dynamic	Causes ... strings to treat backslashes literally.

Parameter name	Apply_Type	Description
statement_timeout	Dynamic	Sets the maximum allowed duration of any statement.
synchronize_seqscans	Dynamic	Enables synchronized sequential scans.
synchronous_commit	Dynamic	Sets the current transactions synchronization level.
tcp_keepalives_count	Dynamic	Maximum number of TCP keepalive retransmits.
tcp_keepalives_idle	Dynamic	Time between issuing TCP keepalives.
tcp_keepalives_interval	Dynamic	Time between TCP keepalive retransmits.
temp_buffers	Dynamic	Sets the maximum number of temporary buffers used by each session.
temp_file_limit	Dynamic	Sets the maximum size in KB to which the temporary files can grow.
temp_tablespaces	Dynamic	Sets the tablespaces to use for temporary tables and sort files.

Parameter name	Apply_Type	Description
timezone	Dynamic	Sets the time zone for displaying and interpreting time stamps. The Internet Assigned Numbers Authority (IANA) publishes new time zones at https://www.iana.org/time-zones several times a year. Every time RDS releases a new minor maintenance release of PostgreSQL, it ships with the latest time zone data at the time of the release. When you use the latest RDS for PostgreSQL versions, you have recent time zone data from RDS. To ensure that your DB instance has recent time zone data, we recommend upgrading to a higher DB engine version. You can't modify the time zone tables in PostgreSQL DB instances manually. RDS doesn't modify or reset the time zone data of running DB instances. New time zone data is installed only when you perform a database engine version upgrade.
track_activities	Dynamic	Collects information about running commands.
track_activity_query_size	Static	Sets the size reserved for pg_stat_activity.current_query, in bytes.
track_counts	Dynamic	Collects statistics on database activity.
track_functions	Dynamic	Collects function-level statistics on database activity.
track_io_timing	Dynamic	Collects timing statistics on database I/O activity.
transaction_deferrable	Dynamic	Indicates whether to defer a read-only serializable transaction until it can be started with no possible serialization failures.

Parameter name	Apply_Type	Description
transaction_isolation	Dynamic	Sets the current transactions isolation level.
transaction_read_only	Dynamic	Sets the current transactions read-only status.
transform_null_equals	Dynamic	Treats expr=NULL as expr IS NULL.
update_process_title	Dynamic	Updates the process title to show the active SQL command.
vacuum_cost_delay	Dynamic	Vacuum cost delay in milliseconds.
vacuum_cost_limit	Dynamic	Vacuum cost amount available before napping.
vacuum_cost_page_dirty	Dynamic	Vacuum cost for a page dirtied by vacuum.
vacuum_cost_page_hit	Dynamic	Vacuum cost for a page found in the buffer cache.
vacuum_cost_page_miss	Dynamic	Vacuum cost for a page not found in the buffer cache.
vacuum_defer_cleanup_age	Dynamic	Number of transactions by which vacuum and hot cleanup should be deferred, if any.
vacuum_freeze_min_age	Dynamic	Minimum age at which vacuum should freeze a table row.
vacuum_freeze_table_age	Dynamic	Age at which vacuum should scan a whole table to freeze tuples.
wal_buffers	Static	Sets the number of disk-page buffers in shared memory for WAL.
wal_writer_delay	Dynamic	WAL writer sleep time between WAL flushes.

Parameter name	Apply_Type	Description
work_mem	Dynamic	Sets the maximum memory to be used for query workspaces.
xmlbinary	Dynamic	Sets how binary values are to be encoded in XML.
xmloption	Dynamic	Sets whether XML data in implicit parsing and serialization operations is to be considered as documents or content fragments.

Amazon RDS uses the default PostgreSQL units for all parameters. The following table shows the PostgreSQL default unit for each parameter.

Parameter name	Unit
archive_timeout	s
authentication_timeout	s
autovacuum_naptime	s
autovacuum_vacuum_cost_delay	ms
bgwriter_delay	ms
checkpoint_timeout	s
checkpoint_warning	s
deadlock_timeout	ms
effective_cache_size	8 KB
lock_timeout	ms
log_autovacuum_min_duration	ms

Parameter name	Unit
log_min_duration_statement	ms
log_rotation_age	minutes
log_rotation_size	KB
log_temp_files	KB
maintenance_work_mem	KB
max_stack_depth	KB
max_standby_archive_delay	ms
max_standby_streaming_delay	ms
post_auth_delay	s
pre_auth_delay	s
segment_size	8 KB
shared_buffers	8 KB
statement_timeout	ms
ssl_renegotiation_limit	KB
tcp_keepalives_idle	s
tcp_keepalives_interval	s
temp_file_limit	KB
work_mem	KB
temp_buffers	8 KB
vacuum_cost_delay	ms

Parameter name	Unit
wal_buffers	8 KB
wal_receiver_timeout	ms
wal_segment_size	B
wal_sender_timeout	ms
wal_writer_delay	ms
wal_receiver_status_interval	s

Tuning with wait events for RDS for PostgreSQL

Wait events are an important tuning tool for RDS for PostgreSQL. When you can find out why sessions are waiting for resources and what they are doing, you're better able to reduce bottlenecks. You can use the information in this section to find possible causes and corrective actions. This section also discusses basic PostgreSQL tuning concepts.

The wait events in this section are specific to RDS for PostgreSQL.

Topics

- Essential concepts for RDS for PostgreSQL tuning
- RDS for PostgreSQL wait events
- Client:ClientRead
- Client:ClientWrite
- CPU
- IO:BufFileRead and IO:BufFileWrite
- IO:DataFileRead
- IO:WALWrite
- IPC:parallel wait events
- IPC:ProcArrayGroupUpdate
- Lock:advisory
- Lock:extend
- Lock:Relation
- Lock:transactionid
- Lock:tuple
- LWLock:BufferMapping (LWLock:buffer_mapping)
- LWLock:BufferIO (IPC:BufferIO)
- LWLock:buffer_content (BufferContent)
- LWLock:lock_manager (LWLock:lockmanager)
- LWLock:pg_stat_statements
- LWLock:SubtransSLRU (LWLock:SubtransControlLock)
- Timeout:PgSleep
- Timeout:VacuumDelay

Essential concepts for RDS for PostgreSQL tuning

Before you tune your RDS for PostgreSQL database, make sure to learn what wait events are and why they occur. Also review the basic memory and disk architecture of RDS for PostgreSQL. For a helpful architecture diagram, see the PostgreSQL wikibook.

Topics

- RDS for PostgreSQL wait events
- RDS for PostgreSQL memory
- RDS for PostgreSQL processes

RDS for PostgreSQL wait events

A *wait event* is an indication that the session is waiting for a resource. For example, the wait event `Client:ClientRead` occurs when RDS for PostgreSQL is waiting to receive data from the client. Sessions typically wait for resources such as the following.

- Single-threaded access to a buffer, for example, when a session is attempting to modify a buffer
- A row that is currently locked by another session
- A data file read
- A log file write

For example, to satisfy a query, the session might perform a full table scan. If the data isn't already in memory, the session waits for the disk I/O to complete. When the buffers are read into memory, the session might need to wait because other sessions are accessing the same buffers. The database records the waits by using a predefined wait event. These events are grouped into categories.

By itself, a single wait event doesn't indicate a performance problem. For example, if requested data isn't in memory, reading data from disk is necessary. If one session locks a row for an update, another session waits for the row to be unlocked so that it can update it. A commit requires waiting for the write to a log file to complete. Waits are integral to the normal functioning of a database.

On the other hand, large numbers of wait events typically show a performance problem. In such cases, you can use wait event data to determine where sessions are spending time. For example, if a report that typically runs in minutes now takes hours to run, you can identify the wait events that

contribute the most to total wait time. If you can determine the causes of the top wait events, you can sometimes make changes that improve performance. For example, if your session is waiting on a row that has been locked by another session, you can end the locking session.

RDS for PostgreSQL memory

RDS for PostgreSQL memory is divided into shared and local.

Topics

- Shared memory in RDS for PostgreSQL
- Local memory in RDS for PostgreSQL

Shared memory in RDS for PostgreSQL

RDS for PostgreSQL allocates shared memory when the instance starts. Shared memory is divided into multiple subareas. The following sections provide descriptions of the most important ones.

Topics

- Shared buffers
- Write ahead log (WAL) buffers

Shared buffers

The *shared buffer pool* is an RDS for PostgreSQL memory area that holds all pages that are or were being used by application connections. A *page* is the memory version of a disk block. The shared buffer pool caches the data blocks read from disk. The pool reduces the need to reread data from disk, making the database operate more efficiently.

Every table and index is stored as an array of pages of a fixed size. Each block contains multiple tuples, which correspond to rows. A tuple can be stored in any page.

The shared buffer pool has finite memory. If a new request requires a page that isn't in memory, and no more memory exists, RDS for PostgreSQL evicts a less frequently used page to accommodate the request. The eviction policy is implemented by a clock sweep algorithm.

The `shared_buffers` parameter determines how much memory the server dedicates to caching data. The default value is set to `{DBInstanceClassMemory/32768}` bytes, based on the available memory for the DB instance.

Write ahead log (WAL) buffers

A *write-ahead log (WAL) buffer* holds transaction data that RDS for PostgreSQL later writes to persistent storage. Using the WAL mechanism, RDS for PostgreSQL can do the following:

- Recover data after a failure
- Reduce disk I/O by avoiding frequent writes to disk

When a client changes data, RDS for PostgreSQL writes the changes to the WAL buffer. When the client issues a `COMMIT`, the WAL writer process writes transaction data to the WAL file.

The `wal_level` parameter determines how much information is written to the WAL, with possible values such as `minimal`, `replica`, and `logical`.

Local memory in RDS for PostgreSQL

Every backend process allocates local memory for query processing.

Topics

- Work memory area
- Maintenance work memory area
- Temporary buffer area

Work memory area

The *work memory area* holds temporary data for queries that performs sorts and hashes. For example, a query with an `ORDER BY` clause performs a sort. Queries use hash tables in hash joins and aggregations.

The `work_mem` parameter the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files, measured in megabytes. The default value is 4 MB. Multiple sessions can run simultaneously, and each session can run maintenance operations in parallel. For this reason, the total work memory used can be multiples of the `work_mem` setting.

Maintenance work memory area

The *maintenance work memory area* caches data for maintenance operations. These operations include vacuuming, creating an index, and adding foreign keys.

The `maintenance_work_mem` parameter specifies the maximum amount of memory to be used by maintenance operations, measured in megabytes. The default value is 64 MB. A database session can only run one maintenance operation at a time.

Temporary buffer area

The *temporary buffer area* caches temporary tables for each database session.

Each session allocates temporary buffers as needed up to the limit you specify. When the session ends, the server clears the buffers.

The `temp_buffers` parameter sets the maximum number of temporary buffers used by each session, measured in megabytes. The default value is 8 MB. Before the first use of temporary tables within a session, you can change the `temp_buffers` value.

RDS for PostgreSQL processes

RDS for PostgreSQL uses multiple processes.

Topics

- Postmaster process
- Backend processes
- Background processes

Postmaster process

The *postmaster process* is the first process started when you start RDS for PostgreSQL. The postmaster process has the following primary responsibilities:

- Fork and monitor background processes
- Receive authentication requests from client processes, and authenticate them before allowing the database to service requests

Backend processes

If the postmaster authenticates a client request, the postmaster forks a new backend process, also called a postgres process. One client process connects to exactly one backend process. The client process and the backend process communicate directly without intervention by the postmaster process.

Background processes

The postmaster process forks several processes that perform different backend tasks. Some of the more important include the following:

- WAL writer

  RDS for PostgreSQL writes data in the WAL (write ahead logging) buffer to the log files. The principle of write ahead logging is that the database can't write changes to the data files until after the database writes log records describing those changes to disk. The WAL mechanism reduces disk I/O, and allows RDS for PostgreSQL to use the logs to recover the database after a failure.

- Background writer

  This process periodically write dirty (modified) pages from the memory buffers to the data files. A page becomes dirty when a backend process modifies it in memory.

- Autovacuum daemon

  The daemon consists of the following:

  - The autovacuum launcher
  - The autovacuum worker processes

  When autovacuum is turned on, it checks for tables that have had a large number of inserted, updated, or deleted tuples. The daemon has the following responsibilities:

  - Recover or reuse disk space occupied by updated or deleted rows
  - Update statistics used by the planner
  - Protect against loss of old data because of transaction ID wraparound

  The autovacuum feature automates the execution of `VACUUM` and `ANALYZE` commands. `VACUUM` has the following variants: standard and full. Standard vacuum runs in parallel with other database operations. `VACUUM FULL` requires an exclusive lock on the table it is working on. Thus, it can't run in parallel with operations that access the same table. `VACUUM` creates a substantial amount of I/O traffic, which can cause poor performance for other active sessions.

RDS for PostgreSQL wait events

The following table lists the wait events for RDS for PostgreSQL that most commonly indicate performance problems, and summarizes the most common causes and corrective actions..

Wait event	Definition
Client:ClientRead	This event occurs when RDS for PostgreSQL is waiting to receive data from the client.
Client:ClientWrite	This event occurs when RDS for PostgreSQL is waiting to write data to the client.
CPU	This event occurs when a thread is active in CPU or is waiting for CPU.
IO:BufFileRead and IO:BufFileWrite	These events occur when RDS for PostgreSQL creates temporary files.
IO:DataFileRead	This event occurs when a connection waits on a backend process to read a required page from storage because the page isn't available in shared memory.
IO:WALWrite	This event occurs when RDS for PostgreSQL is waiting for the write-ahead log (WAL) buffers to be written to a WAL file.
Lock:advisory	This event occurs when a PostgreSQL application uses a lock to coordinate activity across multiple sessions.
Lock:extend	This event occurs when a backend process is waiting to lock a relation to extend it while another process has a lock on that relation for the same purpose.

Wait event	Definition
Lock:Relation	This event occurs when a query is waiting to acquire a lock on a table or view that's currently locked by another transaction.
Lock:transactionid	This event occurs when a transaction is waiting for a row-level lock.
Lock:tuple	This event occurs when a backend process is waiting to acquire a lock on a tuple.
LWLock:BufferMapping (LWLock:buffer_mapping)	This event occurs when a session is waiting to associate a data block with a buffer in the shared buffer pool.
LWLock:BufferIO (IPC:BufferIO)	This event occurs when RDS for PostgreSQL is waiting for other processes to finish their input/output (I/O) operations when concurrently trying to access a page.
LWLock:buffer_content (BufferContent)	This event occurs when a session is waiting to read or write a data page in memory while another session has that page locked for writing.
LWLock:lock_manager (LWLock:lockmanager)	This event occurs when the RDS for PostgreSQL engine maintains the shared lock's memory area to allocate, check, and deallocate a lock when a fast path lock isn't possible.
LWLock:SubtransSLRU (LWLock:SubtransControlLock)	This event occurs when a process is waiting to access the simple least-recently used (SLRU) cache for a subtransaction.
Timeout:PgSleep	This event occurs when a server process has called the `pg_sleep` function and is waiting for the sleep timeout to expire.

Wait event	Definition
Timeout:VacuumDelay	This event indicates that the vacuum process is sleeping because the estimated cost limit has been reached.

Client:ClientRead

The `Client:ClientRead` event occurs when RDS for PostgreSQL is waiting to receive data from the client.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for RDS for PostgreSQL version 10 and higher.

Context

An RDS for PostgreSQL DB instance is waiting to receive data from the client. The RDS for PostgreSQL DB instance must receive the data from the client before it can send more data to the client. The time that the instance waits before receiving data from the client is a `Client:ClientRead` event.

Likely causes of increased waits

Common causes for the `Client:ClientRead` event to appear in top waits include the following:

Increased network latency

There might be increased network latency between the RDS for PostgreSQL DB instance and client. Higher network latency increases the time required for DB instance to receive data from the client.

Increased load on the client

There might be CPU pressure or network saturation on the client. An increase in load on the client can delay transmission of data from the client to the RDS for PostgreSQL DB instance.

Excessive network round trips

A large number of network round trips between the RDS for PostgreSQL DB instance and the client can delay transmission of data from the client to the RDS for PostgreSQL DB instance.

Large copy operation

During a copy operation, the data is transferred from the client's file system to the RDS for PostgreSQL DB instance. Sending a large amount of data to the DB instance can delay transmission of data from the client to the DB instance.

Idle client connection

When a client connects to the RDS for PostgreSQL DB instance in an `idle in transaction` state, the DB instance might wait for the client to send more data or issue a command. A connection in this state can lead to an increase in `Client:ClientRead` events.

PgBouncer used for connection pooling

PgBouncer has a low-level network configuration setting called `pkt_buf`, which is set to 4,096 by default. If the workload is sending query packets larger than 4,096 bytes through PgBouncer, we recommend increasing the `pkt_buf` setting to 8,192. If the new setting doesn't decrease the number of `Client:ClientRead` events, we recommend increasing the `pkt_buf` setting to larger values, such as 16,384 or 32,768. If the query text is large, the larger setting can be particularly helpful.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- Place the clients in the same Availability Zone and VPC subnet as the instance
- Scale your client
- Use current generation instances
- Increase network bandwidth

- Monitor maximums for network performance
- Monitor for transactions in the "idle in transaction" state

Place the clients in the same Availability Zone and VPC subnet as the instance

To reduce network latency and increase network throughput, place clients in the same Availability Zone and virtual private cloud (VPC) subnet as the RDS for PostgreSQL DB instance. Make sure that the clients are as geographically close to the DB instance as possible.

Scale your client

Using Amazon CloudWatch or other host metrics, determine if your client is currently constrained by CPU or network bandwidth, or both. If the client is constrained, scale your client accordingly.

Use current generation instances

In some cases, you might not be using a DB instance class that supports jumbo frames. If you're running your application on Amazon EC2, consider using a current generation instance for the client. Also, configure the maximum transmission unit (MTU) on the client operating system. This technique might reduce the number of network round trips and increase network throughput. For more information, see Jumbo frames (9001 MTU) in the *Amazon EC2 User Guide*.

For information about DB instance classes, see DB instance classes. To determine the DB instance class that is equivalent to an Amazon EC2 instance type, place `db.` before the Amazon EC2 instance type name. For example, the `r5.8xlarge` Amazon EC2 instance is equivalent to the `db.r5.8xlarge` DB instance class.

Increase network bandwidth

Use `NetworkReceiveThroughput` and `NetworkTransmitThroughput` Amazon CloudWatch metrics to monitor incoming and outgoing network traffic on the DB instance. These metrics can help you to determine if network bandwidth is sufficient for your workload.

If your network bandwidth isn't enough, increase it. If the AWS client or your DB instance is reaching the network bandwidth limits, the only way to increase the bandwidth is to increase your DB instance size. For more information, see DB instance class types.

For more information about CloudWatch metrics, see Amazon CloudWatch metrics for Amazon RDS.

Monitor maximums for network performance

If you are using Amazon EC2 clients, Amazon EC2 provides maximums for network performance metrics, including aggregate inbound and outbound network bandwidth. It also provides connection tracking to ensure that packets are returned as expected and link-local services access for services such as the Domain Name System (DNS). To monitor these maximums, use a current enhanced networking driver and monitor network performance for your client.

For more information, see Monitor network performance for your Amazon EC2 instance in the *Amazon EC2 User Guide* and Monitor network performance for your Amazon EC2 instance in the *Amazon EC2 User Guide*.

Monitor for transactions in the "idle in transaction" state

Check whether you have an increasing number of `idle in transaction` connections. To do this, monitor the `state` column in the `pg_stat_activity` table. You might be able to identify the connection source by running a query similar to the following.

```
select client_addr, state, count(1) from pg_stat_activity
where state like 'idle in transaction%'
group by 1,2
order by 3 desc
```

Client:ClientWrite

The `Client:ClientWrite` event occurs when RDS for PostgreSQL is waiting to write data to the client.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for RDS for PostgreSQL version 10 and higher.

Context

A client process must read all of the data received from an RDS for PostgreSQL DB cluster before the cluster can send more data. The time that the cluster waits before sending more data to the client is a `Client:ClientWrite` event.

Reduced network throughput between the RDS for PostgreSQL DB instance and the client can cause this event. CPU pressure and network saturation on the client can also cause this event. *CPU pressure* is when the CPU is fully utilized and there are tasks waiting for CPU time. *Network saturation* is when the network between the database and client is carrying more data than it can handle.

Likely causes of increased waits

Common causes for the `Client:ClientWrite` event to appear in top waits include the following:

Increased network latency

There might be increased network latency between the RDS for PostgreSQL DB instance and client. Higher network latency increases the time required for the client to receive the data.

Increased load on the client

There might be CPU pressure or network saturation on the client. An increase in load on the client delays the reception of data from the RDS for PostgreSQL DB instance.

Large volume of data sent to the client

The RDS for PostgreSQL DB instance might be sending a large amount of data to the client. A client might not be able to receive the data as fast as the cluster is sending it. Activities such as a copy of a large table can result in an increase in `Client:ClientWrite` events.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- Place the clients in the same Availability Zone and VPC subnet as the cluster
- Use current generation instances
- Reduce the amount of data sent to the client

- Scale your client

Place the clients in the same Availability Zone and VPC subnet as the cluster

To reduce network latency and increase network throughput, place clients in the same Availability Zone and virtual private cloud (VPC) subnet as the RDS for PostgreSQL DB instance.

Use current generation instances

In some cases, you might not be using a DB instance class that supports jumbo frames. If you're running your application on Amazon EC2, consider using a current generation instance for the client. Also, configure the maximum transmission unit (MTU) on the client operating system. This technique might reduce the number of network round trips and increase network throughput. For more information, see Jumbo frames (9001 MTU) in the *Amazon EC2 User Guide*.

For information about DB instance classes, see DB instance classes. To determine the DB instance class that is equivalent to an Amazon EC2 instance type, place `db.` before the Amazon EC2 instance type name. For example, the `r5.8xlarge` Amazon EC2 instance is equivalent to the `db.r5.8xlarge` DB instance class.

Reduce the amount of data sent to the client

When possible, adjust your application to reduce the amount of data that the RDS for PostgreSQL DB instance sends to the client. Making such adjustments relieves CPU and network contention on the client.

Scale your client

Using Amazon CloudWatch or other host metrics, determine if your client is currently constrained by CPU or network bandwidth, or both. If the client is constrained, scale your client accordingly.

CPU

This event occurs when a thread is active in CPU or is waiting for CPU.

Topics

- Supported engine versions
- Context

- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is relevant for all all versions of RDS for PostgreSQL.

Context

The *central processing unit (CPU)* is the component of a computer that runs instructions. For example, CPU instructions perform arithmetic operations and exchange data in memory. If a query increases the number of instructions that it performs through the database engine, the time spent running the query increases. *CPU scheduling* is giving CPU time to a process. Scheduling is orchestrated by the kernel of the operating system.

Topics

- How to tell when this wait occurs
- DBLoadCPU metric
- os.cpuUtilization metrics
- Likely cause of CPU scheduling

How to tell when this wait occurs

This CPU wait event indicates that a backend process is active in CPU or is waiting for CPU. You know that it's occurring when a query shows the following information:

- The `pg_stat_activity.state` column has the value `active`.
- The `wait_event_type` and `wait_event` columns in `pg_stat_activity` are both `null`.

To see the backend processes that are using or waiting on CPU, run the following query.

```
SELECT *
FROM   pg_stat_activity
WHERE  state = 'active'
AND    wait_event_type IS NULL
AND    wait_event IS NULL;
```

DBLoadCPU metric

The Performance Insights metric for CPU is DBLoadCPU. The value for DBLoadCPU can differ from the value for the Amazon CloudWatch metric CPUUtilization. The latter metric is collected from the HyperVisor for a database instance.

os.cpuUtilization metrics

Performance Insights operating-system metrics provide detailed information about CPU utilization. For example, you can display the following metrics:

- os.cpuUtilization.nice.avg
- os.cpuUtilization.total.avg
- os.cpuUtilization.wait.avg
- os.cpuUtilization.idle.avg

Performance Insights reports the CPU usage by the database engine as os.cpuUtilization.nice.avg.

Likely cause of CPU scheduling

The operating system (OS) kernel handles scheduling for the CPU. When the CPU is *active*, a process might need to wait to get scheduled. The CPU is active while it's performing computations. It's also active while it has an idle thread that it's not running, that is, an idle thread that's waiting on memory I/O. This type of I/O dominates the typical database workload.

Processes are likely to wait to get scheduled on a CPU when the following conditions are met:

- The CloudWatch CPUUtilization metric is near 100 percent.
- The average load is greater than the number of vCPUs, indicating a heavy load. You can find the loadAverageMinute metric in the OS metrics section in Performance Insights.

Likely causes of increased waits

When the CPU wait event occurs more than normal, possibly indicating a performance problem, typical causes include the following.

Topics

- Likely causes of sudden spikes

- Likely causes of long-term high frequency
- Corner cases

Likely causes of sudden spikes

The most likely causes of sudden spikes are as follows:

- Your application has opened too many simultaneous connections to the database. This scenario is known as a "connection storm."
- Your application workload changed in any of the following ways:
  - New queries
  - An increase in the size of your dataset
  - Index maintenance or creation
  - New functions
  - New operators
  - An increase in parallel query execution
- Your query execution plans have changed. In some cases, a change can cause an increase in buffers. For example, the query is now using a sequential scan when it previously used an index. In this case, the queries need more CPU to accomplish the same goal.

Likely causes of long-term high frequency

The most likely causes of events that recur over a long period:

- Too many backend processes are running concurrently on CPU. These processes can be parallel workers.
- Queries are performing suboptimally because they need a large number of buffers.

Corner cases

If none of the likely causes turn out to be actual causes, the following situations might be occurring:

- The CPU is swapping processes in and out.
- The CPU might be managing page table entries if the *huge pages* feature has been turned off. This memory management feature is turned on by default for all DB instance classes other than

micro, small, and medium DB instance classes. For more information, see Huge pages for RDS for PostgreSQL.

Actions

If the CPU wait event dominates database activity, it doesn't necessarily indicate a performance problem. Respond to this event only when performance degrades.

Topics

- Investigate whether the database is causing the CPU increase
- Determine whether the number of connections increased
- Respond to workload changes

Investigate whether the database is causing the CPU increase

Examine the `os.cpuUtilization.nice.avg` metric in Performance Insights. If this value is far less than the CPU usage, nondatabase processes are the main contributor to CPU.

Determine whether the number of connections increased

Examine the `DatabaseConnections` metric in Amazon CloudWatch. Your action depends on whether the number increased or decreased during the period of increased CPU wait events.

The connections increased

If the number of connections went up, compare the number of backend processes consuming CPU to the number of vCPUs. The following scenarios are possible:

- The number of backend processes consuming CPU is less than the number of vCPUs.

  In this case, the number of connections isn't an issue. However, you might still try to reduce CPU utilization.
- The number of backend processes consuming CPU is greater than the number of vCPUs.

  In this case, consider the following options:

  - Decrease the number of backend processes connected to your database. For example, implement a connection pooling solution such as RDS Proxy. To learn more, see Amazon RDS Proxy.

- Upgrade your instance size to get a higher number of vCPUs.
- Redirect some read-only workloads to reader nodes, if applicable.

The connections didn't increase

Examine the `blks_hit` metrics in Performance Insights. Look for a correlation between an increase in `blks_hit` and CPU usage. The following scenarios are possible:

- CPU usage and `blks_hit` are correlated.

  In this case, find the top SQL statements that are linked to the CPU usage, and look for plan changes. You can use either of the following techniques:

  - Explain the plans manually and compare them to the expected execution plan.
  - Look for an increase in block hits per second and local block hits per second. In the **Top SQL** section of Performance Insights dashboard, choose **Preferences**.
- CPU usage and `blks_hit` aren't correlated.

  In this case, determine whether any of the following occurs:

  - The application is rapidly connecting to and disconnecting from the database.

    Diagnose this behavior by turning on `log_connections` and `log_disconnections`, then analyzing the PostgreSQL logs. Consider using the `pgbadger` log analyzer. For more information, see https://github.com/darold/pgbadger.
  - The OS is overloaded.

    In this case, Performance Insights shows that backend processes are consuming CPU for a longer time than usual. Look for evidence in the Performance Insights `os.cpuUtilization` metrics or the CloudWatch `CPUUtilization` metric. If the operating system is overloaded, look at Enhanced Monitoring metrics to diagnose further. Specifically, look at the process list and the percentage of CPU consumed by each process.
  - Top SQL statements are consuming too much CPU.

    Examine statements that are linked to the CPU usage to see whether they can use less CPU. Run an `EXPLAIN` command, and focus on the plan nodes that have the most impact. Consider using a PostgreSQL execution plan visualizer. To try out this tool, see http://explain.dalibo.com/.

Respond to workload changes

If your workload has changed, look for the following types of changes:

New queries

Check whether the new queries are expected. If so, ensure that their execution plans and the number of executions per second are expected.

An increase in the size of the data set

Determine whether partitioning, if it's not already implemented, might help. This strategy might reduce the number of pages that a query needs to retrieve.

Index maintenance or creation

Check whether the schedule for the maintenance is expected. A best practice is to schedule maintenance activities outside of peak activities.

New functions

Check whether these functions perform as expected during testing. Specifically, check whether the number of executions per second is expected.

New operators

Check whether they perform as expected during the testing.

An increase in running parallel queries

Determine whether any of the following situations has occurred:

- The relations or indexes involved have suddenly grown in size so that they differ significantly from `min_parallel_table_scan_size` or `min_parallel_index_scan_size`.
- Recent changes have been made to `parallel_setup_cost` or `parallel_tuple_cost`.
- Recent changes have been made to `max_parallel_workers` or `max_parallel_workers_per_gather`.

IO:BufFileRead and IO:BufFileWrite

The `IO:BufFileRead` and `IO:BufFileWrite` events occur when RDS for PostgreSQL creates temporary files. When operations require more memory than the working memory parameters currently define, they write temporary data to persistent storage. This operation is sometimes

called *spilling to disk*. For more information about the temporary files and their usage, see Managing temporary files with PostgreSQL.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

`IO:BufFileRead` and `IO:BufFileWrite` relate to the work memory area and maintenance work memory area. For more information about these local memory areas, see Resource Consumption in the PostgreSQL documentation.

The default value for `work_mem` is 4 MB. If one session performs operations in parallel, each worker handling the parallelism uses 4 MB of memory. For this reason, set `work_mem` carefully. If you increase the value too much, a database running many sessions might consume too much memory. If you set the value too low, RDS for PostgreSQL creates temporary files in local storage. The disk I/O for these temporary files can reduce performance.

If you observe the following sequence of events, your database might be generating temporary files:

1. Sudden and sharp decreases in availability
2. Fast recovery for the free space

You might also see a "chainsaw" pattern. This pattern can indicate that your database is creating small files constantly.

Likely causes of increased waits

In general, these wait events are caused by operations that consume more memory than the `work_mem` or `maintenance_work_mem` parameters allocate. To compensate, the operations write

to temporary files. Common causes for the `IO:BufFileRead` and `IO:BufFileWrite` events include the following:

Queries that need more memory than exists in the work memory area

Queries with the following characteristics use the work memory area:

- Hash joins
- `ORDER BY` clause
- `GROUP BY` clause
- `DISTINCT`
- Window functions
- `CREATE TABLE AS SELECT`
- Materialized view refresh

Statements that need more memory than exists in the maintenance work memory area

The following statements use the maintenance work memory area:

- `CREATE INDEX`
- `CLUSTER`

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- Identify the problem
- Examine your join queries
- Examine your ORDER BY and GROUP BY queries
- Avoid using the DISTINCT operation
- Consider using window functions instead of GROUP BY functions
- Investigate materialized views and CTAS statements
- Use pg_repack when you rebuild indexes
- Increase maintenance_work_mem when you cluster tables

- Tune memory to prevent IO:BufFileRead and IO:BufFileWrite

Identify the problem

You can view temporary file usage directly in Performance Insights. For more information, see Viewing temporary file usage with Performance Insights. When Performance Insights is disabled, you might notice increased `IO:BufFileRead` and `IO:BufFileWrite` operations.

To identify the source of the problem, you can set the `log_temp_files` parameter to log all queries that generate more than your specified threshold KB of temporary files. By default, `log_temp_files` is set to `-1`, which turns off this logging feature. If you set this parameter to `0`, RDS for PostgreSQL logs all temporary files. If you set it to is `1024`, RDS for PostgreSQL logs all queries that produce temporary files larger than 1 MB. For more information about `log_temp_files`, see Error Reporting and Logging in the PostgreSQL documentation.

Examine your join queries

It's likely that your query uses joins. For example, the following query joins four tables.

```
SELECT *
       FROM "order"
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = order.customer_id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

A possible cause of spikes in temporary file usage is a problem in the query itself. For example, a broken clause might not filter the joins properly. Consider the second inner join in the following example.

```
SELECT *
       FROM "order"
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = customer.id)
```

```
INNER JOIN customer_address
      ON (customer_address.customer_id = customer.id AND
          order.customer_address_id = customer_address.id)
WHERE customer.id = 1234567890;
```

The preceding query mistakenly joins `customer.id` to `customer.id`, generating a Cartesian product between every customer and every order. This type of accidental join generates large temporary files. Depending on the size of the tables, a Cartesian query can even fill up storage. Your application might have Cartesian joins when the following conditions are met:

- You see large, sharp decreases in storage availability, followed by fast recovery.
- No indexes are being created.
- No `CREATE TABLE FROM SELECT` statements are being issued.
- No materialized views are being refreshed.

To see whether the tables are being joined using the proper keys, inspect your query and object-relational mapping directives. Bear in mind that certain queries of your application are not called all the time, and some queries are dynamically generated.

Examine your ORDER BY and GROUP BY queries

In some cases, an `ORDER BY` clause can result in excessive temporary files. Consider the following guidelines:

- Only include columns in an `ORDER BY` clause when they need to be ordered. This guideline is especially important for queries that return thousands of rows and specify many columns in the `ORDER BY` clause.
- Considering creating indexes to accelerate `ORDER BY` clauses when they match columns that have the same ascending or descending order. Partial indexes are preferable because they are smaller. Smaller indexes are read and traversed more quickly.
- If you create indexes for columns that can accept null values, consider whether you want the null values stored at the end or at the beginning of the indexes.

  If possible, reduce the number of rows that need to be ordered by filtering the result set. If you use `WITH` clause statements or subqueries, remember that an inner query generates a result set and passes it to the outside query. The more rows that a query can filter out, the less ordering the query needs to do.

- If you don't need to obtain the full result set, use the LIMIT clause. For example, if you only want the top five rows, a query using the LIMIT clause doesn't keep generating results. In this way, the query requires less memory and temporary files.

A query that uses a GROUP BY clause can also require temporary files. GROUP BY queries summarize values by using functions such as the following:

- COUNT
- AVG
- MIN
- MAX
- SUM
- STDDEV

To tune GROUP BY queries, follow the recommendations for ORDER BY queries.

Avoid using the DISTINCT operation

If possible, avoid using the DISTINCT operation to remove duplicated rows. The more unnecessary and duplicated rows that your query returns, the more expensive the DISTINCT operation becomes. If possible, add filters in the WHERE clause even if you use the same filters for different tables. Filtering the query and joining correctly improves your performance and reduces resource use. It also prevents incorrect reports and results.

If you need to use DISTINCT for multiple rows of a same table, consider creating a composite index. Grouping multiple columns in an index can improve the time to evaluate distinct rows. Also, if you use RDS for PostgreSQL version 10 or higher, you can correlate statistics among multiple columns by using the CREATE STATISTICS command.

Consider using window functions instead of GROUP BY functions

Using GROUP BY, you change the result set, and then retrieve the aggregated result. Using window functions, you aggregate data without changing the result set. A window function uses the OVER clause to perform calculations across the sets defined by the query, correlating one row with another. You can use all the GROUP BY functions in window functions, but also use functions such as the following:

- RANK
- ARRAY_AGG
- ROW_NUMBER
- LAG
- LEAD

To minimize the number of temporary files generated by a window function, remove duplications for the same result set when you need two distinct aggregations. Consider the following query.

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
     , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
  FROM empsalary;
```

You can rewrite the query with the `WINDOW` clause as follows.

```
SELECT sum(salary) OVER w as sum_salary
         , avg(salary) OVER w as_avg_salary
    FROM empsalary
  WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

By default, the RDS for PostgreSQL execution planner consolidates similar nodes so that it doesn't duplicate operations. However, by using an explicit declaration for the window block, you can maintain the query more easily. You might also improve performance by preventing duplication.

Investigate materialized views and CTAS statements

When a materialized view refreshes, it runs a query. This query can contain an operation such as `GROUP BY`, `ORDER BY`, or `DISTINCT`. During a refresh, you might observe large numbers of temporary files and the wait events `IO:BufFileWrite` and `IO:BufFileRead`. Similarly, when you create a table based on a `SELECT` statement, the `CREATE TABLE` statement runs a query. To reduce the temporary files needed, optimize the query.

Use pg_repack when you rebuild indexes

When you create an index, the engine orders the result set. As tables grow in size, and as values in the indexed column become more diverse, the temporary files require more space. In most cases, you can't prevent the creation of temporary files for large tables without modifying the maintenance work memory area. For more information about `maintenance_work_mem`, see

https://www.postgresql.org/docs/current/runtime-config-resource.html in the PostgreSQL documentation.

A possible workaround when recreating a large index is to use the pg_repack extension. For more information, see Reorganize tables in PostgreSQL databases with minimal locks in the pg_repack documentation. For information about setting up the extension in your RDS for PostgreSQL DB instance, see Reducing bloat in tables and indexes with the pg_repack extension.

Increase maintenance_work_mem when you cluster tables

The `CLUSTER` command clusters the table specified by *table_name* based on an existing index specified by *index_name*. RDS for PostgreSQL physically recreates the table to match the order of a given index.

When magnetic storage was prevalent, clustering was common because storage throughput was limited. Now that SSD-based storage is common, clustering is less popular. However, if you cluster tables, you can still increase performance slightly depending on the table size, index, query, and so on.

If you run the `CLUSTER` command and observe the wait events `IO:BufFileWrite` and `IO:BufFileRead`, tune `maintenance_work_mem`. Increase the memory size to a fairly large amount. A high value means that the engine can use more memory for the clustering operation.

Tune memory to prevent IO:BufFileRead and IO:BufFileWrite

In some situations, you need to tune memory. Your goal is to balance memory across the following areas of consumption using the appropriate parameters, as follows.

- The `work_mem` value
- The memory remaining after discounting the `shared_buffers` value
- The maximum connections opened and in use, which is limited by `max_connections`

For more information about tuning memory, see Resource Consumption in the PostgreSQL documentation.

Increase the size of the work memory area

In some situations, your only option is to increase the memory used by your session. If your queries are correctly written and are using the correct keys for joins, consider increasing the `work_mem` value.

To find out how many temporary files a query generates, set `log_temp_files` to `0`. If you increase the `work_mem` value to the maximum value identified in the logs, you prevent the query from generating temporary files. However, `work_mem` sets the maximum per plan node for each connection or parallel worker. If the database has 5,000 connections, and if each one uses 256 MiB memory, the engine needs 1.2 TiB of RAM. Thus, your instance might run out of memory.

Reserve sufficient memory for the shared buffer pool

Your database uses memory areas such as the shared buffer pool, not just the work memory area. Consider the requirements of these additional memory areas before you increase `work_mem`.

For example, assume that your RDS for PostgreSQL instance class is db.r5.2xlarge. This class has 64 GiB of memory. By default, 25 percent of the memory is reserved for the shared buffer pool. After you subtract the amount allocated to the shared memory area, 16,384 MB remains. Don't allocate the remaining memory exclusively to the work memory area because the operating system and the engine also require memory.

The memory that you can allocate to `work_mem` depends on the instance class. If you use a larger instance class, more memory is available. However, in the preceding example, you can't use more than 16 GiB. Otherwise, your instance becomes unavailable when it runs out of memory. To recover the instance from the unavailable state, the RDS for PostgreSQL automation services automatically restart.

Manage the number of connections

Suppose that your database instance has 5,000 simultaneous connections. Each connection uses at least 4 MiB of `work_mem`. The high memory consumption of the connections is likely to degrade performance. In response, you have the following options:

- Upgrade to a larger instance class.
- Decrease the number of simultaneous database connections by using a connection proxy or pooler.

For proxies, consider Amazon RDS Proxy, pgBouncer, or a connection pooler based on your application. This solution alleviates the CPU load. It also reduces the risk when all connections require the work memory area. When fewer database connections exist, you can increase the value of `work_mem`. In this way, you reduce the occurrence of the `IO:BufFileRead` and `IO:BufFileWrite` wait events. Also, the queries waiting for the work memory area speed up significantly.

IO:DataFileRead

The `IO:DataFileRead` event occurs when a connection waits on a backend process to read a required page from storage because the page isn't available in shared memory.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

All queries and data manipulation (DML) operations access pages in the buffer pool. Statements that can induce reads include `SELECT`, `UPDATE`, and `DELETE`. For example, an `UPDATE` can read pages from tables or indexes. If the page being requested or updated isn't in the shared buffer pool, this read can lead to the `IO:DataFileRead` event.

Because the shared buffer pool is finite, it can fill up. In this case, requests for pages that aren't in memory force the database to read blocks from disk. If the `IO:DataFileRead` event occurs frequently, your shared buffer pool might be too small to accommodate your workload. This problem is acute for `SELECT` queries that read a large number of rows that don't fit in the buffer pool. For more information about the buffer pool, see Resource Consumption in the PostgreSQL documentation.

Likely causes of increased waits

Common causes for the `IO:DataFileRead` event include the following:

Connection spikes

You might find multiple connections generating the same number of IO:DataFileRead wait events. In this case, a spike (sudden and large increase) in `IO:DataFileRead` events can occur.

SELECT and DML statements performing sequential scans

Your application might be performing a new operation. Or an existing operation might change because of a new execution plan. In such cases, look for tables (particularly large tables) that have a greater `seq_scan` value. Find them by querying `pg_stat_user_tables`. To track queries that are generating more read operations, use the extension `pg_stat_statements`.

CTAS and CREATE INDEX for large data sets

A *CTAS* is a `CREATE TABLE AS SELECT` statement. If you run a CTAS using a large data set as a source, or create an index on a large table, the `IO:DataFileRead` event can occur. When you create an index, the database might need to read the entire object using a sequential scan. A CTAS generates `IO:DataFile` reads when pages aren't in memory.

Multiple vacuum workers running at the same time

Vacuum workers can be triggered manually or automatically. We recommend adopting an aggressive vacuum strategy. However, when a table has many updated or deleted rows, the `IO:DataFileRead` waits increase. After space is reclaimed, the vacuum time spent on `IO:DataFileRead` decreases.

Ingesting large amounts of data

When your application ingests large amounts of data, `ANALYZE` operations might occur more often. The `ANALYZE` process can be triggered by an autovacuum launcher or invoked manually.

The `ANALYZE` operation reads a subset of the table. The number of pages that must be scanned is calculated by multiplying 30 by the `default_statistics_target` value. For more information, see the PostgreSQL documentation. The `default_statistics_target` parameter accepts values between 1 and 10,000, where the default is 100.

Resource starvation

If instance network bandwidth or CPU are consumed, the `IO:DataFileRead` event might occur more frequently.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- Check predicate filters for queries that generate waits

- Minimize the effect of maintenance operations
- Respond to high numbers of connections

Check predicate filters for queries that generate waits

Assume that you identify specific queries that are generating `IO:DataFileRead` wait events. You might identify them using the following techniques:

- Performance Insights
- Catalog views such as the one provided by the extension `pg_stat_statements`
- The catalog view `pg_stat_all_tables`, if it periodically shows an increased number of physical reads
- The `pg_statio_all_tables` view, if it shows that `_read` counters are increasing

We recommend that you determine which filters are used in the predicate (`WHERE` clause) of these queries. Follow these guidelines:

- Run the `EXPLAIN` command. In the output, identify which types of scans are used. A sequential scan doesn't necessarily indicate a problem. Queries that use sequential scans naturally produce more `IO:DataFileRead` events when compared to queries that use filters.

  Find out whether the column listed in the `WHERE` clause is indexed. If not, consider creating an index for this column. This approach avoids the sequential scans and reduces the `IO:DataFileRead` events. If a query has restrictive filters and still produces sequential scans, evaluate whether the proper indexes are being used.
- Find out whether the query is accessing a very large table. In some cases, partitioning a table can improve performance, allowing the query to only read necessary partitions.
- Examine the cardinality (total number of rows) from your join operations. Note how restrictive the values are that you're passing in the filters for your `WHERE` clause. If possible, tune your query to reduce the number of rows that are passed in each step of the plan.

Minimize the effect of maintenance operations

Maintenance operations such as `VACUUM` and `ANALYZE` are important. We recommend that you don't turn them off because you find `IO:DataFileRead` wait events related to these maintenance operations. The following approaches can minimize the effect of these operations:

- Run maintenance operations manually during off-peak hours. This technique prevents the database from reaching the threshold for automatic operations.
- For very large tables, consider partitioning the table. This technique reduces the overhead of maintenance operations. The database only accesses the partitions that require maintenance.
- When you ingest large amounts of data, consider disabling the autoanalyze feature.

The autovacuum feature is automatically triggered for a table when the following formula is true.

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples x autovacuum_vacuum_scale_factor)
 + autovacuum_vacuum_threshold
```

The view `pg_stat_user_tables` and catalog `pg_class` have multiple rows. One row can correspond to one row in your table. This formula assumes that the `reltuples` are for a specific table. The parameters `autovacuum_vacuum_scale_factor` (0.20 by default) and `autovacuum_vacuum_threshold` (50 tuples by default) are usually set globally for the whole instance. However, you can set different values for a specific table.

Topics

- Find tables consuming space unnecessarily
- Find indexes consuming space unnecessarily
- Find tables that are eligible to be autovacuumed

Find tables consuming space unnecessarily

To find tables consuming space unnecessarily, you can use functions from the PostgreSQL `pgstattuple` extension. This extension (module) is available by default on all RDS for PostgreSQL DB instances and can be instantiated on the instance with the following command.

```
CREATE EXTENSION pgstattuple;
```

For more information about this extension, see pgstattuple in the PostgreSQL documentation.

You can check for table and index bloat in your application. For more information, see Diagnosing table and index bloat.

Find indexes consuming space unnecessarily

To find bloated indexes and estimate the amount of space consumed unnecessarily on the tables for which you have read privileges, you can run the following query.

```
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
-- This query is compatible with PostgreSQL 8.2 and later.

SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_ratio,
  fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
  is_na
  -- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
  -- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
  -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
       -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
    ) AS est_pages_ff,
    bs, nspname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- , stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS
 pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth,
 reltuples
    -- (DEBUG INFO)
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, relam,
 table_oid, fillfactor,
      ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
          END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
 MAXALIGN
```

```
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
          END
      )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
      -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
    FROM (
      SELECT
        i.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid
AS table_oid,
        current_setting('block_size')::numeric AS bs, fillfactor,
        CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
          WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64'
THEN 8
          ELSE 4
        END AS maxalign,
        /* per page header, fixed size: 20 for 7.X, 24 for others */
        24 AS pagehdr,
        /* per page btree opaque data */
        16 AS pageopqdata,
        /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
        CASE WHEN max(coalesce(s.null_frac,0)) = 0
          THEN 2 -- IndexTupleData size
          ELSE 2 + (( 32 + 8 - 1 ) / 8)
          -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per
index + 8 - 1 /8)
        END AS index_tuple_hdr_bm,
        /* data len: we remove null values save space using it fractionnal part from
stats */
        sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
        max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0
AS is_na
      FROM pg_attribute AS a
        JOIN (
          SELECT nspname, tbl.relname AS tblname, idx.relname AS idxname,
            idx.reltuples, idx.relpages, idx.relam,
            indrelid, indexrelid, indkey::smallint[] AS attnum,
            coalesce(substring(
              array_to_string(idx.reloptions, ' ')
               from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor
          FROM pg_index
            JOIN pg_class idx ON idx.oid=pg_index.indexrelid
            JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
```

```
            JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
          WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
        ) AS i ON a.attrelid = i.indexrelid
        JOIN pg_stats AS s ON s.schemaname = i.nspname
          AND ((s.tablename = i.tblname AND s.attname =
 pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
          -- stats from tbl
          OR  (s.tablename = i.idxname AND s.attname = a.attname))
          -- stats from functional cols
        JOIN pg_type AS t ON a.atttypid = t.oid
      WHERE a.attnum > 0
      GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
  ) AS s2
    JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;
```

Find tables that are eligible to be autovacuumed

To find tables that are eligible to be autovacuumed, run the following query.

```
--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH  vbt AS (SELECT setting AS autovacuum_vacuum_threshold
              FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
              FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age
              FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
    , sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
                split_part(setting, '=', 2) as value
              FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    '"'||ns.nspname||'"."'||c.relname||'"' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
         coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
 c.reltuples)
```

```
         as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
JOIN fma on (1=1)
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid =
 cvbt.opt_oid
LEFT JOIN sto cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
 cvsf.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
    or
    coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
      coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples
 <= n_dead_tup
    -- or 1 = 1
)
ORDER BY age(relfrozenxid) DESC;
```

Respond to high numbers of connections

When you monitor Amazon CloudWatch, you might find that the `DatabaseConnections` metric spikes. This increase indicates an increased number of connections to your database. We recommend the following approach:

- Limit the number of connections that the application can open with each instance. If your application has an embedded connection pool feature, set a reasonable number of connections. Base the number on what the vCPUs in your instance can parallelize effectively.

  If your application doesn't use a connection pool feature, considering using Amazon RDS Proxy or an alternative. This approach lets your application open multiple connections with the load balancer. The balancer can then open a restricted number of connections with the database. As fewer connections are running in parallel, your DB instance performs less context switching in the kernel. Queries should progress faster, leading to fewer wait events. For more information, see Amazon RDS Proxy.

- Whenever possible, take advantage of read replicas for RDS for PostgreSQL. When your application runs a read-only operation, send these requests to the read replica(s). This technique reduces the I/O pressure on the primary (writer) node.
- Consider scaling up your DB instance. A higher-capacity instance class gives more memory, which gives RDS for PostgreSQL a larger shared buffer pool to hold pages. The larger size also gives the DB instance more vCPUs to handle connections. More vCPUs are particularly helpful when the operations that are generating `IO:DataFileRead` wait events are writes.

IO:WALWrite

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL 10 and higher.

Context

Activity in the database that's generating write-ahead log data fills up the WAL buffers first and then writes to disk, asynchronously. The wait event `IO:WALWrite` is generated when the SQL session is waiting for the WAL data to complete writing to disk so that it can release the transaction's COMMIT call.

Likely causes of increased waits

If this wait event occurs often, you should review your workload and the type of updates that your workload performs and their frequency. In particular, look for the following types of activity.

Heavy DML activity

Changing data in database tables doesn't happen instantaneously. An insert to one table might need to wait for an insert or an update to the same table from another client. The data

manipulation language (DML) statements for changing data values (INSERT, UPDATE, DELETE, COMMIT, ROLLBACK TRANSACTION) can result in contention that causes the write-ahead logfile to be waiting for the buffers to be flushed. This situation is captured in the following Amazon RDS Performance Insights metrics that indicate heavy DML activity.

- `tup_inserted`
- `tup_updated`
- `tup_deleted`
- `xact_rollback`
- `xact_commit`

For more information about these metrics, see Performance Insights counters for Amazon RDS for PostgreSQL.

Frequent checkpoint activity

Frequent checkpoints contribute to a higher number of WAL files. In RDS for PostgreSQL, full page writes are always "on." Full page writes help protect against data loss. However, when checkpointing occurs too frequently, the system can suffer overall performance issues. This is especially true on systems with heavy DML activity. In some cases, you might find error messages in your `postgresql.log` stating that “checkpoints are occurring too frequently."

We recommend that when tuning checkpoints, you carefully balance performance against expected time need to recover in the event of an abnormal shutdown.

Actions

We recommend the following actions to reduce the numbers of this wait event.

Topics

- Reduce the number of commits
- Monitor your checkpoints
- Scale up IO
- Dedicated log volume (DLV)

Reduce the number of commits

To reduce the number of commits, you can combine statements into transaction blocks. Use Amazon RDS Performance Insights to examine the type of queries being run. You can also move large maintenance operations to off-peak hours. For example, create indexes or use `pg_repack` operations during non-production hours.

Monitor your checkpoints

There are two parameters that you can monitor to see how frequently your RDS for PostgreSQL DB instance is writing to the WAL file for checkpoints.

- `log_checkpoints` – This parameter is set to "on" by default. It causes a message to get sent to the PostgreSQL log for each checkpoint. These log messages include the number of buffers written, the time spent writing them, and the number of WAL files added, removed, or recycled for the given checkpoint.

  For more information about this parameter, see Error Reporting and Logging in the PostgreSQL documentation.
- `checkpoint_warning` – This parameter sets a threshold value (in seconds) for checkpoint frequency above which a warning is generated. By default, this parameter isn't set in RDS for PostgreSQL. You can set the value of this parameter to get a warning when the database changes in your RDS for PostgreSQL DB instance are written at a rate for which the WAL files are not sized to handle. For example, say you set this parameter to 30. If your RDS for PostgreSQL instance needs to write changes more often than every 30 seconds, the warning that "checkpoints are occurring too frequently" is sent to the PostgreSQL log. This can indicate that your `max_wal_size` value should be increased.

  For more information, see Write Ahead Log in the PostgreSQL documentation.

Scale up IO

This type of input/output (IO) wait event can remediated by scaling the input/output operations per second (IOPs) to provide faster IO. Scaling IO is preferable to scaling CPU, because scaling CPU can result in even more IO contention because the increased CPU can handle more work and thus make the IO bottleneck even worse. In general, we recommend that you consider tuning your workload before performing scaling operations.

Dedicated log volume (DLV)

You can use a dedicated log volume (DLV) for a DB instance that uses Provisioned IOPS (PIOPS) storage by using the Amazon RDS console, AWS CLI, or Amazon RDS API. A DLV moves PostgreSQL database transaction logs to a storage volume that's separate from the volume containing the database tables. For more information, see Dedicated log volume (DLV).

IPC:parallel wait events

The following `IPC:parallel wait events` indicate that a session is waiting for inter-process communication related to parallel query execution operations.

- `IPC:BgWorkerStartup` - A process is waiting for a parallel worker process to complete its startup sequence. This happens when initializing workers for parallel query execution.
- `IPC:BgWorkerShutdown` - A process is waiting for a parallel worker process to complete its shutdown sequence. This occurs during the cleanup phase of parallel query execution.
- `IPC:ExecuteGather` - A process is waiting to receive data from parallel worker processes during query execution. This occurs when the leader process needs to gather results from its workers.
- `IPC:ParallelFinish` - A process is waiting for parallel workers to finish their execution and report their final results. This happens during the completion phase of parallel query execution.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

Parallel query execution in PostgreSQL involves multiple processes working together to process a single query. When a query is determined to be suitable for parallelization,

a leader process coordinates with one or more parallel worker processes based on the `max_parallel_workers_per_gather` parameter setting. The leader process divides the work among workers, each worker processes its portion of data independently, and results are gathered back to the leader process.

> **Note**
>
> Each parallel worker operates as a separate process with resource requirements similar to a full user session. This means a parallel query with 4 workers can consume up to 5 times the resources (CPU, memory, I/O bandwidth) compared to a non-parallel query, as both the leader process and each worker process maintain their own resource allocations. For instance, settings like `work_mem` are applied individually to each worker, potentially multiplying the total memory usage across all processes.

The parallel query architecture consists of three main components:

- Leader process: The main process that initiates the parallel operation, divides the workload, and coordinates with worker processes.
- Worker processes: Background processes that execute portions of the query in parallel.
- Gather/Gather merge: Operations that combine results from multiple worker processes back to the leader

During parallel execution, processes need to communicate with each other through Inter-Process Communication (IPC) mechanisms. These IPC wait events occur during different phases:

- Worker startup: When parallel workers are being initialized
- Data exchange: When workers are processing data and sending results to the leader
- Worker shutdown: When parallel execution completes and workers are being terminated
- Synchronization points: When processes need to coordinate or wait for other processes to complete their tasks

Understanding these wait events is crucial for diagnosing performance issues related to parallel query execution, especially in high-concurrency environments where multiple parallel queries may be executing simultaneously.

Likely causes of increased waits

Several factors can contribute to an increase in parallel-related IPC wait events:

High concurrency of parallel queries

When many parallel queries are running simultaneously, it can lead to resource contention and increased waiting times for IPC operations. This is particularly common in systems with high transaction volumes or analytical workloads.

Suboptimal parallel query plans

If the query planner chooses inefficient parallel plans, it may result in unnecessary parallelization or poor work distribution among workers. This can lead to increased IPC waits, especially for `IPC:ExecuteGather` and `IPC:ParallelFinish` events. These planning issues often stem from outdated statistics and table/index bloat.

Frequent startup and shutdown of parallel workers

Short-lived queries that frequently initiate and terminate parallel workers can cause an increase in `IPC:BgWorkerStartup` and `IPC:BgWorkerShutdown` events. This is often seen in OLTP workloads with many small, parallelizable queries.

Resource constraints

Limited CPU, memory, or I/O capacity can cause bottlenecks in parallel execution, leading to increased wait times across all IPC events. For example, if CPU is saturated, worker processes may take longer to start up or process their portion of work.

Complex query structures

Queries with multiple levels of parallelism (e.g., parallel joins followed by parallel aggregations) can lead to more complex IPC patterns and potentially increased wait times, especially for `IPC:ExecuteGather` events.

Large result sets

Queries that produce large result sets may cause increased `IPC:ExecuteGather` wait times as the leader process spends more time collecting and processing results from worker processes.

Understanding these factors can help in diagnosing and addressing performance issues related to parallel query execution in Aurora PostgreSQL.

Actions

When you see waits related to parallel query, it typically means that a backend process is coordinating or waiting on parallel worker processes. These waits are common during execution of parallel plans. You can investigate and mitigate the impact of these waits by monitoring parallel worker usage, reviewing the parameter settings, and tuning query execution and resource allocation.

Topics

- Analyze query plans for inefficient parallelism
- Monitor parallel query usage
- Review and adjust parallel query settings
- Optimize resource allocation
- Investigate connection management
- Review and optimize maintenance operations

Analyze query plans for inefficient parallelism

Parallel query execution can often lead to system instability, CPU spikes, and unpredictable query performance variance. It's crucial to thoroughly analyze whether parallelism actually improves your specific workload. Use EXPLAIN ANALYZE to review parallel query execution plans.

Temporarily disable parallelism at the session level to compare plan efficiency:

```
SET max_parallel_workers_per_gather = 0;
EXPLAIN ANALYZE <your_query>;
```

Re-enable parallelism and compare:

```
RESET max_parallel_workers_per_gather;
EXPLAIN ANALYZE <your_query>;
```

If disabling parallelism yields better or more consistent results, consider disabling it for specific queries at the session level using SET commands. For a broader impact, you might want to disable

parallelism at the instance level by adjusting the relevant parameters in your DB parameter group. For more information, see Modifying parameters in a DB parameter group in Amazon RDS.

Monitor parallel query usage

Use the following queries to gain visibility into parallel query activity and capacity:

Check active parallel worker processes:

```
SELECT
    COUNT(*)
FROM
    pg_stat_activity
WHERE
    backend_type = 'parallel worker';
```

This query shows the number of active parallel worker processes. A high value may indicate that your `max_parallel_workers` is configured with a high value and you might want to consider reducing it.

Check concurrent parallel queries:

```
SELECT
    COUNT(DISTINCT leader_pid)
FROM
    pg_stat_activity
WHERE
    leader_pid IS NOT NULL;
```

This query returns the number of distinct leader processes that have launched parallel queries. A high number here indicates that multiple sessions are running parallel queries concurrently, which can increase demand on CPU and memory.

Review and adjust parallel query settings

Review the following parameters to ensure they align with your workload:

- `max_parallel_workers`: Total number of parallel workers across all sessions.

- `max_parallel_workers_per_gather`: Max workers per query.

For OLAP workloads, increasing these values can improve performance. For OLTP workloads, lower values are generally preferred.

```
SHOW max_parallel_workers;
SHOW max_parallel_workers_per_gather;
```

Optimize resource allocation

Monitor CPU utilization and consider adjusting the number of vCPUs if consistently high and if your application benefits from parallel queries. Ensure adequate memory is available for parallel operations.

- Use Performance Insights metrics to determine if the system is CPU-bound.
- Each parallel worker uses its own `work_mem`. Ensure total memory usage is within instance limits.

The parallel queries may consume very substantially more resources than non-parallel queries, because each worker process is a completely separate process which has roughly the same impact on the system as an additional user session. This should be taken into account when choosing a value for this setting, as well as when configuring other settings that control resource utilization, such as `work_mem`. For more information, see the PostgreSQL documentation. Resource limits such as `work_mem` are applied individually to each worker, which means the total utilization may be much higher across all processes than it would normally be for any single process.

Consider increasing vCPUs or tuning memory parameters if your workload is heavily parallelized.

Investigate connection management

If experiencing connection exhaustion, review application connection pooling strategies. Consider implementing connection pooling at the application level if not already in use.

Review and optimize maintenance operations

Coordinate index creation and other maintenance tasks to prevent resource contention. Consider scheduling these operations during off-peak hours. Avoid scheduling heavy maintenance (e.g.,

parallel index builds) during periods of high user query load. These operations can consume parallel workers and impact performance for regular queries.

IPC:ProcArrayGroupUpdate

The `IPC:ProcArrayGroupUpdate` event occurs when a session is waiting for the group leader to update the transaction status at the end of that operation. While PostgreSQL generally associates IPC type wait events with parallel query operations, this particular wait event is not specific to parallel queries.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

Understanding the process array – The process (proc) array is a shared memory structure in PostgreSQL. It holds information about all running processes, including transaction details. During transaction completion (`COMMIT` or `ROLLBACK`), the ProcArray needs to be updated to reflect the change and clear the transactionID from the array. The session attempting to finish its transaction must acquire an exclusive lock on the ProcArray. This prevents other processes from obtaining shared or exclusive locks on it.

Group update mechanism – While performing a COMMIT or ROLLBACK, if a backend process cannot obtain a ProcArrayLock in exclusive mode, it updates a special field called ProcArrayGroupMember. This adds the transaction to the list of sessions that intend to end. This backend process then sleeps and the time it sleeps is instrumented as the ProcArrayGroupUpdate wait event. The first process in the ProcArray with procArrayGroupMember, referred to as the leader process, acquires the ProcArrayLock in exclusive mode. It then clears the list of processes waiting for group transactionID clearing. Once this completes, the leader releases the ProcArrayLock and then wakes up all processes in this list, notifying them that their transaction is completed.

Likely causes of increased waits

The more processes that are running, the longer a leader will hold on to a procArrayLock in exclusive mode. Consequently, the more write transactions end up in a group update scenario causing a potential pile up of processes waiting on the `ProcArrayGroupUpdate` wait event. In Database Insights' Top SQL view, you will see that COMMIT is the statement with the majority of this wait event. This is expected but will require deeper investigation into the specific write SQL being run to determine what appropriate action to take.

Actions

We recommend different actions depending on the causes of your wait event. Identify `IPC:ProcArrayGroupUpdate` events by using Amazon RDS Performance Insights or by querying the PostgreSQL system view `pg_stat_activity`.

Topics

- Monitoring transaction commit and rollback operations
- Reducing concurrency
- Implementing connection pooling
- Using faster storage

Monitoring transaction commit and rollback operations

Monitor commits and rollbacks – An increased number of commits and rollbacks can lead to increased pressure on the ProcArray. For example, if a SQL statement begins to fail due to increased duplicate key violations, you may see an increase in rollbacks which can increase ProcArray contention and table bloat.

Amazon RDS Database Insights provides the PostgreSQL metrics `xact_commit` and `xact_rollback` to report the number of commits and rollbacks per second.

Reducing concurrency

Batching transactions – Where possible, batch operations in single transactions to reduce commit/rollback operations.

Limit concurrency – Reduce the number of concurrently active transactions to alleviate lock contention on the ProcArray. While it will require some testing, reducing the total number of concurrent connections can reduce contention and maintain throughput.

Implementing connection pooling

Connection pooling solutions – Use connection pooling to manage database connections efficiently, reducing the total number of backends and therefore the workload on the ProcArray. While it will require some testing, reducing the total number of concurrent connections can reduce contention and maintain throughput.

Reduce connection storms – Similarly, a pattern of frequently creating and terminating connections causes additional pressure on the ProcArray. By reducing this pattern, overall contention is reduced.

Using faster storage

Dedicated log volume – If the `IPC:ProcArrayGroupUpdate` wait event is accompanied with high `IO:WALWrite` wait events, setting up a dedicated log volume can reduce the bottleneck writing to WAL. In turn, this improves the performance of commits.

For more information, see Dedicated log volume.

Lock:advisory

The `Lock:advisory` event occurs when a PostgreSQL application uses a lock to coordinate activity across multiple sessions.

Topics

- Relevant engine versions
- Context
- Causes
- Actions

Relevant engine versions

This wait event information is relevant for RDS for PostgreSQL versions 9.6 and higher.

Context

PostgreSQL advisory locks are application-level, cooperative locks explicitly locked and unlocked by the user's application code. An application can use PostgreSQL advisory locks to coordinate activity across multiple sessions. Unlike regular, object- or row-level locks, the application has full

control over the lifetime of the lock. For more information, see Advisory Locks in the PostgreSQL documentation.

Advisory locks can be released before a transaction ends or be held by a session across transactions. This isn't true for implicit, system-enforced locks, such as an access-exclusive lock on a table acquired by a `CREATE INDEX` statement.

For a description of the functions used to acquire (lock) and release (unlock) advisory locks, see Advisory Lock Functions in the PostgreSQL documentation.

Advisory locks are implemented on top of the regular PostgreSQL locking system and are visible in the `pg_locks` system view.

Causes

This lock type is exclusively controlled by an application explicitly using it. Advisory locks that are acquired for each row as part of a query can cause a spike in locks or a long-term buildup.

These effects happen when the query is run in a way that acquires locks on more rows than are returned by the query. The application must eventually release every lock, but if locks are acquired on rows that aren't returned, the application can't find all of the locks.

The following example is from Advisory Locks in the PostgreSQL documentation.

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

In this example, the `LIMIT` clause can only stop the query's output after the rows have already been internally selected and their ID values locked. This can happen suddenly when a growing data volume causes the planner to choose a different execution plan that wasn't tested during development. The buildup in this case happens because the application explicitly calls `pg_advisory_unlock` for every ID value that was locked. However, in this case it can't find the set of locks acquired on rows that weren't returned. Because the locks are acquired on the session level, they aren't released automatically at the end of the transaction.

Another possible cause for spikes in blocked lock attempts is unintended conflicts. In these conflicts, unrelated parts of the application share the same lock ID space by mistake.

Actions

Review application usage of advisory locks and detail where and when in the application flow each type of advisory lock is acquired and released.

Determine whether a session is acquiring too many locks or a long-running session isn't releasing locks early enough, leading to a slow buildup of locks. You can correct a slow buildup of session-level locks by ending the session using `pg_terminate_backend(pid)`.

A client waiting for an advisory lock appears in `pg_stat_activity` with `wait_event_type=Lock` and `wait_event=advisory`. You can obtain specific lock values by querying the `pg_locks` system view for the same `pid`, looking for `locktype=advisory` and `granted=f`.

You can then identify the blocking session by querying `pg_locks` for the same advisory lock having `granted=t`, as shown in the following example.

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
         blocking_activity.usename AS blocking_user,
         now() - blocked_activity.xact_start AS blocked_transaction_duration,
         now() - blocking_activity.xact_start AS blocking_transaction_duration,
         concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
 blocked_wait_event,
         concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
 blocking_wait_event,
         blocked_activity.state AS blocked_state,
         blocking_activity.state AS blocking_state,
         blocked_locks.locktype AS blocked_locktype,
         blocking_locks.locktype AS blocking_locktype,
         blocked_activity.query AS blocked_statement,
         blocking_activity.query AS blocking_statement
    FROM pg_catalog.pg_locks blocked_locks
    JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
 blocked_locks.pid
    JOIN pg_catalog.pg_locks blocking_locks
        ON blocking_locks.locktype = blocked_locks.locktype
        AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
        AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
        AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
        AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
        AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
        AND blocking_locks.transactionid IS NOT DISTINCT FROM
 blocked_locks.transactionid
        AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
        AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
        AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
```

```
        AND blocking_locks.pid != blocked_locks.pid
    JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
    WHERE NOT blocked_locks.GRANTED;
```

All of the advisory lock API functions have two sets of arguments, either one `bigint` argument or two `integer` arguments:

- For the API functions with one `bigint` argument, the upper 32 bits are in `pg_locks.classid` and the lower 32 bits are in `pg_locks.objid`.
- For the API functions with two `integer` arguments, the first argument is `pg_locks.classid` and the second argument is `pg_locks.objid`.

The `pg_locks.objsubid` value indicates which API form was used: 1 means one `bigint` argument; 2 means two `integer` arguments.

Lock:extend

The `Lock:extend` event occurs when a backend process is waiting to lock a relation to extend it while another process has a lock on that relation for the same purpose.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

The event `Lock:extend` indicates that a backend process is waiting to extend a relation that another backend process holds a lock on while it's extending that relation. Because only one process at a time can extend a relation, the system generates a `Lock:extend` wait event. `INSERT`, `COPY`, and `UPDATE` operations can generate this event.

Likely causes of increased waits

When the `Lock:extend` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

Surge in concurrent inserts or updates to the same table

There might be an increase in the number of concurrent sessions with queries that insert into or update the same table.

Insufficient network bandwidth

The network bandwidth on the DB instance might be insufficient for the storage communication needs of the current workload. This can contribute to storage latency that causes an increase in `Lock:extend` events.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- Reduce concurrent inserts and updates to the same relation
- Increase network bandwidth

Reduce concurrent inserts and updates to the same relation

First, determine whether there's an increase in `tup_inserted` and `tup_updated` metrics and an accompanying increase in this wait event. If so, check which relations are in high contention for insert and update operations. To determine this, query the `pg_stat_all_tables` view for the values in `n_tup_ins` and `n_tup_upd` fields. For information about the `pg_stat_all_tables` view, see pg_stat_all_tables in the PostgreSQL documentation.

To get more information about blocking and blocked queries, query `pg_stat_activity` as in the following example:

```
SELECT
    blocked.pid,
    blocked.usename,
    blocked.query,
    blocking.pid AS blocking_id,
```

```
    blocking.query AS blocking_query,
    blocking.wait_event AS blocking_wait_event,
    blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

   pid  | usename  |            query             | blocking_id |
            blocking_query                          | blocking_wait_event |
 blocking_wait_event_type
  ------+----------+------------------------------+-------------
+-------------------------------------------------------------------
+---------------------+--------------------------
   7143 |  myuser  | insert into tab1 values (1); |        4600 | INSERT INTO tab1 (a)
 SELECT s FROM generate_series(1,1000000) s; | DataFileExtend      | IO
```

After you identify relations that contribute to increase `Lock:extend` events, use the following techniques to reduce the contention:

- Find out whether you can use partitioning to reduce contention for the same table. Separating inserted or updated tuples into different partitions can reduce contention. For information about partitioning, see Managing PostgreSQL partitions with the pg_partman extension.
- If the wait event is mainly due to update activity, consider reducing the relation's fillfactor value. This can reduce requests for new blocks during the update. The fillfactor is a storage parameter for a table that determines the maximum amount of space for packing a table page. It's expressed as a percentage of the total space for a page. For more information about the fillfactor parameter, see CREATE TABLE in the PostgreSQL documentation.

> **Important**
>
> We highly recommend that you test your system if you change the fillfactor because changing this value can negatively impact performance, depending on your workload.

Increase network bandwidth

To see whether there's an increase in write latency, check the `WriteLatency` metric in CloudWatch. If there is, use the `WriteThroughput` and `ReadThroughput` Amazon CloudWatch

metrics to monitor the storage related traffic on the DB instance. These metrics can help you to determine if network bandwidth is sufficient for the storage activity of your workload.

If your network bandwidth isn't enough, increase it. If your DB instance is reaching the network bandwidth limits, the only way to increase the bandwidth is to increase your DB instance size.

For more information about CloudWatch metrics, see Amazon CloudWatch instance-level metrics for Amazon RDS. For information about network performance for each DB instance class, see Amazon CloudWatch instance-level metrics for Amazon RDS.

Lock:Relation

The `Lock:Relation` event occurs when a query is waiting to acquire a lock on a table or view (relation) that's currently locked by another transaction.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

Most PostgreSQL commands implicitly use locks to control concurrent access to data in tables. You can also use these locks explicitly in your application code with the `LOCK` command. Many lock modes aren't compatible with each other, and they can block transactions when they're trying to access the same object. When this happens, RDS for PostgreSQL generates a `Lock:Relation` event. Some common examples are the following:

- Exclusive locks such as `ACCESS EXCLUSIVE` can block all concurrent access. Data definition language (DDL) operations such as `DROP TABLE`, `TRUNCATE`, `VACUUM FULL`, and `CLUSTER` acquire `ACCESS EXCLUSIVE` locks implicitly. `ACCESS EXCLUSIVE` is also the default lock mode for `LOCK TABLE` statements that don't specify a mode explicitly.

- Using `CREATE INDEX (without CONCURRENT)` on a table conflicts with data manipulation language (DML) statements `UPDATE`, `DELETE`, and `INSERT`, which acquire `ROW EXCLUSIVE` locks.

For more information about table-level locks and conflicting lock modes, see Explicit Locking in the PostgreSQL documentation.

Blocking queries and transactions typically unblock in one of the following ways:

- Blocking query – The application can cancel the query or the user can end the process. The engine can also force the query to end because of a session's statement-timeout or a deadlock detection mechanism.
- Blocking transaction – A transaction stops blocking when it runs a `ROLLBACK` or `COMMIT` statement. Rollbacks also happen automatically when sessions are disconnected by a client or by network issues, or are ended. Sessions can be ended when the database engine is shut down, when the system is out of memory, and so forth.

Likely causes of increased waits

When the `Lock:Relation` event occurs more frequently than normal, it can indicate a performance issue. Typical causes include the following:

Increased concurrent sessions with conflicting table locks

There might be an increase in the number of concurrent sessions with queries that lock the same table with conflicting locking modes.

Maintenance operations

Health maintenance operations such as `VACUUM` and `ANALYZE` can significantly increase the number of conflicting locks. `VACUUM FULL` acquires an `ACCESS EXCLUSIVE` lock, and `ANALYSE` acquires a `SHARE UPDATE EXCLUSIVE` lock. Both types of locks can cause a `Lock:Relation` wait event. Application data maintenance operations such as refreshing a materialized view can also increase blocked queries and transactions.

Locks on reader instances

There might be a conflict between the relation locks held by the writer and readers. Currently, only `ACCESS EXCLUSIVE` relation locks are replicated to reader instances. However, the

ACCESS EXCLUSIVE relation lock will conflict with any ACCESS SHARE relation locks held by the reader. This can cause an increase in lock relation wait events on the reader.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- Reduce the impact of blocking SQL statements
- Minimize the effect of maintenance operations

Reduce the impact of blocking SQL statements

To reduce the impact of blocking SQL statements, modify your application code where possible. Following are two common techniques for reducing blocks:

- Use the NOWAIT option – Some SQL commands, such as SELECT and LOCK statements, support this option. The NOWAIT directive cancels the lock-requesting query if the lock can't be acquired immediately. This technique can help prevent a blocking session from causing a pile-up of blocked sessions behind it.

  For example: Assume that transaction A is waiting on a lock held by transaction B. Now, if B requests a lock on a table that's locked by transaction C, transaction A might be blocked until transaction C completes. But if transaction B uses a NOWAIT when it requests the lock on C, it can fail fast and ensure that transaction A doesn't have to wait indefinitely.
- Use SET lock_timeout – Set a lock_timeout value to limit the time a SQL statement waits to acquire a lock on a relation. If the lock isn't acquired within the timeout specified, the transaction requesting the lock is cancelled. Set this value at the session level.

Minimize the effect of maintenance operations

Maintenance operations such as VACUUM and ANALYZE are important. We recommend that you don't turn them off because you find Lock:Relation wait events related to these maintenance operations. The following approaches can minimize the effect of these operations:

- Run maintenance operations manually during off-peak hours.

- To reduce `Lock:Relation` waits caused by autovacuum tasks, perform any needed autovacuum tuning. For information about tuning autovacuum, see Working with PostgreSQL autovacuum on Amazon RDS in the *Amazon RDS User Guide*.

Lock:transactionid

The `Lock:transactionid` event occurs when a transaction is waiting for a row-level lock.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

The event `Lock:transactionid` occurs when a transaction is trying to acquire a row-level lock that has already been granted to a transaction that is running at the same time. The session that shows the `Lock:transactionid` wait event is blocked because of this lock. After the blocking transaction ends in either a `COMMIT` or `ROLLBACK` statement, the blocked transaction can proceed.

The multiversion concurrency control semantics of RDS for PostgreSQL guarantee that readers don't block writers and writers don't block readers. For row-level conflicts to occur, blocking and blocked transactions must issue conflicting statements of the following types:

- `UPDATE`
- `SELECT … FOR UPDATE`
- `SELECT … FOR KEY SHARE`

The statement `SELECT … FOR KEY SHARE` is a special case. The database uses the clause `FOR KEY SHARE` to optimize the performance of referential integrity. A row-level lock on a row can block `INSERT`, `UPDATE`, and `DELETE` commands on other tables that reference the row.

Likely causes of increased waits

When this event appears more than normal, the cause is typically `UPDATE`, `SELECT ... FOR UPDATE`, or `SELECT ... FOR KEY SHARE` statements combined with the following conditions.

Topics

- High concurrency
- Idle in transaction
- Long-running transactions

High concurrency

RDS for PostgreSQL can use granular row-level locking semantics. The probability of row-level conflicts increases when the following conditions are met:

- A highly concurrent workload contends for the same rows.
- Concurrency increases.

Idle in transaction

Sometimes the `pg_stat_activity.state` column shows the value `idle in transaction`. This value appears for sessions that have started a transaction, but haven't yet issued a `COMMIT` or `ROLLBACK`. If the `pg_stat_activity.state` value isn't `active`, the query shown in `pg_stat_activity` is the most recent one to finish running. The blocking session isn't actively processing a query because an open transaction is holding a lock.

If an idle transaction acquired a row-level lock, it might be preventing other sessions from acquiring it. This condition leads to frequent occurrence of the wait event `Lock:transactionid`. To diagnose the issue, examine the output from `pg_stat_activity` and `pg_locks`.

Long-running transactions

Transactions that run for a long time get locks for a long time. These long-held locks can block other transactions from running.

Actions

Row-locking is a conflict among `UPDATE`, `SELECT ... FOR UPDATE`, or `SELECT ... FOR KEY SHARE` statements. Before attempting a solution, find out when these statements are running on the same row. Use this information to choose a strategy described in the following sections.

Topics

- Respond to high concurrency
- Respond to idle transactions
- Respond to long-running transactions

Respond to high concurrency

If concurrency is the issue, try one of the following techniques:

- Lower the concurrency in the application. For example, decrease the number of active sessions.
- Implement a connection pool. To learn how to pool connections with RDS Proxy, see Amazon RDS Proxy.
- Design the application or data model to avoid contending `UPDATE` and `SELECT ... FOR UPDATE` statements. You can also decrease the number of foreign keys accessed by `SELECT ... FOR KEY SHARE` statements.

Respond to idle transactions

If `pg_stat_activity.state` shows `idle in transaction`, use the following strategies:

- Turn on autocommit wherever possible. This approach prevents transactions from blocking other transactions while waiting for a `COMMIT` or `ROLLBACK`.
- Search for code paths that are missing `COMMIT`, `ROLLBACK`, or `END`.
- Make sure that the exception handling logic in your application always has a path to a valid `end of transaction`.
- Make sure that your application processes query results after ending the transaction with `COMMIT` or `ROLLBACK`.

Respond to long-running transactions

If long-running transactions are causing the frequent occurrence of `Lock:transactionid`, try the following strategies:

- Keep row locks out of long-running transactions.
- Limit the length of queries by implementing autocommit whenever possible.

Lock:tuple

The `Lock:tuple` event occurs when a backend process is waiting to acquire a lock on a tuple.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

The event `Lock:tuple` indicates that a backend is waiting to acquire a lock on a tuple while another backend holds a conflicting lock on the same tuple. The following table illustrates a scenario in which sessions generate the `Lock:tuple` event.

Time	Session 1	Session 2	Session 3
t1	Starts a transaction.		
t2	Updates row 1.		
t3		Updates row 1. The session acquires an exclusive lock on the tuple and then waits for	

Time	Session 1	Session 2	Session 3
		session 1 to release the lock by committing or rolling back.	
t4			Updates row 1. The session waits for session 2 to release the exclusive lock on the tuple.

Or you can simulate this wait event by using the benchmarking tool pgbench. Configure a high number of concurrent sessions to update the same row in a table with a custom SQL file.

To learn more about conflicting lock modes, see Explicit Locking in the PostgreSQL documentation. To learn more about pgbench, see pgbench in the PostgreSQL documentation.

Likely causes of increased waits

When this event appears more than normal, possibly indicating a performance problem, typical causes include the following:

- A high number of concurrent sessions are trying to acquire a conflicting lock for the same tuple by running `UPDATE` or `DELETE` statements.
- Highly concurrent sessions are running a `SELECT` statement using the `FOR UPDATE` or `FOR NO KEY UPDATE` lock modes.
- Various factors drive application or connection pools to open more sessions to execute the same operations. As new sessions are trying to modify the same rows, DB load can spike, and `Lock:tuple` can appear.

For more information, see Row-Level Locks in the PostgreSQL documentation.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- Investigate your application logic

- Find the blocker session
- Reduce concurrency when it is high
- Troubleshoot bottlenecks

Investigate your application logic

Find out whether a blocker session has been in the `idle in transaction` state for long time. If so, consider ending the blocker session as a short-term solution. You can use the `pg_terminate_backend` function. For more information about this function, see Server Signaling Functions in the PostgreSQL documentation.

For a long-term solution, do the following:

- Adjust the application logic.
- Use the `idle_in_transaction_session_timeout` parameter. This parameter ends any session with an open transaction that has been idle for longer than the specified amount of time. For more information, see Client Connection Defaults in the PostgreSQL documentation.
- Use autocommit as much as possible. For more information, see SET AUTOCOMMIT in the PostgreSQL documentation.

Find the blocker session

While the `Lock:tuple` wait event is occurring, identify the blocker and blocked session by finding out which locks depend on one another. For more information, see Lock dependency information in the PostgreSQL wiki.

The following example queries all sessions, filtering on `tuple` and ordering by `wait_time`.

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
         blocking_activity.usename AS blocking_user,
         now() - blocked_activity.xact_start AS blocked_transaction_duration,
         now() - blocking_activity.xact_start AS blocking_transaction_duration,
         concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
 blocked_wait_event,
         concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
 blocking_wait_event,
```

```
         blocked_activity.state AS blocked_state,
         blocking_activity.state AS blocking_state,
         blocked_locks.locktype AS blocked_locktype,
         blocking_locks.locktype AS blocking_locktype,
         blocked_activity.query AS blocked_statement,
         blocking_activity.query AS blocking_statement
   FROM pg_catalog.pg_locks blocked_locks
   JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
blocked_locks.pid
   JOIN pg_catalog.pg_locks blocking_locks
       ON blocking_locks.locktype = blocked_locks.locktype
       AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
       AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
       AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
       AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
       AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
       AND blocking_locks.transactionid IS NOT DISTINCT FROM
blocked_locks.transactionid
       AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
       AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
       AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
       AND blocking_locks.pid != blocked_locks.pid
   JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
   WHERE NOT blocked_locks.GRANTED;
```

Reduce concurrency when it is high

The `Lock:tuple` event might occur constantly, especially in a busy workload time. In this situation, consider reducing the high concurrency for very busy rows. Often, just a few rows control a queue or the Boolean logic, which makes these rows very busy.

You can reduce concurrency by using different approaches based in the business requirement, application logic, and workload type. For example, you can do the following:

- Redesign your table and data logic to reduce high concurrency.
- Change the application logic to reduce high concurrency at the row level.
- Leverage and redesign queries with row-level locks.
- Use the `NOWAIT` clause with retry operations.
- Consider using optimistic and hybrid-locking logic concurrency control.
- Consider changing the database isolation level.

Troubleshoot bottlenecks

The `Lock:tuple` can occur with bottlenecks such as CPU starvation or maximum usage of Amazon EBS bandwidth. To reduce bottlenecks, consider the following approaches:

- Scale up your instance class type.
- Optimize resource-intensive queries.
- Change the application logic.
- Archive data that is rarely accessed.

LWLock:BufferMapping (LWLock:buffer_mapping)

This event occurs when a session is waiting to associate a data block with a buffer in the shared buffer pool.

> **Note**
>
> This event is named `LWLock:BufferMapping` for RDS for PostgreSQL version 13 and higher versions. For RDS for PostgreSQL version 12 and older versions, this event is named `LWLock:buffer_mapping`.

Topics

- Supported engine versions
- Context
- Causes
- Actions

Supported engine versions

This wait event information is relevant for RDS for PostgreSQL version 9.6 and higher.

Context

The *shared buffer pool* is a PostgreSQL memory area that holds all pages that are or were being used by processes. When a process needs a page, it reads the page into the shared buffer pool. The

`shared_buffers` parameter sets the shared buffer size and reserves a memory area to store the table and index pages. If you change this parameter, make sure to restart the database. .

The `LWLock:buffer_mapping` wait event occurs in the following scenarios:

- A process searches the buffer table for a page and acquires a shared buffer mapping lock.
- A process loads a page into the buffer pool and acquires an exclusive buffer mapping lock.
- A process removes a page from the pool and acquires an exclusive buffer mapping lock.

Causes

When this event appears more than normal, possibly indicating a performance problem, the database is paging in and out of the shared buffer pool. Typical causes include the following:

- Large queries
- Bloated indexes and tables
- Full table scans
- A shared pool size that is smaller than the working set

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- Monitor buffer-related metrics
- Assess your indexing strategy
- Reduce the number of buffers that must be allocated quickly

Monitor buffer-related metrics

When `LWLock:buffer_mapping` waits spike, investigate the buffer hit ratio. You can use these metrics to get a better understanding of what is happening in the buffer cache. Examine the following metrics:

`blks_hit`

This Performance Insights counter metric indicates the number of blocks that were retrieved from the shared buffer pool. After the `LWLock:buffer_mapping` wait event appears, you might observe a spike in `blks_hit`.

`blks_read`

This Performance Insights counter metric indicates the number of blocks that required I/O to be read into the shared buffer pool. You might observe a spike in `blks_read` in the lead-up to the `LWLock:buffer_mapping` wait event.

Assess your indexing strategy

To confirm that your indexing strategy is not degrading performance, check the following:

Index bloat

Ensure that index and table bloat aren't leading to unnecessary pages being read into the shared buffer. If your tables contain unused rows, consider archiving the data and removing the rows from the tables. You can then rebuild the indexes for the resized tables.

Indexes for frequently used queries

To determine whether you have the optimal indexes, monitor DB engine metrics in Performance Insights. The `tup_returned` metric shows the number of rows read. The `tup_fetched` metric shows the number of rows returned to the client. If `tup_returned` is significantly larger than `tup_fetched`, the data might not be properly indexed. Also, your table statistics might not be current.

Reduce the number of buffers that must be allocated quickly

To reduce the `LWLock:buffer_mapping` wait events, try to reduce the number of buffers that must be allocated quickly. One strategy is to perform smaller batch operations. You might be able to achieve smaller batches by partitioning your tables.

LWLock:BufferIO (IPC:BufferIO)

The `LWLock:BufferIO` event occurs when RDS for PostgreSQL is waiting for other processes to finish their input/output (I/O) operations when concurrently trying to access a page. Its purpose is for the same page to be read into the shared buffer.

Topics

- Relevant engine versions
- Context
- Causes
- Actions

Relevant engine versions

This wait event information is relevant for all RDS for PostgreSQL versions. For RDS for PostgreSQL 12 and earlier versions this wait event is named as lwlock:buffer_io whereas in RDS for PostgreSQL 13 version it is named as lwlock:bufferio. From RDS for PostgreSQL 14 version BufferIO wait event moved from `LWLock` to `IPC` wait event type (IPC:BufferIO).

Context

Each shared buffer has an I/O lock that is associated with the `LWLock:BufferIO` wait event, each time a block (or a page) has to be retrieved outside the shared buffer pool.

This lock is used to handle multiple sessions that all require access to the same block. This block has to be read from outside the shared buffer pool, defined by the `shared_buffers` parameter.

As soon as the page is read inside the shared buffer pool, the `LWLock:BufferIO` lock is released.

> **Note**
>
> The `LWLock:BufferIO` wait event precedes the IO:DataFileRead wait event. The `IO:DataFileRead` wait event occurs while data is being read from storage.

For more information on lightweight locks, see Locking Overview.

Causes

Common causes for the `LWLock:BufferIO` event to appear in top waits include the following:

- Multiple backends or connections trying to access the same page that's also pending an I/O operation
- The ratio between the size of the shared buffer pool (defined by the `shared_buffers` parameter) and the number of buffers needed by the current workload

- The size of the shared buffer pool not being well balanced with the number of pages being evicted by other operations
- Large or bloated indexes that require the engine to read more pages than necessary into the shared buffer pool
- Lack of indexes that forces the DB engine to read more pages from the tables than necessary
- Checkpoints occurring too frequently or needing to flush too many modified pages
- Sudden spikes for database connections trying to perform operations on the same page

Actions

We recommend different actions depending on the causes of your wait event:

- Tune `max_wal_size` and `checkpoint_timeout` based on your workload peak time if you see `LWLock:BufferIO` coinciding with `BufferCacheHitRatio` metric dips. Then identify which query might be causing it.
- Verify whether you have unused indexes, then remove them.
- Use partitioned tables (which also have partitioned indexes). Doing this helps to keep index reordering low and reduces its impact.
- Avoid indexing columns unnecessarily.
- Prevent sudden database connection spikes by using a connection pool.
- Restrict the maximum number of connections to the database as a best practice.

LWLock:buffer_content (BufferContent)

The `LWLock:buffer_content` event occurs when a session is waiting to read or write a data page in memory while another session has that page locked for writing. In RDS for PostgreSQL 13 and higher, this wait event is called `BufferContent`.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

To read or manipulate data, PostgreSQL accesses it through shared memory buffers. To read from the buffer, a process gets a lightweight lock (LWLock) on the buffer content in shared mode. To write to the buffer, it gets that lock in exclusive mode. Shared locks allow other processes to concurrently acquire shared locks on that content. Exclusive locks prevent other processes from getting any type of lock on it.

The `LWLock:buffer_content` (`BufferContent`) event indicates that multiple processes are attempting to get a lock on contents of a specific buffer.

Likely causes of increased waits

When the `LWLock:buffer_content` (`BufferContent`) event appears more than normal, possibly indicating a performance problem, typical causes include the following:

Increased concurrent updates to the same data

There might be an increase in the number of concurrent sessions with queries that update the same buffer content. This contention can be more pronounced on tables with a lot of indexes.

Workload data is not in memory

When data that the active workload is processing is not in memory, these wait events can increase. This effect is because processes holding locks can keep them longer while they perform disk I/O operations.

Excessive use of foreign key constraints

Foreign key constraints can increase the amount of time a process holds onto a buffer content lock. This effect is because read operations require a shared buffer content lock on the referenced key while that key is being updated.

Actions

We recommend different actions depending on the causes of your wait event. You might identify `LWLock:buffer_content` (`BufferContent`) events by using Amazon RDS Performance Insights or by querying the view `pg_stat_activity`.

Topics

- Improve in-memory efficiency
- Reduce usage of foreign key constraints
- Remove unused indexes
- Increase the cache size when using sequences

Improve in-memory efficiency

To increase the chance that active workload data is in memory, partition tables or scale up your instance class. For information about DB instance classes, see DB instance classes.

Reduce usage of foreign key constraints

Investigate workloads experiencing high numbers of `LWLock:buffer_content` (`BufferContent`) wait events for usage of foreign key constraints. Remove unnecessary foreign key constraints.

Remove unused indexes

For workloads experiencing high numbers of `LWLock:buffer_content` (`BufferContent`) wait events, identify unused indexes and remove them.

Increase the cache size when using sequences

If your tables uses sequences, increase the cache size to remove contention on sequence pages and index pages. Each sequence is a single page in shared memory. The pre-defined cache is per connection. This might not be enough to handle the workload when many concurrent sessions are getting a sequence value.

LWLock:lock_manager (LWLock:lockmanager)

This event occurs when the RDS for PostgreSQL engine maintains the shared lock's memory area to allocate, check, and deallocate a lock when a fast path lock isn't possible.

Topics

- Supported engine versions
- Context

- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is relevant for RDS for PostgreSQL version 9.6 and higher.
For RDS for PostgreSQL releases older than version 13, the name of this wait event is `LWLock:lock_manager`. For RDS for PostgreSQL version 13 and higher, the name of this wait event is `LWLock:lockmanager`.

Context

When you issue a SQL statement, RDS for PostgreSQL records locks to protect the structure, data, and integrity of your database during concurrent operations. The engine can achieve this goal using a fast path lock or a path lock that isn't fast. A path lock that isn't fast is more expensive and creates more overhead than a fast path lock.

Fast path locking

To reduce the overhead of locks that are taken and released frequently, but that rarely conflict, backend processes can use fast path locking. The database uses this mechanism for locks that meet the following criteria:

- They use the DEFAULT lock method.
- They represent a lock on a database relation rather than a shared relation.
- They are weak locks that are unlikely to conflict.
- The engine can quickly verify that no conflicting locks can possibly exist.

The engine can't use fast path locking when either of the following conditions is true:

- The lock doesn't meet the preceding criteria.
- No more slots are available for the backend process.

To tune your queries for fast-path locking, you can use the following query.

```
SELECT count(*), pid, mode, fastpath
  FROM pg_locks
```

```
 WHERE fastpath IS NOT NULL
 GROUP BY 4,3,2
 ORDER BY pid, mode;
 count | pid  |      mode       | fastpath
-------+------+-----------------+----------
16 | 9185 | AccessShareLock | t
336 | 9185 | AccessShareLock | f
1 | 9185 | ExclusiveLock   | t
```

The following query shows only the total across the database.

```
SELECT count(*), mode, fastpath
  FROM pg_locks
 WHERE fastpath IS NOT NULL
 GROUP BY 3,2
 ORDER BY mode,1;
count |      mode       | fastpath
-------+-----------------+----------
16 | AccessShareLock | t
337 | AccessShareLock | f
1 | ExclusiveLock   | t
(3 rows)
```

For more information about fast path locking, see fast path in the PostgreSQL lock manager README and pg-locks in the PostgreSQL documentation.

Example of a scaling problem for the lock manager

In this example, a table named `purchases` stores five years of data, partitioned by day. Each partition has two indexes. The following sequence of events occurs:

1. You query many days worth of data, which requires the database to read many partitions.
2. The database creates a lock entry for each partition. If partition indexes are part of the optimizer access path, the database creates a lock entry for them, too.
3. When the number of requested locks entries for the same backend process is higher than 16, which is the value of `FP_LOCK_SLOTS_PER_BACKEND`, the lock manager uses the non–fast path lock method.

Modern applications might have hundreds of sessions. If concurrent sessions are querying the parent without proper partition pruning, the database might create hundreds or even thousands

of non–fast path locks. Typically, when this concurrency is higher than the number of vCPUs, the `LWLock:lock_manager` wait event appears.

> **Note**
>
> The `LWLock:lock_manager` wait event isn't related to the number of partitions or indexes in a database schema. Instead, it's related to the number of non–fast path locks that the database must control.

Likely causes of increased waits

When the `LWLock:lock_manager` wait event occurs more than normal, possibly indicating a performance problem, the most likely causes of sudden spikes are as follows:

- Concurrent active sessions are running queries that don't use fast path locks. These sessions also exceed the maximum vCPU.
- A large number of concurrent active sessions are accessing a heavily partitioned table. Each partition has multiple indexes.
- The database is experiencing a connection storm. By default, some applications and connection pool software create more connections when the database is slow. This practice makes the problem worse. Tune your connection pool software so that connection storms don't occur.
- A large number of sessions query a parent table without pruning partitions.
- A data definition language (DDL), data manipulation language (DML), or a maintenance command exclusively locks either a busy relation or tuples that are frequently accessed or modified.

Actions

If the CPU wait event occurs, it doesn't necessarily indicate a performance problem. Respond to this event only when performance degrades and this wait event is dominating DB load.

Topics

- Use partition pruning
- Remove unnecessary indexes
- Tune your queries for fast path locking

- Tune for other wait events
- Reduce hardware bottlenecks
- Use a connection pooler
- Upgrade your RDS for PostgreSQL version

Use partition pruning

Partition pruning is a query optimization strategy for declaratively partitioned tables that excludes unneeded partitions from table scans, thereby improving performance. Partition pruning is turned on by default. If it is turned off, turn it on as follows.

```
SET enable_partition_pruning = on;
```

Queries can take advantage of partition pruning when their `WHERE` clause contains the column used for the partitioning. For more information, see Partition Pruning in the PostgreSQL documentation.

Remove unnecessary indexes

Your database might contain unused or rarely used indexes. If so, consider deleting them. Do either of the following:

- Learn how to find unnecessary indexes by reading Unused Indexes in the PostgreSQL wiki.
- Run PG Collector. This SQL script gathers database information and presents it in a consolidated HTML report. Check the "Unused indexes" section. For more information, see pg-collector in the AWS Labs GitHub repository.

Tune your queries for fast path locking

To find out whether your queries use fast path locking, query the `fastpath` column in the `pg_locks` table. If your queries aren't using fast path locking, try to reduce number of relations per query to fewer than 16.

Tune for other wait events

If `LWLock:lock_manager` is first or second in the list of top waits, check whether the following wait events also appear in the list:

- `Lock:Relation`
- `Lock:transactionid`
- `Lock:tuple`

If the preceding events appear high in the list, consider tuning these wait events first. These events can be a driver for `LWLock:lock_manager`.

Reduce hardware bottlenecks

You might have a hardware bottleneck, such as CPU starvation or maximum usage of your Amazon EBS bandwidth. In these cases, consider reducing the hardware bottlenecks. Consider the following actions:

- Scale up your instance class.
- Optimize queries that consume large amounts of CPU and memory.
- Change your application logic.
- Archive your data.

For more information about CPU, memory, and EBS network bandwidth, see Amazon RDS Instance Types.

Use a connection pooler

If your total number of active connections exceeds the maximum vCPU, more OS processes require CPU than your instance type can support. In this case, consider using or tuning a connection pool. For more information about the vCPUs for your instance type, see Amazon RDS Instance Types.

For more information about connection pooling, see the following resources:

- Amazon RDS Proxy
- pgbouncer
- Connection Pools and Data Sources in the *PostgreSQL Documentation*

Upgrade your RDS for PostgreSQL version

If your current version of RDS for PostgreSQL is lower than 12, upgrade to version 12 or higher. PostgreSQL versions 12 and later have an improved partition mechanism. For more information

about version 12, see PostgreSQL 12.0 Release Notes. For more information about upgrading RDS for PostgreSQL, see Upgrades of the RDS for PostgreSQL DB engine.

LWLock:pg_stat_statements

The LWLock:pg_stat_statements wait event occurs when the `pg_stat_statements` extension takes an exclusive lock on the hash table that tracks SQL statements. This happens in the following scenarios:

- When the number of tracked statements reaches the configured `pg_stat_statements.max` parameter value and there is a need to make room for more entries, the extension performs a sort on the number of calls, removes the 5% of the least-executed statements, and re-populates the hash with the remaining entries.
- When `pg_stat_statements` performs a `garbage collection` operation to the `pgss_query_texts.stat` file on disk and rewrites the file.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

Understanding the pg_stat_statements extension – The pg_stat_statements extension tracks SQL statement execution statistics in a hash table. The extension tracks SQL statements up to the limit defined by the `pg_stat_statements.max` parameter. This parameter determines the maximum number of statements that can be tracked which corresponds to the maximum number of rows in the pg_stat_statements view.

Statement statistics persistence – The extension persists statement statistics across instance restarts by:

- Writing data to a file named pg_stat_statements.stat
- Using the pg_stat_statements.save parameter to control persistence behavior

When pg_stat_statements.save is set to:

- on (default): Statistics are saved at shutdown and reloaded at server start
- off: Statistics are neither saved at shutdown nor reloaded at server start

Query text storage – The extension stores the text of tracked queries in a file named `pgss_query_texts.stat`. This file can grow to double the average size of all tracked SQL statements before garbage collection occurs. The extension requires an exclusive lock on the hash table during cleanup operations and rewrite `pgss_query_texts.stat` file.

Statement deallocation process – When the number of tracked statements reaches the `pg_stat_statements.max` limit and new statements need to be tracked, the extension:

- Takes an exclusive lock (LWLock:pg_stat_statements) on the hash table.
- Loads existing data into local memory.
- Performs a quicksort based on the number of calls.
- Removes the least-called statements (bottom 5%).
- Re-populates the hash table with the remaining entries.

Monitoring statement deallocation – In PostgreSQL 14 and later, you can monitor statement deallocation using the pg_stat_statements_info view. This view includes a dealloc column that shows how many times statements were deallocated to make room for new ones

If the deallocation of statements occurs frequently, it will lead to more frequent garbage collection of the `pgss_query_texts.stat` file on disk.

Likely causes of increased waits

The typical causes of increased `LWLock:pg_stat_statements` waits include:

- An increase in the number of unique queries used by the application.
- The `pg_stat_statements.max` parameter value being small compared to the number of unique queries being used.

Actions

We recommend different actions depending on the causes of your wait event. You might identify `LWLock:pg_stat_statements` events by using Amazon RDS Performance Insights or by querying the view `pg_stat_activity`.

Adjust the following `pg_stat_statements` parameters to control tracking behavior and reduce LWLock:pg_stat_ statements wait events.

Topics

- Disable pg_stat_statements.track parameter
- Increase pg_stat_statements.max parameter
- Disable pg_stat_statements.track_utility parameter

Disable pg_stat_statements.track parameter

If the LWLock:pg_stat_statements wait event is adversely impacting database performance, and a rapid solution is required before further analysis of the `pg_stat_statements` view to identify the root cause, the `pg_stat_statements.track` parameter can be disabled by setting it to none. This will disable the collection of statement statistics.

Increase pg_stat_statements.max parameter

To reduce deallocation and minimize garbage collection of the `pgss_query_texts.stat` file on disk, increase the value of the `pg_stat_statements.max` parameter. The default value is `5,000`.

> **Note**
>
> The `pg_stat_statements.max` parameter is static. You must restart your DB instance to apply any changes to this parameter.

Disable pg_stat_statements.track_utility parameter

You can analyze the pg_stat_statements view to determine which utility commands are consuming the most resources tracked by `pg_stat_statements`.

The pg_stat_statements.track_utility parameter controls whether the module tracks utility commands, which include all commands except SELECT, INSERT, UPDATE, DELETE, and MERGE. By default, this parameter is set to on.

For example, when your application uses many savepoint queries, which are inherently unique, it can increase statement deallocation. To address this, you can disable the pg_stat_statements.track_utility parameter to stop pg_stat_statements from tracking savepoint queries.

> **Note**
>
> The pg_stat_statements.track_utility parameter is a dynamic parameter. You can change its value without restarting your database instance.

Example of unique save point queries in pg_stat_statements

```
                    query                         |       queryid
--------------------------------------------------+---------------------
 SAVEPOINT JDBC_SAVEPOINT_495701                  | -7249565344517699703
 SAVEPOINT JDBC_SAVEPOINT_1320                    | -1572997038849006629
 SAVEPOINT JDBC_SAVEPOINT_26739                   |  54791337410474486
 SAVEPOINT JDBC_SAVEPOINT_1294466                 |  8170064357463507593
 ROLLBACK TO SAVEPOINT JDBC_SAVEPOINT_65016       | -33608214779996400
 SAVEPOINT JDBC_SAVEPOINT_14185                   | -2175035613806809562
 SAVEPOINT JDBC_SAVEPOINT_45837                   | -6201592986750645383
 SAVEPOINT JDBC_SAVEPOINT_1324                    |  6388797791882029332
```

PostgreSQL 17 introduces several enhancements for utility command tracking:

- Savepoint names are now displayed as constants.
- Global Transaction IDs (GIDs) of two-phase commit commands are now displayed as constants.
- Names of DEALLOCATE statements are shown as constants.
- CALL parameters are now displayed as constants.

LWLock:SubtransSLRU (LWLock:SubtransControlLock)

The `LWLock:SubtransSLRU` and `LWLock:SubtransBuffer` wait events indicate that a session is waiting to access the simple least-recently used (SLRU) cache for subtransaction information. This occurs when determining transaction visibility and parent-child relationships.

- `LWLock:SubtransSLRU`: A process is waiting to access the simple least-recently used (SLRU) cache for a subtransaction. In RDS for PostgreSQL prior to version 13, this wait event is called `SubtransControlLock`.
- `LWLock:SubtransBuffer`: A process is waiting for I/O on a simple least-recently used (SLRU) buffer for a subtransaction. In RDS for PostgreSQL prior to version 13, this wait event is called `subtrans`.

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

Understanding subtransactions – A subtransaction is a transaction within a transaction in PostgreSQL. It's also known as a nested transaction.

Subtransactions are typically created when you use:

- `SAVEPOINT` commands
- Exception blocks (`BEGIN/EXCEPTION/END`)

Subtransactions let you roll back parts of a transaction without affecting the entire transaction. This gives you fine-grained control over transaction management.

Implementation details – PostgreSQL implements subtransactions as nested structures within main transactions. Each subtransaction gets its own transaction ID.

Key implementation aspects:

- Transaction IDs are tracked in `pg_xact`
- Parent-child relationships are stored in `pg_subtrans` subdirectory under `PGDATA`
- Each database session can maintain up to 64 active subtransactions
- Exceeding this limit causes subtransaction overflow, which requires accessing the simple least-recently used (SLRU) cache for subtransaction information

Likely causes of increased waits

Common causes of subtransaction SLRU contention include:

- **Excessive use of SAVEPOINT and EXCEPTION handling** – PL/pgSQL procedures with `EXCEPTION` handlers automatically create implicit savepoints, regardless of whether exceptions occur. Each `SAVEPOINT` initiates a new subtransaction. When a single transaction accumulates more than 64 subtransactions, it triggers a subtransaction SLRU overflow.
- **Driver and ORM configurations** – `SAVEPOINT` usage can be explicit in application code or implicit through driver configurations. Many commonly used ORM tools and application frameworks support nested transactions natively. Here are some common examples:
  - The JDBC driver parameter `autosave`, if set to `always` or `conservative`, generates savepoints before each query.
  - Spring Framework transaction definitions when set to `propagation_nested`.
  - Rails when `requires_new: true` is set.
  - SQLAlchemy when `session.begin_nested` is used.
  - Django when nested `atomic()` blocks are used.
  - GORM when `Savepoint` is used.
  - psqlODBC when rollback level setting is set to statement-level rollback (for example, `PROTOCOL=7.4-2`).
- **High concurrent workloads with long-running transactions and subtransactions** – When subtransaction SLRU overflow occurs during high concurrent workloads and long-running transactions and subtransactions, PostgreSQL experiences increased contention. This manifests as elevated wait events for `LWLock:SubtransBuffer` and `LWLock:SubtransSLRU` locks.

Actions

We recommend different actions depending on the causes of your wait event. Some actions provide immediate relief, while others require investigation and long-term correction.

Topics

- Monitoring subtransaction usage
- Configuring memory parameters
- Long-term actions

Monitoring subtransaction usage

For PostgreSQL versions 16.1 and later, use the following query to monitor subtransaction counts and overflow status per backend. This query joins backend statistics with activity information to show which processes are using subtransactions:

```
SELECT a.pid, usename, query, state, wait_event_type,
       wait_event, subxact_count, subxact_overflowed
FROM (SELECT id, pg_stat_get_backend_pid(id) pid, subxact_count, subxact_overflowed
      FROM pg_stat_get_backend_idset() id
           JOIN LATERAL pg_stat_get_backend_subxact(id) AS s ON true
     ) a
JOIN pg_stat_activity b ON a.pid = b.pid;
```

For PostgreSQL versions 13.3 and later, monitor the `pg_stat_slru` view for subtransaction cache pressure. The following SQL query retrieves SLRU cache statistics for the Subtrans component:

```
SELECT * FROM pg_stat_slru WHERE name = 'Subtrans';
```

A consistently increasing `blks_read` value indicates frequent disk access for uncached subtransactions, signaling potential SLRU cache pressure.

Configuring memory parameters

For PostgreSQL 17.1 and later, you can configure the subtransaction SLRU cache size using the `subtransaction_buffers` parameter. The following configuration example shows how to set the subtransaction buffer parameter:

```
subtransaction_buffers = 128
```

This parameter specifies the amount of shared memory used to cache subtransaction contents (`pg_subtrans`). When specified without units, the value represents blocks of `BLCKSZ` bytes, typically 8KB each. For example, setting the value to 128 allocates 1MB (128 * 8kB) of memory for the subtransaction cache.

> **Note**
>
> You can set this parameter at the cluster level so that all instances remain consistent. Test and adjust the value to suit your specific workload requirements and instance class. You must reboot the writer instance for parameter changes to take effect.

Long-term actions

- **Examine application code and configurations** – Review your application code and database driver configurations for both explicit and implicit `SAVEPOINT` usage and subtransactions usage in general. Identify transactions potentially generating over 64 subtransactions.
- **Reduce savepoint usage** – Minimize the use of savepoints in your transactions:
  - Review PL/pgSQL procedures and functions with EXCEPTION blocks. EXCEPTION blocks automatically create implicit savepoints, which can contribute to subtransaction overflow. Each EXCEPTION clause creates a subtransaction, regardless of whether an exception actually occurs during execution.

    Example

    Example 1: Problematic EXCEPTION block usage

    The following code example shows problematic EXCEPTION block usage that creates multiple subtransactions:

```
CREATE OR REPLACE FUNCTION process_user_data()
RETURNS void AS $$
DECLARE
    user_record RECORD;
BEGIN
    FOR user_record IN SELECT * FROM users LOOP
        BEGIN
            -- This creates a subtransaction for each iteration
            INSERT INTO user_audit (user_id, action, timestamp)
            VALUES (user_record.id, 'processed', NOW());

            UPDATE users
            SET last_processed = NOW()
            WHERE id = user_record.id;

        EXCEPTION
            WHEN unique_violation THEN
                -- Handle duplicate audit entries
                UPDATE user_audit
                SET timestamp = NOW()
                WHERE user_id = user_record.id AND action = 'processed';
        END;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

The following improved code example reduces subtransaction usage by using UPSERT instead of exception handling:

```
CREATE OR REPLACE FUNCTION process_user_data()
RETURNS void AS $$
DECLARE
    user_record RECORD;
BEGIN
    FOR user_record IN SELECT * FROM users LOOP
        -- Use UPSERT to avoid exception handling
        INSERT INTO user_audit (user_id, action, timestamp)
        VALUES (user_record.id, 'processed', NOW())
        ON CONFLICT (user_id, action)
        DO UPDATE SET timestamp = NOW();
```

```
        UPDATE users
        SET last_processed = NOW()
        WHERE id = user_record.id;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Example

Example 2: STRICT exception handler

The following code example shows problematic EXCEPTION handling with NO_DATA_FOUND:

```
CREATE OR REPLACE FUNCTION get_user_email(p_user_id INTEGER)
RETURNS TEXT AS $$
DECLARE
    user_email TEXT;
BEGIN
    BEGIN
        -- STRICT causes an exception if no rows or multiple rows found
        SELECT email INTO STRICT user_email
        FROM users
        WHERE id = p_user_id;

        RETURN user_email;

    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN 'Email not found';
    END;
END;
$$ LANGUAGE plpgsql;
```

The following improved code example avoids subtransactions by using IF NOT FOUND instead of exception handling:

```
CREATE OR REPLACE FUNCTION get_user_email(p_user_id INTEGER)
RETURNS TEXT AS $$
```

```
DECLARE
    user_email TEXT;
BEGIN
     SELECT email INTO user_email
     FROM users
     WHERE id = p_user_id;

     IF NOT FOUND THEN
         RETURN 'Email not found';
     ELSE
         RETURN user_email;
     END IF;
END;
$$ LANGUAGE plpgsql;
```

  - JDBC driver – The `autosave` parameter, if set to `always` or `conservative`, generates savepoints before each query. Evaluate whether the `never` setting would be acceptable for your application.
  - PostgreSQL ODBC driver (psqlODBC) — The rollback level setting (for statement-level rollback) creates implicit savepoints to enable statement rollback functionality. Evaluate whether transaction-level or no rollback would be acceptable for your application.
  - Examine ORM transaction configurations
  - Consider alternative error handling strategies that don't require savepoints
- **Optimize transaction design** – Restructure transactions to avoid excessive nesting and reduce the likelihood of subtransaction overflow conditions.
- **Reduce long-running transactions** – Long-running transactions can exacerbate subtransaction issues by holding onto subtransaction information longer. Monitor Performance Insights metrics and configure the `idle_in_transaction_session_timeout` parameter to automatically terminate idle transactions.
- Monitor Performance Insights metrics – Track metrics including `idle_in_transaction_count` (number of sessions in idle in transaction state) and `idle_in_transaction_max_time` (duration of the longest running idle transaction) to detect long-running transactions.
- Configure `idle_in_transaction_session_timeout` – Set this parameter in your parameter group to automatically terminate idle transactions after a specified duration.
- Proactive monitoring – Monitor for high occurrences of `LWLock:SubtransBuffer` and `LWLock:SubtransSLRU` wait events to detect subtransaction-related contention before it becomes critical.

Timeout:PgSleep

The `Timeout:PgSleep` event occurs when a server process has called the `pg_sleep` function and is waiting for the sleep timeout to expire.

Topics

- Supported engine versions
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Likely causes of increased waits

This wait event occurs when an application, stored function, or user issues a SQL statement that calls one of the following functions:

- `pg_sleep`
- `pg_sleep_for`
- `pg_sleep_until`

The preceding functions delay execution until the specified number of seconds have elapsed. For example, `SELECT pg_sleep(1)` pauses for 1 second. For more information, see Delaying Execution in the PostgreSQL documentation.

Actions

Identify the statement that was running the `pg_sleep` function. Determine if the use of the function is appropriate.

Timeout:VacuumDelay

The `Timeout:VacuumDelay` event indicates that the cost limit for vacuum I/O has been exceeded and that the vacuum process has been put to sleep. Vacuum operations stop for the

duration specified in the respective cost delay parameter and then it resumes its work. For the manual vacuum command, the delay is specified in the `vacuum_cost_delay` parameter. For the autovacuum daemon, the delay is specified in the `autovacuum_vacuum_cost_delay parameter.`

Topics

- Supported engine versions
- Context
- Likely causes of increased waits
- Actions

Supported engine versions

This wait event information is supported for all versions of RDS for PostgreSQL.

Context

PostgreSQL has both an autovacuum daemon and a manual vacuum command. The autovacuum process is "on" by default for RDS for PostgreSQL DB instances. The manual vacuum command is used on an as-needed basis, for example, to purge tables of dead tuples or generate new statistics.

When vacuuming is underway, PostgreSQL uses an internal counter to keep track of estimated costs as the system performs various I/O operations. When the counter reaches the value specified by the cost limit parameter, the process performing the operation sleeps for the brief duration specified in the cost delay parameter. It then resets the counter and continues operations.

The vacuum process has parameters that can be used to regulate resource consumption. The autovacuum and the manual vacuum command have their own parameters for setting the cost limit value. They also have their own parameters to specify a cost delay, an amount of time to put the vacuum to sleep when the limit is reached. In this way, the cost delay parameter works as a throttling mechanism for resource consumption. In the following lists, you can find description of these parameters.

Parameters that affect throttling of the autovacuum daemon

- `autovacuum_vacuum_cost_limit` – Specifies the cost limit value to use in automatic vacuum operations. Increasing the setting for this parameter allows the vacuum process to use more resources and decreases the `Timeout:VacuumDelay` wait event.

- `autovacuum_vacuum_cost_delay` – Specifies the cost delay value to use in automatic vacuum operations. The default value is 2 milliseconds. Setting the delay parameter to 0 turns off the throttling mechanism and thus, the `Timeout:VacuumDelay` wait event won't appear.

For more information, see Automatic Vacuuming in the PostgreSQL documentation.

Parameters that affect throttling of the manual vacuum process

- `vacuum_cost_limit` – The threshold at which the vacuuming process is put to sleep. By default, the limit is 200. This number represents the accumulated cost estimates for extra I/O needed by various resources. Increasing this value reduces the number of the `Timeout:VacuumDelay` wait event.
- `vacuum_cost_delay` – The amount of time that the vacuum process sleeps when the vacuum cost limit has been reached. The default setting is 0, which means that this feature is off. You can set this to an integer value to specify the number of milliseconds to turn on this feature, but we recommend that you leave it at its default setting.

For more information about the `vacuum_cost_delay` parameter, see Resource Consumption in the PostgreSQL documentation.

To learn more about how to configure and use the autovacuum with RDS for PostgreSQL, see Working with PostgreSQL autovacuum on Amazon RDS for PostgreSQL.

Likely causes of increased waits

The `Timeout:VacuumDelay` is affected by the balance between the cost limit parameter settings (`vacuum_cost_limit`, `autovacuum_vacuum_cost_limit`) and the cost delay parameters (`vacuum_cost_delay`, `autovacuum_vacuum_cost_delay`) that control the vacuum's sleep duration. Raising a cost limit parameter value allows more resources to be used by the vacuum before being put to sleep. That results in fewer `Timeout:VacuumDelay` wait events. Increasing either of the delay parameters causes the `Timeout:VacuumDelay` wait event to occur more frequently and for longer periods of time.

The `autovacuum_max_workers` parameter setting can also increase numbers of the `Timeout:VacuumDelay`. Each additional autovacuum worker process contributes to the internal counter mechanism, and thus the limit can be reached more quickly than with a single autovacuum worker process. As the cost limit is reached more quickly, the cost delay is put to effect more

frequently, resulting in more `Timeout:VacuumDelay` wait events. For more information, see autovacuum_max_workers in the PostgreSQL documentation.

Large objects, such as 500GB or larger, also raise this wait event because it can take some time for the vacuum to complete processing large objects.

Actions

If the vacuum operations complete as expected, no remediation is needed. In other words, this wait event doesn't necessarily indicate a problem. It indicates that the vacuum is being put to sleep for the period of time specified in the delay parameter so that resources can be applied to other processes that need to complete.

If you want vacuum operations to complete faster, you can lower the delay parameters. This shortens the time that the vacuum sleeps.

Tuning RDS for PostgreSQL with Amazon DevOps Guru proactive insights

DevOps Guru proactive insights detects conditions on your RDS for PostgreSQL DB instances that can cause problems, and lets you know about them before they occur. Proactive insights can alert you to a long running idle in transaction connection. For more information about troubleshooting long running idle in transaction connections, see Database has long running idle in transaction connection

DevOps Guru can do the following:

- Prevent many common database issues by cross-checking your database configuration against common recommended settings.
- Alert you to critical issues in your fleet that, if left unchecked, can lead to larger problems later.
- Alert you to newly discovered problems.

Every proactive insight contains an analysis of the cause of the problem and recommendations for corrective actions.

For more information about Amazon DevOps Guru for Amazon RDS, see Analyzing performance anomalies with Amazon DevOps Guru for Amazon RDS.

Database has long running idle in transaction connection

A connection to the database has been in the `idle in transaction` state for more than 1800 seconds.

Topics

- Supported engine versions
- Context
- Likely causes for this issue
- Actions
- Relevant metrics

Supported engine versions

This insight information is supported for all versions of RDS for PostgreSQL.

Context

A transaction in the `idle in transaction` state can hold locks that block other queries. It can also prevent VACUUM (including autovacuum) from cleaning up dead rows, leading to index or table bloat or transaction ID wraparound.

Likely causes for this issue

A transaction initiated in an interactive session with BEGIN or START TRANSACTION hasn't ended by using a COMMIT, ROLLBACK, or END command. This causes the transaction to move to `idle in transaction` state.

Actions

You can find idle transactions by querying `pg_stat_activity`.

In your SQL client, run the following query to list all connections in `idle in transaction` state and to order them by duration:

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
WHERE state  = 'idle in transaction'
AND   xact_start is not null
ORDER BY 1 DESC;
```

We recommend different actions depending on the causes of your insight.

Topics

- End transaction
- Terminate the connection
- Configure the idle_in_transaction_session_timeout parameter
- Check the AUTOCOMMIT status
- Check the transaction logic in your application code

End transaction

When you initiate a transaction in an interactive session with BEGIN or START TRANSACTION, it moves to `idle in transaction` state. It remains in this state until you end the transaction by issuing a COMMIT, ROLLBACK, END command or disconnect the connection completely to roll back the transaction.

Terminate the connection

Terminate the connection with an idle transaction using the following query:

```
SELECT pg_terminate_backend(pid);
```

pid is the process ID of the connection.

Configure the idle_in_transaction_session_timeout parameter

Configure the `idle_in_transaction_session_timeout` parameter in the parameter group. The advantage of configuring this parameter is that it does not require a manual intervention to terminate the long idle in transaction. For more information on this parameter, see the PostgreSQL documentation.

The following message will be reported in the PostgreSQL log file after the connection is terminated, when a transaction is in the idle_in_transaction state for longer than the specified time.

```
FATAL: terminating connection due to idle in transaction timeout
```

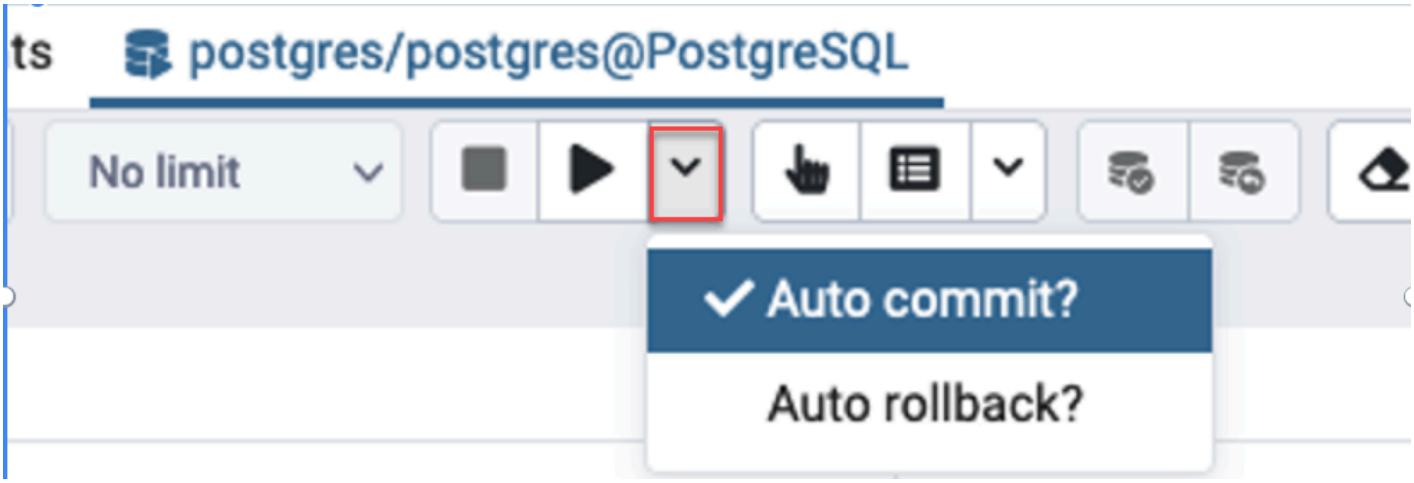
Check the AUTOCOMMIT status

AUTOCOMMIT is turned on by default. But if it is accidentally turned off in the client ensure that you turn it back on.

- In your psql client, run the following command:

  ```
  postgres=> \set AUTOCOMMIT on
  ```

- In pgadmin, turn it on by choosing the AUTOCOMMIT option from the down arrow.



Check the transaction logic in your application code

Investigate your application logic for possible problems. Consider the following actions:

- Check if the JDBC auto commit is set true in your application. Also, consider using explicit `COMMIT` commands in your code.
- Check your error handling logic to see whether it closes a transaction after errors.
- Check whether your application is taking long to process the rows returned by a query while the transaction is open. If so, consider coding the application to close the transaction before processing the rows.
- Check whether a transaction contains many long-running operations. If so, divide a single transaction into multiple transactions.

Relevant metrics

The following PI metrics are related to this insight:

- idle_in_transaction_count - Number of sessions in `idle in transaction` state.
- idle_in_transaction_max_time - The duration of the longest running transaction in the `idle in transaction` state.

Using PostgreSQL extensions with Amazon RDS for PostgreSQL

You can extend the functionality of PostgreSQL by installing a variety of extensions and modules. For example, to work with spatial data you can install and use the PostGIS extension. For more information, see Managing spatial data with the PostGIS extension. As another example, if you want to improve data entry for very large tables, you can consider partitioning your data by using the `pg_partman` extension. To learn more, see Managing PostgreSQL partitions with the pg_partman extension.

> **Note**
>
> RDS for PostgreSQL supports Trusted Language Extensions for PostgreSQL through the `pg_tle` extension, which you can add to your DB instance. By using this extension, developers can create their own PostgreSQL extensions in a safe environment that simplifies the setup and configuration requirements. To learn about RDS for PostgreSQL versions supporting `pg_tle` extension and for more information, see Working with Trusted Language Extensions for PostgreSQL.

In some cases, rather than installing an extension, you might add a specific module to the list of `shared_preload_libraries` in your RDS for PostgreSQL DB instance's custom DB parameter group. Typically, the default DB cluster parameter group loads only the `pg_stat_statements`, but several other modules are available to add to the list. For example, you can add scheduling capability by adding the `pg_cron` module, as detailed in Scheduling maintenance with the PostgreSQL pg_cron extension. As another example, you can log query execution plans by loading the `auto_explain` module. To learn more, see Logging execution plans of queries in the AWS knowledge center.

Depending on your version of RDS for PostgreSQL, installing an extension might require `rds_superuser` permissions, as follows:

- For RDS for PostgreSQL versions 12 and earlier versions, installing extensions requires `rds_superuser` privileges.
- For RDS for PostgreSQL version 13 and higher versions, users (roles) with create permissions on a given database instance can install and use any *trusted extensions*. For a list of trusted extensions, see PostgreSQL trusted extensions.

You can also specify precisely which extensions can be installed on your RDS for PostgreSQL DB instance, by listing them in the `rds.allowed_extensions` parameter. For more information, see Restricting installation of PostgreSQL extensions.

To learn more about the `rds_superuser` role, see Understanding PostgreSQL roles and permissions.

Topics

- Using functions from the orafce extension
- Using Amazon RDS delegated extension support for PostgreSQL
- Managing PostgreSQL partitions with the pg_partman extension
- Using pgAudit to log database activity
- Scheduling maintenance with the PostgreSQL pg_cron extension
- Using pglogical to synchronize data across instances
- Using pgactive to support active-active replication
- Reducing bloat in tables and indexes with the pg_repack extension
- Upgrading and using the PLV8 extension
- Using PL/Rust to write PostgreSQL functions in the Rust language
- Managing spatial data with the PostGIS extension

Using functions from the orafce extension

The orafce extension provides functions and operators that emulate a subset of functions and packages from an Oracle database. The orafce extension makes it easier for you to port an Oracle application to PostgreSQL. RDS for PostgreSQL versions 9.6.6 and higher support this extension. For more information about orafce, see orafce on GitHub.

> **Note**
>
> RDS for PostgreSQL doesn't support the `utl_file` package that is part of the orafce extension. This is because the `utl_file` schema functions provide read and write operations on operating-system text files, which requires superuser access to the underlying host. As a managed service, RDS for PostgreSQL doesn't provide host access.

To use the orafce extension

1. Connect to the DB instance with the primary user name that you used to create the DB instance.

   If you want to turn on orafce for a different database in the same DB instance, use the /c dbname psql command. Using this command, you change from the primary database after initiating the connection.
2. Turn on the orafce extension with the CREATE EXTENSION statement.

   ```
   CREATE EXTENSION orafce;
   ```
3. Transfer ownership of the oracle schema to the rds_superuser role with the ALTER SCHEMA statement.

   ```
   ALTER SCHEMA oracle OWNER TO rds_superuser;
   ```

   If you want to see the list of owners for the oracle schema, use the \dn psql command.

Using Amazon RDS delegated extension support for PostgreSQL

Using Amazon RDS delegated extension support for PostgreSQL, you can delegate the extension management to a user who need not be an `rds_superuser`. With this delegated extension support, a new role called `rds_extension` is created and you must assign this to a user to manage other extensions. This role can create, update, and drop extensions.

You can specify the extensions that can be installed on your RDS DB instance, by listing them in the `rds.allowed_extensions` parameter. For more information, see Using PostgreSQL extensions with Amazon RDS for PostgreSQL.

You can restrict the list of extensions available that can be managed by the user with the `rds_extension` role using `rds.allowed_delegated_extensions` parameter.

The delegated extension support is available in the following versions:

- All higher versions
- 16.4 and higher 16 versions
- 15.8 and higher 15 versions
- 14.13 and higher 14 versions
- 13.16 and higher 13 versions
- 12.20 and higher 12 versions

Topics

- Turning on delegate extension support to a user
- Configuration used in RDS delegated extension support for PostgreSQL
- Turning off the support for the delegated extension
- Benefits of using Amazon RDS delegated extension support
- Limitation of Amazon RDS delegated extension support for PostgreSQL
- Permissions required for certain extensions
- Security Considerations
- Drop extension cascade disabled
- Example extensions that can be added using delegated extension support

Turning on delegate extension support to a user

You must perform the following to enable delegate extension support to a user:

1. **Grant `rds_extension` role to a user** – Connect to the database as `rds_superuser` and execute the following command:

```
Postgres => grant rds_extension to user_name;
```

2. **Set the list of extensions available for delegated users to manage** – The `rds.allowed_delegated_extensions` allows you to specify a subset of the available extensions using `rds.allowed_extensions` in the DB cluster parameter. You can perform this at one of the following levels:

   - In the cluster or the instance parameter group, through the AWS Management Console or API. For more information, see Parameter groups for Amazon RDS.
   - Use the following command at the database level:

```
alter database database_name set rds.allowed_delegated_extensions =
 'extension_name_1,
                  extension_name_2,...extension_name_n';
```

   - Use the following command at the user level:

```
alter user user_name set rds.allowed_delegated_extensions = 'extension_name_1,
                  extension_name_2,...extension_name_n';
```

> **Note**
>
> You need not restart the database after changing the `rds.allowed_delegated_extensions` dynamic parameter.

3. **Allow access to the delegated user to objects created during the extension creation process** – Certain extensions create objects that require additional permissions to be granted before the user with `rds_extension` role can access them. The `rds_superuser` must grant the delegated user access to those objects. One of the options is to use an event trigger to automatically grant permission to the delegated user.

Example of event trigger

If you want to allow a delegated user with `rds_extension` to use extensions that require setting permissions on their objects created by the extension creation, you can customize the below example of an event trigger and add only the extensions for which you want the delegated users to have access to the full functionality. This event trigger can be created on template1 (the default template), therefore all database created from template1 will have that event trigger. When a delegated user installs the extension, this trigger will automatically grant ownership on the objects created by the extension.

```
CREATE OR REPLACE FUNCTION create_ext()

  RETURNS event_trigger AS $$

DECLARE

  schemaname TEXT;
  databaseowner TEXT;

  r RECORD;

BEGIN

  IF tg_tag = 'CREATE EXTENSION' and current_user != 'rds_superuser' THEN
    RAISE NOTICE 'SECURITY INVOKER';
    RAISE NOTICE 'user: %', current_user;
    FOR r IN SELECT * FROM pg_catalog.pg_event_trigger_ddl_commands()
    LOOP
        CONTINUE WHEN r.command_tag != 'CREATE EXTENSION' OR r.object_type !=
 'extension';

        schemaname = (
            SELECT n.nspname
            FROM pg_catalog.pg_extension AS e
            INNER JOIN pg_catalog.pg_namespace AS n
            ON e.extnamespace = n.oid
            WHERE e.oid = r.objid
        );

        databaseowner = (
```

```
            SELECT pg_catalog.pg_get_userbyid(d.datdba)
            FROM pg_catalog.pg_database d
            WHERE d.datname = current_database()
        );
        RAISE NOTICE 'Record for event trigger %, objid: %,tag: %, current_user: %,
 schema: %, database_owenr: %', r.object_identity, r.objid, tg_tag, current_user,
 schemaname, databaseowner;
        IF r.object_identity = 'address_standardizer_data_us' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_gaz TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_lex TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.us_rules TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'dict_int' THEN
            EXECUTE pg_catalog.format('ALTER TEXT SEARCH DICTIONARY %I.intdict
 OWNER TO %I;', schemaname, databaseowner);
        ELSIF r.object_identity = 'pg_partman' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.part_config TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON
 TABLE %I.part_config_sub TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE
 ON TABLE %I.custom_time_partitions TO %I WITH GRANT OPTION;', schemaname,
 databaseowner);
        ELSIF r.object_identity = 'postgis_topology' THEN
            EXECUTE pg_catalog.format('GRANT SELECT, UPDATE, INSERT, DELETE ON ALL
 TABLES IN SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT USAGE, SELECT ON ALL SEQUENCES IN
 SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA
 topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE pg_catalog.format('GRANT USAGE ON SCHEMA topology TO %I WITH
 GRANT OPTION;', databaseowner);
        END IF;
    END LOOP;
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE EVENT TRIGGER log_create_ext ON ddl_command_end EXECUTE PROCEDURE
 create_ext();
```

Configuration used in RDS delegated extension support for PostgreSQL

Configuration Name	Description	Default Value	Notes	Who can modify or grant permission
`rds.allowed_delegated_extensions`	This parameter limits the extensions a rds_extension role can manage in a database. It must be a subset of rds.allowed_extensions.	empty string	• By default, this parameter is empty string, which means that no extensions have been delegated to users with `rds_extension` . • Any supported extension can be added if the user has permission to do so. To do this, set the `rds.allowed_delegated_extensions` parameter to a string of comma-separated extension names. By adding a list of extensions to this parameter, you explicitly identify the extensions that	rds_superuser

Configuration Name	Description	Default Value	Notes	Who can modify or grant permission
			the user with the `rds_extension` role can install. • When set to `*`, it means that all extensions listed in `rds_allowed_extensions` are delegated to users with `rds_extension` role. To learn more about setting up this parameter, see Turning on delegate extension support to a user.	

Configuration Name	Description	Default Value	Notes	Who can modify or grant permission
rds.allowed_extensions	This parameter lets the customer limit the extensions that can be installed in the RDS DB instance. For more information, see Restricting installation of PostgreSQL extensions	"*"	By default, this parameter is set to "*", which means that all extensions supported on RDS for PostgreSQL and Aurora PostgreSQL are allowed to be created by users with necessary privileges. Empty means no extensions can be installed in the RDS DB instance.	administrator

Configuration Name	Description	Default Value	Notes	Who can modify or grant permission
rds-delegated_extension_allow_drascac	This parameter controls the ability for user with rds_extension to drop the extension using a cascade option.	off	By default, rds-delegated_extension_allow_drop_cascade is set to off. This means that users with rds_extension are not allowed to drop an extension using the cascade option. To grant that ability, the rds.delegated_extension_allow_drop_cascade parameter should be set to on.	rds_superuser

Turning off the support for the delegated extension

Turning off partially

The delegated users can't create new extensions but can still update existing extensions.

- Reset rds.allowed_delegated_extensions to the default value in the DB cluster parameter group.
- Use the following command at the database level:

```
alter database database_name reset rds.allowed_delegated_extensions;
```

- Use the following command at the user level:

```
alter user user_name reset rds.allowed_delegated_extensions;
```

Turning off fully

Revoking `rds_extension` role from a user will revert the user to standard permissions. The user can no longer create, update, or drop extensions.

```
postgres => revoke rds_extension from user_name;
```

Benefits of using Amazon RDS delegated extension support

By using Amazon RDS delegated extension support for PostgreSQL, you securely delegate the extension management to users who do not have the `rds_superuser` role. This feature provides the following benefits:

- You can easily delegate extension management to users of your choice.
- This doesn't require `rds_superuser` role.
- Provides ability to support different set of extensions for different databases in the same DB cluster.

Limitation of Amazon RDS delegated extension support for PostgreSQL

- Objects created during the extension creation process may require additional privileges for the extension to function properly.
- Some extensions can't be managed by the delegated extension user by default, including the following: `log_fdw`, `pg_cron`, `pg_tle`, `pgactive`, `pglogical`, `postgis_raster`, `postgis_tiger_geocoder`, `postgis_topology`.

Permissions required for certain extensions

In order to create, use, or update the following extensions, the delegated user should have the necessary privileges on the following functions, tables, and schema.

Exte s that need owne or perm ns	Function	Tables	Schema	Text Search Dictionary	Comment
addr tand er_da s	none	us_gaz, us_lex, us_lex, I.us_rule s	none	none	none
amch	bt_index_ check, bt_index_ parent_check	none	none	none	none
dict_	none	none	none	intdict	none
pg_p n	none	custom_ti me_partit ions, part_conf ig, part_conf ig_sub	none	none	none
pg_s tater	none	none	none	none	none
Post(	st_tileenvelope	spatial_ref_sys	none	none	none
post aster	none	none	none	none	none
post opol	none	topology, layer	topology	none	the delegated user Must be

Exte s that need own or perm ns	Function	Tables	Schema	Text Search Dictionary	Comment
					the database owner
log_f	create_fo reign_tab le_for_log_file	none	none	none	none
rds_t	role_pass word_encr yption_type	none	none	none	none
post iger_ oder	none	geocode_s ettings_d efault, geocode_s ettings	tiger	none	none
pg_f acem	pg_freespace	none	none	none	none
pg_v lity	pg_visibility	none	none	none	none

Security Considerations

Keep in mind that a user with `rds_extension` role will be able to manage extensions on all databases they have the connect privilege on. If the intention is to have a delegated user manage extension on a single database, a good practice is to revoke all privileges from public on each database, then explicitly grant the connect privilege for that specific database to the delegate user.

There are several extensions that can allow a user to access information from multiple database. Ensure the users you grant `rds_extension` has cross database capabilities before adding these extensions to `rds.allowed_delegated_extensions`. For example, `postgres_fdw` and `dblink` provide functionality to query across databases on the same instance or remote instances. `log_fdw` reads the postgres engine log files, which are for all databases in the instance, potentially containing slow queries or error messages from multiple databases. `pg_cron` enables running scheduled background jobs on the DB instance and can configure jobs to run in a different database.

Drop extension cascade disabled

The ability to drop the extension with cascade option by a user with the `rds_extension` role is controlled by `rds.delegated_extension_allow_drop_cascade` parameter. By default, `rds-delegated_extension_allow_drop_cascade` is set to `off`. This means that users with the `rds_extension` role are not allowed to drop an extension using the cascade option as shown in the below query.

```
DROP EXTENSION CASCADE;
```

As this will automatically drop objects that depend on the extension, and in turn all objects that depend on those objects. Attempting to use the cascade option will result in an error.

To grant that ability, the `rds.delegated_extension_allow_drop_cascade` parameter should be set to on.

Changing the `rds.delegated_extension_allow_drop_cascade` dynamic parameter doesn't require a database restart. You can do this at one of the following levels:

- In the cluster or the instance parameter group, through the AWS Management Console or API.
- Using the following command at the database level:

```
alter database database_name set rds.delegated_extension_allow_drop_cascade = 'on';
```

- Using the following command at the user level:

```
alter role tenant_user set rds.delegated_extension_allow_drop_cascade = 'on';
```

Example extensions that can be added using delegated extension support

- rds_tools

```
extension_test_db=> create extension rds_tools;
CREATE EXTENSION
extension_test_db=> SELECT * from rds_tools.role_password_encryption_type() where
 rolname = 'pg_read_server_files';
ERROR: permission denied for function role_password_encryption_type
```

- amcheck

```
extension_test_db=> CREATE TABLE amcheck_test (id int);
CREATE TABLE
extension_test_db=> INSERT INTO amcheck_test VALUES (generate_series(1,100000));
INSERT 0 100000
extension_test_db=> CREATE INDEX amcheck_test_btree_idx ON amcheck_test USING btree
 (id);
CREATE INDEX
extension_test_db=> create extension amcheck;
CREATE EXTENSION
extension_test_db=> SELECT bt_index_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_check
extension_test_db=> SELECT bt_index_parent_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_parent_check
```

- pg_freespacemap

```
extension_test_db=> create extension pg_freespacemap;
CREATE EXTENSION
extension_test_db=> SELECT * FROM pg_freespace('pg_authid');
ERROR: permission denied for function pg_freespace
extension_test_db=> SELECT * FROM pg_freespace('pg_authid',0);
ERROR: permission denied for function pg_freespace
```

- pg_visibility

```
extension_test_db=> create extension pg_visibility;
CREATE EXTENSION
extension_test_db=> select * from pg_visibility('pg_database'::regclass);
ERROR: permission denied for function pg_visibility
```

- postgres_fdw

```
extension_test_db=> create extension postgres_fdw;
CREATE EXTENSION
extension_test_db=> create server myserver foreign data wrapper postgres_fdw options
 (host 'foo', dbname 'foodb', port '5432');
ERROR: permission denied for foreign-data wrapper postgres_fdw
```

Managing PostgreSQL partitions with the pg_partman extension

PostgreSQL table partitioning provides a framework for high-performance handling of data input and reporting. Use partitioning for databases that require very fast input of large amounts of data. Partitioning also provides for faster queries of large tables. Partitioning helps maintain data without impacting the database instance because it requires less I/O resources.

By using partitioning, you can split data into custom-sized chunks for processing. For example, you can partition time-series data for ranges such as hourly, daily, weekly, monthly, quarterly, yearly, custom, or any combination of these. For a time-series data example, if you partition the table by hour, each partition contains one hour of data. If you partition the time-series table by day, the partitions holds one day's worth of data, and so on. The partition key controls the size of a partition.

When you use an `INSERT` or `UPDATE` SQL command on a partitioned table, the database engine routes the data to the appropriate partition. PostgreSQL table partitions that store the data are child tables of the main table.

During database query reads, the PostgreSQL optimizer examines the `WHERE` clause of the query and, if possible, directs the database scan to only the relevant partitions.

Starting with version 10, PostgreSQL uses declarative partitioning to implement table partitioning. This is also known as native PostgreSQL partitioning. Before PostgreSQL version 10, you used triggers to implement partitions.

PostgreSQL table partitioning provides the following features:

- Creation of new partitions at any time.
- Variable partition ranges.
- Detachable and reattachable partitions using data definition language (DDL) statements.

  For example, detachable partitions are useful for removing historical data from the main partition but keeping historical data for analysis.
- New partitions inherit the parent database table properties, including the following:
  - Indexes
  - Primary keys, which must include the partition key column
  - Foreign keys
  - Check constraints

- References
- Creating indexes for the full table or each specific partition.

You can't alter the schema for an individual partition. However, you can alter the parent table (such as adding a new column), which propagates to partitions.

Topics

- Overview of the PostgreSQL pg_partman extension
- Enabling the pg_partman extension
- Configuring partitions using the create_parent function
- Configuring partition maintenance using the run_maintenance_proc function

Overview of the PostgreSQL pg_partman extension

You can use the PostgreSQL `pg_partman` extension to automate the creation and maintenance of table partitions. For more general information, see PG Partition Manager in the `pg_partman` documentation.

> **Note**
>
> The `pg_partman` extension is supported on RDS for PostgreSQL versions 12.5 and higher.

Instead of having to manually create each partition, you configure `pg_partman` with the following settings:

- Table to be partitioned
- Partition type
- Partition key
- Partition granularity
- Partition precreation and management options

After you create a PostgreSQL partitioned table, you register it with `pg_partman` by calling the `create_parent` function. Doing this creates the necessary partitions based on the parameters you pass to the function.

The pg_partman extension also provides the run_maintenance_proc function, which you can call on a scheduled basis to automatically manage partitions. To ensure that the proper partitions are created as needed, schedule this function to run periodically (such as hourly). You can also ensure that partitions are automatically dropped.

Enabling the pg_partman extension

If you have multiple databases inside the same PostgreSQL DB instance for which you want to manage partitions, enable the pg_partman extension separately for each database. To enable the pg_partman extension for a specific database, create the partition maintenance schema and then create the pg_partman extension as follows.

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Note**
>
> To create the pg_partman extension, make sure that you have rds_superuser privileges.

If you receive an error such as the following, grant the rds_superuser privileges to the account or use your superuser account.

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

To grant rds_superuser privileges, connect with your superuser account and run the following command.

```
GRANT rds_superuser TO user-or-role;
```

For the examples that show using the pg_partman extension, we use the following sample database table and partition. This database uses a partitioned table based on a timestamp. A schema data_mart contains a table named events with a column named created_at. The following settings are included in the events table:

- Primary keys event_id and created_at, which must have the column used to guide the partition.

- A check constraint `ck_valid_operation` to enforce values for an `operation` table column.
- Two foreign keys, where one (`fk_orga_membership`) points to the external table `organization` and the other (`fk_parent_event_id`) is a self-referenced foreign key.
- Two indexes, where one (`idx_org_id`) is for the foreign key and the other (`idx_event_type`) is for the event type.

The following DDL statements create these objects, which are automatically included on each partition.

```
CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization ( org_id BIGSERIAL,
        org_name TEXT,
        CONSTRAINT pk_organization PRIMARY KEY (org_id)
    );

CREATE TABLE data_mart.events(
        event_id        BIGSERIAL,
        operation       CHAR(1),
        value           FLOAT(24),
        parent_event_id BIGINT,
        event_type      VARCHAR(25),
        org_id          BIGSERIAL,
        created_at      timestamp,
        CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
        CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
        CONSTRAINT fk_orga_membership
            FOREIGN KEY(org_id)
            REFERENCES data_mart.organization (org_id),
        CONSTRAINT fk_parent_event_id
            FOREIGN KEY(parent_event_id, created_at)
            REFERENCES data_mart.events (event_id,created_at)
    ) PARTITION BY RANGE (created_at);

CREATE INDEX idx_org_id     ON  data_mart.events(org_id);
CREATE INDEX idx_event_type ON  data_mart.events(event_type);
```

Configuring partitions using the create_parent function

After you enable the `pg_partman` extension, use the `create_parent` function to configure partitions inside the partition maintenance schema. The following example uses the `events` table

example created in Enabling the pg_partman extension. Call the `create_parent` function as follows.

```
SELECT partman.create_parent(
 p_parent_table => 'data_mart.events',
 p_control      => 'created_at',
 p_type         => 'range',
 p_interval     => '1 day',
 p_premake      => 30);
```

The parameters are as follows:

- `p_parent_table` – The parent partitioned table. This table must already exist and be fully qualified, including the schema.
- `p_control` – The column on which the partitioning is to be based. The data type must be an integer or time-based.
- `p_type` – The type is either `'range'` or `'list'`.
- `p_interval` – The time interval or integer range for each partition. Example values include `1 day`, `1 hour`, and so on.
- `p_premake` – The number of partitions to create in advance to support new inserts.

For a complete description of the `create_parent` function, see Creation Functions in the `pg_partman` documentation.

Configuring partition maintenance using the run_maintenance_proc function

You can run partition maintenance operations to automatically create new partitions, detach partitions, or remove old partitions. Partition maintenance relies on the `run_maintenance_proc` function of the `pg_partman` extension and the `pg_cron` extension, which initiates an internal scheduler. The `pg_cron` scheduler automatically executes SQL statements, functions, and procedures defined in your databases.

The following example uses the `events` table example created in Enabling the pg_partman extension to set partition maintenance operations to run automatically. As a prerequisite, add `pg_cron` to the `shared_preload_libraries` parameter in the DB instance's parameter group.

```
CREATE EXTENSION pg_cron;
```

```
UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

Following, you can find a step-by-step explanation of the preceding example:

1. Modify the parameter group associated with your DB instance and add `pg_cron` to the `shared_preload_libraries` parameter value. This change requires a DB instance restart for it to take effect. For more information, see Modifying parameters in a DB parameter group in Amazon RDS.
2. Run the command `CREATE EXTENSION pg_cron;` using an account that has the `rds_superuser` permissions. Doing this enables the `pg_cron` extension. For more information, see Scheduling maintenance with the PostgreSQL pg_cron extension.
3. Run the command `UPDATE partman.part_config` to adjust the `pg_partman` settings for the `data_mart.events` table.
4. Run the command `SET ...` to configure the `data_mart.events` table, with these clauses:
   a. `infinite_time_partitions = true,` – Configures the table to be able to automatically create new partitions without any limit.
   b. `retention = '3 months',` – Configures the table to have a maximum retention of three months.
   c. `retention_keep_table=true` – Configures the table so that when the retention period is due, the table isn't deleted automatically. Instead, partitions that are older than the retention period are only detached from the parent table.
5. Run the command `SELECT cron.schedule...` to make a `pg_cron` function call. This call defines how often the scheduler runs the `pg_partman` maintenance procedure, `partman.run_maintenance_proc`. For this example, the procedure runs every hour.

For a complete description of the `run_maintenance_proc` function, see Maintenance Functions in the `pg_partman` documentation.

Using pgAudit to log database activity

Financial institutions, government agencies, and many industries need to keep *audit logs* to meet regulatory requirements. By using the PostgreSQL Audit extension (pgAudit) with your RDS for PostgreSQL DB instance, you can capture the detailed records that are typically needed by auditors or to meet regulatory requirements. For example, you can set up the pgAudit extension to track changes made to specific databases and tables, to record the user who made the change, and many other details.

The pgAudit extension builds on the functionality of the native PostgreSQL logging infrastructure by extending the log messages with more detail. In other words, you use the same approach to view your audit log as you do to view any log messages. For more information about PostgreSQL logging, see RDS for PostgreSQL database log files.

The pgAudit extension redacts sensitive data such as cleartext passwords from the logs. If your RDS for PostgreSQL DB instance is configured to log data manipulation language (DML) statements as detailed in Turning on query logging for your RDS for PostgreSQL DB instance, you can avoid the cleartext password issue by using the PostgreSQL Audit extension.

You can configure auditing on your database instances with a great degree of specificity. You can audit all databases and all users. Or, you can choose to audit only certain databases, users, and other objects. You can also explicitly exclude certain users and databases from being audited. For more information, see Excluding users or databases from audit logging.

Given the amount of detail that can be captured, we recommend that if you do use pgAudit, you monitor your storage consumption.

The pgAudit extension is supported on all available RDS for PostgreSQL versions. For a list of pgAudit versions supported by available RDS for PostgreSQL versions, see Extension versions for Amazon RDS for PostgreSQL in the *Amazon RDS for PostgreSQL Release Notes.*

Topics

- Setting up the pgAudit extension
- Auditing database objects
- Excluding users or databases from audit logging
- Reference for the pgAudit extension

Setting up the pgAudit extension

To set up the pgAudit extension on your RDS for PostgreSQL DB instance , you first add pgAudit to the shared libraries on the custom DB parameter group for your RDS for PostgreSQL DB instance. For information about creating a custom DB parameter group, see Parameter groups for Amazon RDS. Next, you install the pgAudit extension. Finally, you specify the databases and objects that you want to audit. The procedures in this section show you how. You can use the AWS Management Console or the AWS CLI.

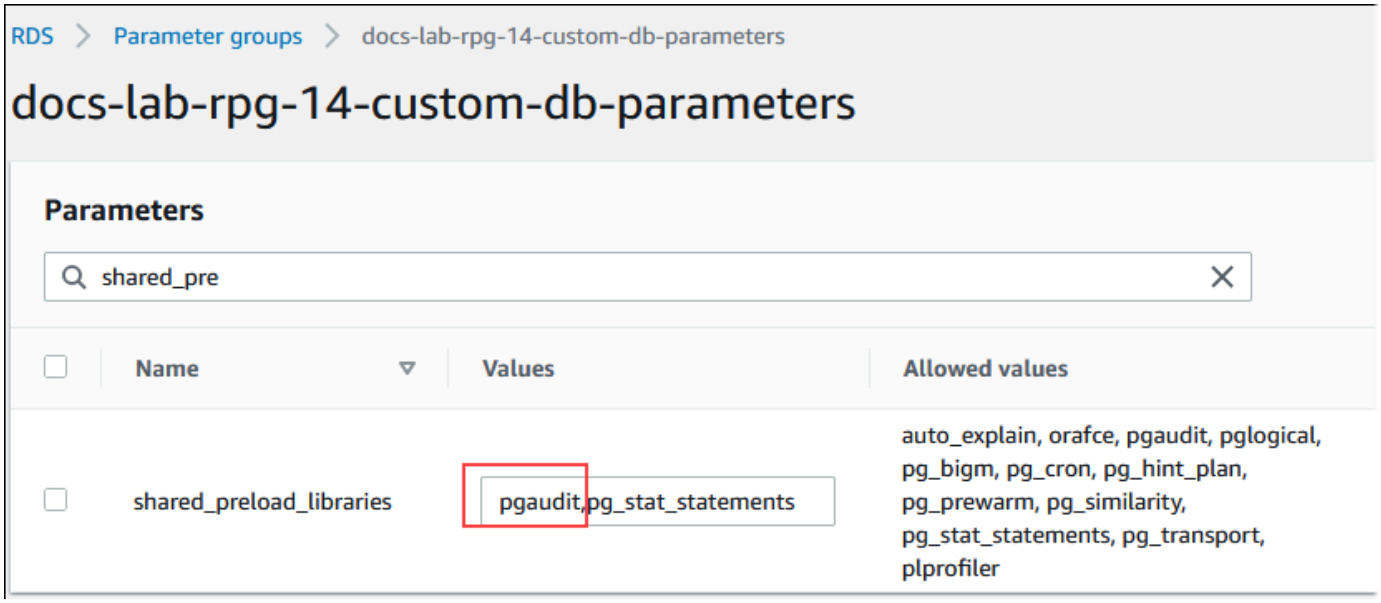
You must have permissions as the `rds_superuser` role to perform all these tasks.

The steps following assume that your RDS for PostgreSQL DB instance is associated with a custom DB parameter group.

Console

To set up the pgAudit extension

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose your RDS for PostgreSQL DB instance.
3. Open the **Configuration** tab for your RDS for PostgreSQL DB instance. Among the Instance details, find the **Parameter group** link.
4. Choose the link to open the custom parameters associated with your RDS for PostgreSQL DB instance.
5. In the **Parameters** search field, type `shared_pre` to find the `shared_preload_libraries` parameter.
6. Choose **Edit parameters** to access the property values.
7. Add `pgaudit` to the list in the **Values** field. Use a comma to separate items in the list of values.



8. Reboot the RDS for PostgreSQL DB instance so that your change to the `shared_preload_libraries` parameter takes effect.
9. When the instance is available, verify that pgAudit has been initialized. Use `psql` to connect to the RDS for PostgreSQL DB instance, and then run the following command.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

10. With pgAudit initialized, you can now create the extension. You need to create the extension after initializing the library because the `pgaudit` extension installs event triggers for auditing data definition language (DDL) statements.

```
CREATE EXTENSION pgaudit;
```

11. Close the `psql` session.

```
labdb=> \q
```

12. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

13. Find the `pgaudit.log` parameter in the list and set to the appropriate value for your use case. For example, setting the `pgaudit.log` parameter to `write` as shown in the following image captures inserts, updates, deletes, and some other types changes to the log.



You can also choose one of the following values for the `pgaudit.log` parameter.

- none – This is the default. No database changes are logged.
- all – Logs everything (read, write, function, role, ddl, misc).
- ddl – Logs all data definition language (DDL) statements that aren't included in the `ROLE` class.
- function – Logs function calls and `DO` blocks.
- misc – Logs miscellaneous commands, such as `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, and `SET`.
- read – Logs `SELECT` and `COPY` when the source is a relation (such as a table) or a query.
- role – Logs statements related to roles and privileges, such as `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, and `DROP ROLE`.
- write – Logs `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, and `COPY` when the destination is a relation (table).

14. Choose **Save changes**.
15. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
16. Choose your RDS for PostgreSQL DB instance from the Databases list.

AWS CLI

To setup pgAudit

To setup pgAudit using the AWS CLI, you call the modify-db-parameter-group operation to modify the audit log parameters in your custom parameter group, as shown in the following procedure.

1. Use the following AWS CLI command to add `pgaudit` to the `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pgaudit,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Use the following AWS CLI command to reboot the RDS for PostgreSQL DB instance so that the pgaudit library is initialized.

```
aws rds reboot-db-instance \
    --db-instance-identifier your-instance \
    --region aws-region
```

3. When the instance is available, you can verify that `pgaudit` has been initialized. Use `psql` to connect to the RDS for PostgreSQL DB instance, and then run the following command.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

   With pgAudit initialized, you can now create the extension.

```
CREATE EXTENSION pgaudit;
```

4. Close the `psql` session so that you can use the AWS CLI.

```
labdb=> \q
```

5. Use the following AWS CLI command to specify the classes of statement that want logged by session audit logging. The example sets the `pgaudit.log` parameter to `write`, which captures inserts, updates, and deletes to the log.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.log,ParameterValue=write,ApplyMethod=pending-reboot" \
   --region aws-region
```

   You can also choose one of the following values for the `pgaudit.log` parameter.

   - none – This is the default. No database changes are logged.
   - all – Logs everything (read, write, function, role, ddl, misc).
   - ddl – Logs all data definition language (DDL) statements that aren't included in the `ROLE` class.
   - function – Logs function calls and `DO` blocks.
   - misc – Logs miscellaneous commands, such as `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, and `SET`.
   - read – Logs `SELECT` and `COPY` when the source is a relation (such as a table) or a query.
   - role – Logs statements related to roles and privileges, such as `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, and `DROP ROLE`.
   - write – Logs `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, and `COPY` when the destination is a relation (table).

   Reboot the RDS for PostgreSQL DB instance using the following AWS CLI command.

```
aws rds reboot-db-instance \
    --db-instance-identifier your-instance \
    --region aws-region
```

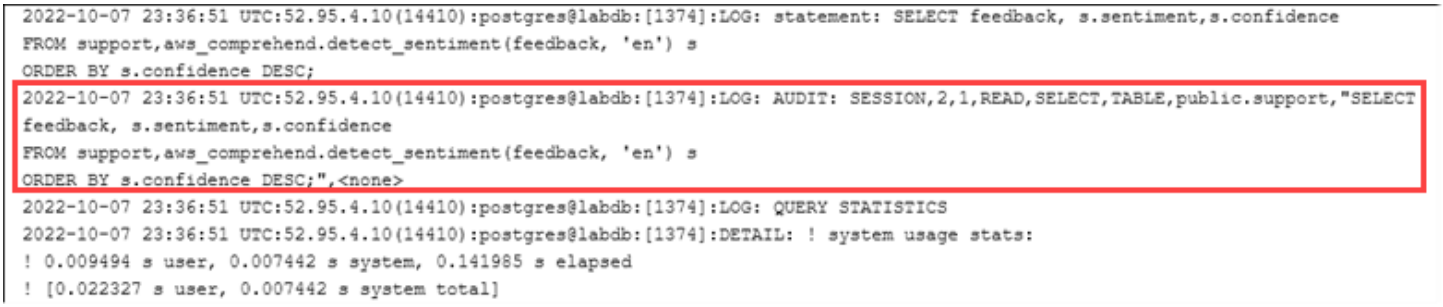
Auditing database objects

With pgAudit set up on your RDS for PostgreSQL DB instance and configured for your requirements, more detailed information is captured in the PostgreSQL log. For example, while the default PostgreSQL logging configuration identifies the date and time that a change was made in

a database table, with the pgAudit extension the log entry can include the schema, user who made the change, and other details depending on how the extension parameters are configured. You can set up auditing to track changes in the following ways.

- For each session, by user. For the session level, you can capture the fully qualified command text.
- For each object, by user and by database.

The object auditing capability is activated when you create the `rds_pgaudit` role on your system and then add this role to the `pgaudit.role` parameter in your custom parameter parameter group. By default, the `pgaudit.role` parameter is unset and the only allowable value is `rds_pgaudit`. The following steps assume that `pgaudit` has been initialized and that you have created the `pgaudit` extension by following the procedure in Setting up the pgAudit extension.



```
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: statement: SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: AUDIT: SESSION,2,1,READ,SELECT,TABLE,public.support,"SELECT
feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;",<none>
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: QUERY STATISTICS
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:DETAIL: ! system usage stats:
! 0.009494 s user, 0.007442 s system, 0.141985 s elapsed
! [0.022327 s user, 0.007442 s system total]
```

As shown in this example, the "LOG: AUDIT: SESSION" line provides information about the table and its schema, among other details.

To set up object auditing

1. Use `psql` to connect to the RDS for PostgreSQL DB instance.

   ```
   psql --host=your-instance-name.aws-region.rds.amazonaws.com --port=5432 --
   username=postgrespostgres --password --dbname=labdb
   ```

2. Create a database role named `rds_pgaudit` using the following command.

   ```
   labdb=> CREATE ROLE rds_pgaudit;
   CREATE ROLE
   labdb=>
   ```

3. Close the `psql` session.

   ```
   labdb=> \q
   ```

In the next few steps, use the AWS CLI to modify the audit log parameters in your custom parameter group.

4. Use the following AWS CLI command to set the `pgaudit.role` parameter to `rds_pgaudit`. By default, this parameter is empty, and `rds_pgaudit` is the only allowable value.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.role,ParameterValue=rds_pgaudit,ApplyMethod=pending-reboot"
 \
   --region aws-region
```

5. Use the following AWS CLI command to reboot the RDS for PostgreSQL DB instance so that your changes to the parameters take effect.

```
aws rds reboot-db-instance \
    --db-instance-identifier your-instance \
    --region aws-region
```

6. Run the following command to confirm that the `pgaudit.role` is set to `rds_pgaudit`.

```
SHOW pgaudit.role;
pgaudit.role
------------------
rds_pgaudit
```

To test pgAudit logging, you can run several example commands that you want to audit. For example, you might run the following commands.

```
CREATE TABLE t1 (id int);
GRANT SELECT ON t1 TO rds_pgaudit;
SELECT * FROM t1;
id
----
(0 rows)
```

The database logs should contain an entry similar to the following.

```
...
```

```
2017-06-12 19:09:49 UTC:...:rds_test@postgres:[11701]:LOG: AUDIT:
OBJECT,1,1,READ,SELECT,TABLE,public.t1,select * from t1;
...
```

For information on viewing the logs, see Monitoring Amazon RDS log files.

To learn more about the pgAudit extension, see pgAudit on GitHub.

Excluding users or databases from audit logging

As discussed in RDS for PostgreSQL database log files, PostgreSQL logs consume storage space. Using the pgAudit extension adds to the volume of data gathered in your logs to varying degrees, depending on the changes that you track. You might not need to audit every user or database in your RDS for PostgreSQL DB instance.

To minimize impacts to your storage and to avoid needlessly capturing audit records, you can exclude users and databases from being audited. You can also change logging within a given session. The following examples show you how.

> **Note**
>
> Parameter settings at the session level take precedence over the settings in the custom DB parameter group for the RDS for PostgreSQL DB instance. If you don't want database users to bypass your audit logging configuration settings, be sure to change their permissions.

Suppose that your RDS for PostgreSQL DB instance is configured to audit the same level of activity for all users and databases. You then decide that you don't want to audit the user `myuser`. You can turn off auditing for `myuser` with the following SQL command.

```
ALTER USER myuser SET pgaudit.log TO 'NONE';
```

Then, you can use the following query to check the `user_specific_settings` column for `pgaudit.log` to confirm that the parameter is set to `NONE`.

```
SELECT
    usename AS user_name,
    useconfig AS user_specific_settings
FROM
    pg_user
WHERE
```

```
    usename = 'myuser';
```

You see output such as the following.

```
 user_name | user_specific_settings
-----------+-----------------------
 myuser    | {pgaudit.log=NONE}
(1 row)
```

You can turn off logging for a given user in the midst of their session with the database with the following command.

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'none';
```

Use the following query to check the settings column for pgaudit.log for a specific user and database combination.

```
SELECT
    usename AS "user_name",
    datname AS "database_name",
    pg_catalog.array_to_string(setconfig, E'\n') AS "settings"
FROM
    pg_catalog.pg_db_role_setting s
    LEFT JOIN pg_catalog.pg_database d ON d.oid = setdatabase
    LEFT JOIN pg_catalog.pg_user r ON r.usesysid = setrole
WHERE
    usename = 'myuser'
    AND datname = 'mydatabase'
ORDER BY
    1,
    2;
```

You see output similar to the following.

```
  user_name | database_name |     settings
-----------+---------------+------------------
 myuser    | mydatabase    | pgaudit.log=none
(1 row)
```

After turning off auditing for `myuser`, you decide that you don't want to track changes to `mydatabase`. You turn off auditing for that specific database by using the following command.

```
ALTER DATABASE mydatabase SET pgaudit.log to 'NONE';
```

Then, use the following query to check the database_specific_settings column to confirm that pgaudit.log is set to NONE.

```
SELECT
a.datname AS database_name,
b.setconfig AS database_specific_settings
FROM
pg_database a
FULL JOIN pg_db_role_setting b ON a.oid = b.setdatabase
WHERE
a.datname = 'mydatabase';
```

You see output such as the following.

```
 database_name | database_specific_settings
---------------+----------------------------
 mydatabase    | {pgaudit.log=NONE}
(1 row)
```

To return settings to the default setting for myuser, use the following command:

```
ALTER USER myuser RESET pgaudit.log;
```

To return settings to their default setting for a database, use the following command.

```
ALTER DATABASE mydatabase RESET pgaudit.log;
```

To reset user and database to the default setting, use the following command.

```
ALTER USER myuser IN DATABASE mydatabase RESET pgaudit.log;
```

You can also capture specific events to the log by setting the `pgaudit.log` to one of the other allowed values for the `pgaudit.log` parameter. For more information, see List of allowable settings for the `pgaudit.log` parameter.

```
ALTER USER myuser SET pgaudit.log TO 'read';
ALTER DATABASE mydatabase SET pgaudit.log TO 'function';
```

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'read,function'
```

Reference for the pgAudit extension

You can specify the level of detail that you want for your audit log by changing one or more of the parameters listed in this section.

Controlling pgAudit behavior

You can control the audit logging by changing one or more of the parameters listed in the following table.

Parameter	Description
`pgaudit.log`	Specifies the statement classes that will be logged by session audit logging. Allowable values include ddl, function, misc, read, role, write, none, all. For more information, see List of allowable settings for the `pgaudit.log` parameter.
`pgaudit.log_catalog`	When turned on (set to 1), adds statements to audit trail if all relations in a statement are in pg_catalog.
`pgaudit.log_level`	Specifies the log level to use for log entries. Allowed values: debug5, debug4, debug3, debug2, debug1, info, notice, warning, log
`pgaudit.log_parameter`	When turned on (set to 1), parameters passed with the statement are captured in the audit log.
`pgaudit.log_relation`	When turned on (set to 1), the audit log for the session creates a separate log entry for each relation (TABLE, VIEW, and so on) referenced in a SELECT or DML statement.
`pgaudit.log_statement_once`	Specifies whether logging will include the statement text and parameters with the first log entry for a statement/substatement combination or with every entry.
`pgaudit.role`	Specifies the master role to use for object audit logging. The only allowable entry is `rds_pgaudit` .

List of allowable settings for the `pgaudit.log` parameter

Value	Description
none	This is the default. No database changes are logged.
all	Logs everything (read, write, function, role, ddl, misc).
ddl	Logs all data definition language (DDL) statements that aren't included in the `ROLE` class.
function	Logs function calls and `DO` blocks.
misc	Logs miscellaneous commands, such as `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, and `SET`.
read	Logs `SELECT` and `COPY` when the source is a relation (such as a table) or a query.
role	Logs statements related to roles and privileges, such as `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, and `DROP ROLE`.
write	Logs `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, and `COPY` when the destination is a relation (table).

To log multiple event types with session auditing, use a comma-separated list. To log all event types, set `pgaudit.log` to `ALL`. Reboot your DB instance to apply the changes.

With object auditing, you can refine audit logging to work with specific relations. For example, you can specify that you want audit logging for `READ` operations on one or more tables.

Scheduling maintenance with the PostgreSQL pg_cron extension

You can use the PostgreSQL `pg_cron` extension to schedule maintenance commands within a PostgreSQL database. For more information about the extension, see What is pg_cron? in the pg_cron documentation.

The `pg_cron` extension is supported on RDS for PostgreSQL engine versions 12.5 and higher.

To learn more about using `pg_cron`, see Schedule jobs with pg_cron on your RDS for PostgreSQL or your Aurora PostgreSQL-Compatible Edition databases.

> **Note**
>
> The `pg_cron` extension version is displayed as a two digit version, for example, 1.6, in the pg_available_extensions view. While you might see three digit versions, for example, 1.6.4 or 1.6.5, listed in some contexts, you must specify the two digit version when performing an extension upgrade.

Topics

- Setting up the pg_cron extension
- Granting database users permissions to use pg_cron
- Scheduling pg_cron jobs
- Reference for the pg_cron extension

Setting up the pg_cron extension

Set up the `pg_cron` extension as follows:

1. Modify the custom parameter group associated with your PostgreSQL DB instance by adding `pg_cron` to the `shared_preload_libraries` parameter value.
   - If your RDS for PostgreSQL DB instance uses the `rds.allowed_extensions` parameter to explicitly list extensions that can be installed, you need to add the `pg_cron` extension to the list. Only certain versions of RDS for PostgreSQL support the `rds.allowed_extensions` parameter. By default, all available extensions are allowed. For more information, see Restricting installation of PostgreSQL extensions.

Restart the PostgreSQL DB instance to have changes to the parameter group take effect. To learn more about working with parameter groups, see Modifying parameters in a DB parameter group in Amazon RDS.

2. After the PostgreSQL DB instance has restarted, run the following command using an account that has `rds_superuser` permissions. For example, if you used the default settings when you created your RDS for PostgreSQL DB instance, connect as user `postgres` and create the extension.

```
CREATE EXTENSION pg_cron;
```

   The `pg_cron` scheduler is set in the default PostgreSQL database named `postgres`. The `pg_cron` objects are created in this `postgres` database and all scheduling actions run in this database.

3. You can use the default settings, or you can schedule jobs to run in other databases within your PostgreSQL DB instance. To schedule jobs for other databases within your PostgreSQL DB instance, see the example in Scheduling a cron job for a database other than the default database.

Granting database users permissions to use pg_cron

Installing the `pg_cron` extension requires the `rds_superuser` privileges. However, permissions to use the `pg_cron` can be granted (by a member of the `rds_superuser` group/role) to other database users, so that they can schedule their own jobs. We recommend that you grant permissions to the `cron` schema only as needed if it improves operations in your production environment.

To grant a database user permission in the `cron` schema, run the following command:

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

This gives *db-user* permission to access the `cron` schema to schedule cron jobs for the objects that they have permissions to access. If the database user doesn't have permissions, the job fails after posting the error message to the `postgresql.log` file, as shown in the following:

```
2020-12-08 16:41:00 UTC::@:[30647]:ERROR: permission denied for table table-name
```

```
2020-12-08 16:41:00 UTC::@:[27071]:LOG: background worker "pg_cron" (PID 30647) exited
 with exit code 1
```

In other words, make sure that database users that are granted permissions on the `cron` schema also have permissions on the objects (tables, schemas, and so on) that they plan to schedule.

The details of the cron job and its success or failure are also captured in the `cron.job_run_details` table. For more information, see Tables for scheduling jobs and capturing status.

Scheduling pg_cron jobs

The following sections show how you can schedule various management tasks using `pg_cron` jobs.

> **Note**
>
> When you create `pg_cron` jobs, check that the `max_worker_processes` setting is larger than the number of `cron.max_running_jobs`. A `pg_cron` job fails if it runs out of background worker processes. The default number of `pg_cron` jobs is 5. For more information, see Parameters for managing the pg_cron extension.

Topics

- Vacuuming a table
- Purging the pg_cron history table
- Logging errors to the postgresql.log file only
- Scheduling a cron job for a database other than the default database

Vacuuming a table

Autovacuum handles vacuum maintenance for most cases. However, you might want to schedule a vacuum of a specific table at a time of your choosing.

See also, Working with PostgreSQL autovacuum on Amazon RDS for PostgreSQL.

Following is an example of using the `cron.schedule` function to set up a job to use `VACUUM FREEZE` on a specific table every day at 22:00 (GMT).

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
 schedule
----------
1
(1 row)
```

After the preceding example runs, you can check the history in the `cron.job_run_details` table as follows.

```
postgres=> SELECT * FROM cron.job_run_details;
jobid  | runid | job_pid | database | username | command                        |
 status    | return_message | start_time                    | end_time
-------+-------+---------+----------+----------+--------------------------------
+-----------+----------------+-------------------------------
+--------------------------------
 1     | 1     | 3395    | postgres | adminuser| vacuum freeze pgbench_accounts
 | succeeded | VACUUM         | 2020-12-04 21:10:00.050386+00 | 2020-12-04
 21:10:00.072028+00
(1 row)
```

Following is a query of the `cron.job_run_details` table to see the failed jobs.

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command                        | status
 | return_message                                | start_time                    |
 end_time
------+-------+---------+----------+----------+--------------------------------+--------
+-----------------------------------------------+-------------------------------
+-------------------------------
 5    | 4     | 30339   | postgres | adminuser| vacuum freeze pgbench_account | failed
 | ERROR: relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 |
 2020-12-04 21:48:00.029567+00
(1 row)
```

For more information, see Tables for scheduling jobs and capturing status.

Purging the pg_cron history table

The `cron.job_run_details` table contains a history of cron jobs that can become very large over time. We recommend that you schedule a job that purges this table. For example, keeping a week's worth of entries might be sufficient for troubleshooting purposes.

The following example uses the cron.schedule function to schedule a job that runs every day at midnight to purge the `cron.job_run_details` table. The job keeps only the last seven days. Use your `rds_superuser` account to schedule the job such as the following.

```
SELECT cron.schedule('0 0 * * *', $$DELETE
    FROM cron.job_run_details
    WHERE end_time < now() - interval '7 days'$$);
```

For more information, see Tables for scheduling jobs and capturing status.

Logging errors to the postgresql.log file only

To prevent writing to the `cron.job_run_details` table, modify the parameter group associated with the PostgreSQL DB instance and set the `cron.log_run` parameter to off. The `pg_cron` extension no longer writes to the table and captures errors to the `postgresql.log` file only. For more information, see Modifying parameters in a DB parameter group in Amazon RDS.

Use the following command to check the value of the `cron.log_run` parameter.

```
postgres=> SHOW cron.log_run;
```

For more information, see Parameters for managing the pg_cron extension.

Scheduling a cron job for a database other than the default database

The metadata for `pg_cron` is all held in the PostgreSQL default database named `postgres`. Because background workers are used for running the maintenance cron jobs, you can schedule a job in any of your databases within the PostgreSQL DB instance:

> **Note**
>
> Only users with `rds_superuser` role or `rds_superuser` privileges can list all cron jobs in the database. Other users can view only their own jobs in the `cron.job` table.

1. In the cron database, schedule the job as you normally do using the cron.schedule.

```
postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum
 freeze test_table');
```

2. As a user with the `rds_superuser` role, update the database column for the job that you just created so that it runs in another database within your PostgreSQL DB instance.

```
postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
```

3. Verify by querying the `cron.job` table.

```
postgres=> SELECT * FROM cron.job;
jobid | schedule    | command                        | nodename  | nodeport |
 database | username  | active | jobname
------+-------------+--------------------------------+-----------+----------
+----------+----------+--------+-----------------------
106   | 29 03 * * * | vacuum freeze test_table       | localhost | 8192     |
 database1| adminuser | t      | database1 manual vacuum
  1   | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192     |
 postgres | adminuser | t      | manual vacuum
(2 rows)
```

> **Note**
>
> In some situations, you might add a cron job that you intend to run on a different database. In such cases, the job might try to run in the default database (`postgres`) before you update the correct database column. If the user name has permissions, the job successfully runs in the default database.

Reference for the pg_cron extension

You can use the following parameters, functions, and tables with the `pg_cron` extension. For more information, see What is pg_cron? in the pg_cron documentation.

Topics

- Parameters for managing the pg_cron extension
- Function reference: cron.schedule
- Function reference: cron.unschedule
- Tables for scheduling jobs and capturing status

Parameters for managing the pg_cron extension

Following is a list of parameters that control the pg_cron extension behavior.

Parameter	Description
cron.database_name	The database in which pg_cron metadata is kept.
cron.host	The hostname to connect to PostgreSQL. You can't modify this value.
cron.log_run	Log every job that runs in the job_run_details table. Values are on or off. For more information, see Tables for scheduling jobs and capturing status.
cron.log_statement	Log all cron statements before running them. Values are on or off.
cron.max_running_jobs	The maximum number of jobs that can run concurrently.
cron.use_background_workers	Use background workers instead of client sessions. You can't modify this value.

Use the following SQL command to display these parameters and their values.

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%'
 ORDER BY name;
```

Function reference: cron.schedule

This function schedules a cron job. The job is initially scheduled in the default postgres database. The function returns a bigint value representing the job identifier. To schedule jobs to run in other databases within your PostgreSQL DB instance, see the example in Scheduling a cron job for a database other than the default database.

The function has two syntax formats.

Syntax

```
cron.schedule (job_name,
    schedule,
    command
);

cron.schedule (schedule,
    command
);
```

Parameters

Parameter	Description
job_name	The name of the cron job.
schedule	Text indicating the schedule for the cron job. The format is the standard cron format.
command	Text of the command to run.

Examples

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
 schedule
----------
      145
(1 row)

postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
 schedule
----------
      146
(1 row)
```

Function reference: cron.unschedule

This function deletes a cron job. You can specify either the `job_name` or the `job_id`. A policy makes sure that you are the owner to remove the schedule for the job. The function returns a Boolean indicating success or failure.

The function has the following syntax formats.

Syntax

```
cron.unschedule (job_id);

cron.unschedule (job_name);
```

Parameters

Parameter	Description
`job_id`	A job identifier that was returned from the `cron.schedule` function when the cron job was scheduled.
`job_name`	The name of a cron job that was scheduled with the `cron.schedule` function.

Examples

```
postgres=> SELECT cron.unschedule(108);
 unschedule
------------
 t
(1 row)

postgres=> SELECT cron.unschedule('test');
 unschedule
------------
 t
(1 row)
```

Tables for scheduling jobs and capturing status

The following tables are used to schedule the cron jobs and record how the jobs completed.

Table	Description
`cron.job`	Contains the metadata about each scheduled job. Most interactions with this table should be done by using the `cron.schedule` and `cron.unschedule` functions. **Important** We recommend that you don't give update or insert privileges directly to this table. Doing so would allow the user to update the `username` column to run as `rds-superuser` .
`cron.job_run_details`	Contains historic information about past scheduled jobs that ran. This is useful to investigate the status, return messages, and start and end time from the job that ran. **Note** To prevent this table from growing indefinitely, purge it on a regular basis. For an example, see Purging the pg_cron history table.

Using pglogical to synchronize data across instances

All currently available RDS for PostgreSQL versions support the `pglogical` extension. The pglogical extension predates the functionally similar logical replication feature that was introduced by PostgreSQL in version 10. For more information, see Performing logical replication for Amazon RDS for PostgreSQL.

The `pglogical` extension supports logical replication between two or more RDS for PostgreSQL DB instances. It also supports replication between different PostgreSQL versions, and between databases running on RDS for PostgreSQL DB instances and Aurora PostgreSQL DB clusters. The `pglogical` extension uses a publish-subscribe model to replicate changes to tables and other objects, such as sequences, from a publisher to a subscriber. It relies on a replication slot to ensure that changes are synchronized from a publisher node to a subscriber node, defined as follows.

- The *publisher node* is the RDS for PostgreSQL DB instance that's the source of data to be replicated to other nodes. The publisher node defines the tables to be replicated in a publication set.
- The *subscriber node* is the RDS for PostgreSQL DB instance that receives WAL updates from the publisher. The subscriber creates a subscription to connect to the publisher and get the decoded WAL data. When the subscriber creates the subscription, the replication slot is created on the publisher node.

Following, you can find information about setting up the `pglogical` extension.

Topics

- Requirements and limitations for the pglogical extension
- Setting up the pglogical extension
- Setting up logical replication for RDS for PostgreSQL DB instance
- Reestablishing logical replication after a major upgrade
- Managing logical replication slots for RDS for PostgreSQL
- Parameter reference for the pglogical extension

Requirements and limitations for the pglogical extension

All currently available releases of RDS for PostgreSQL support the `pglogical` extension.

Both the publisher node and the subscriber node must be set up for logical replication.

The tables that you want to replicate from a publisher to a subscriber must have the same names and the same schema. These tables must also contain the same columns, and the columns must use the same data types. Both publisher and subscriber tables must have the same primary keys. We recommend that you use only the PRIMARY KEY as the unique constraint.

The tables on the subscriber node can have more permissive constraints than those on the publisher node for CHECK constraints and NOT NULL constraints.

The `pglogical` extension provides features such as two-way replication that aren't supported by the logical replication feature built into PostgreSQL (version 10 and higher). For more information, see PostgreSQL bi-directional replication using pglogical.

Setting up the pglogical extension

To set up the `pglogical` extension on your RDS for PostgreSQL DB instance , you add `pglogical` to the shared libraries on the custom DB parameter group for your RDS for PostgreSQL DB instance. You also need to set the value of the `rds.logical_replication` parameter to `1`, to turn on logical decoding. Finally, you create the extension in the database. You can use the AWS Management Console or the AWS CLI for these tasks.

You must have permissions as the `rds_superuser` role to perform these tasks.

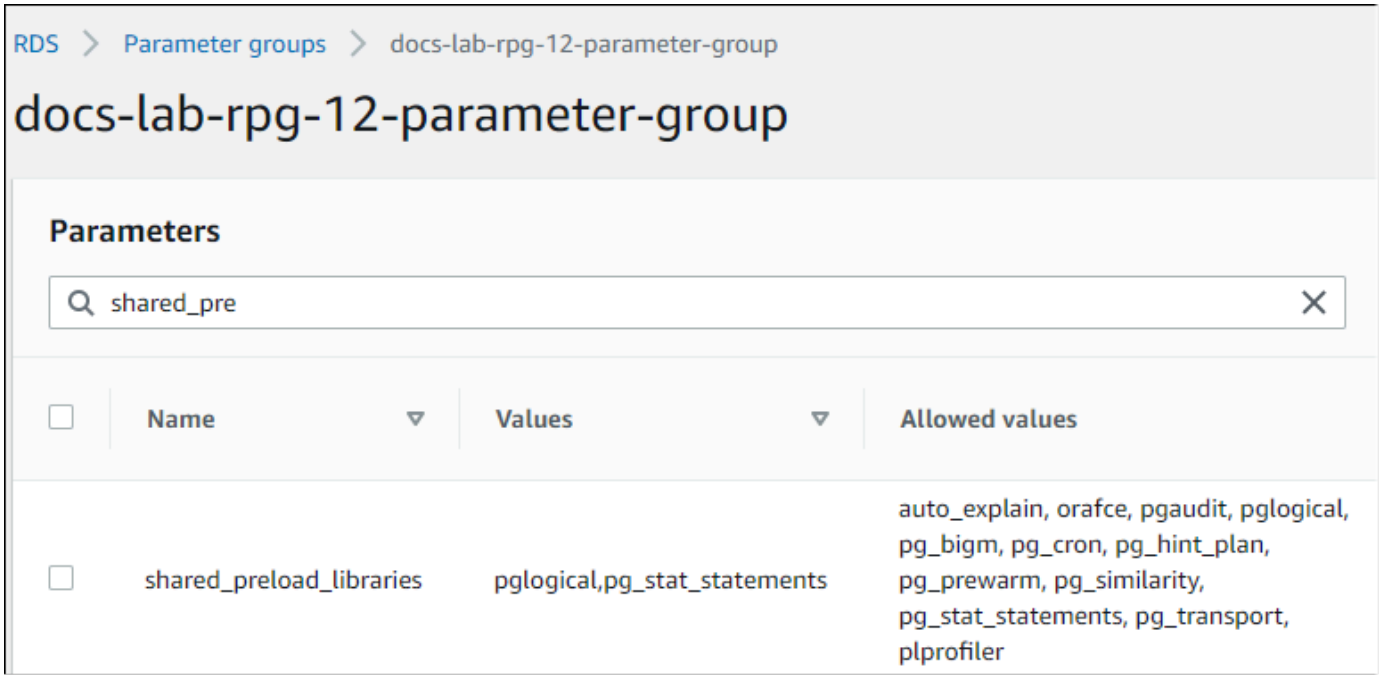
The steps following assume that your RDS for PostgreSQL DB instance is associated with a custom DB parameter group. For information about creating a custom DB parameter group, see Parameter groups for Amazon RDS.

Console

To set up the pglogical extension

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose your RDS for PostgreSQL DB instance.
3. Open the **Configuration** tab for your RDS for PostgreSQL DB instance. Among the Instance details, find the **Parameter group** link.
4. Choose the link to open the custom parameters associated with your RDS for PostgreSQL DB instance.

5. In the **Parameters** search field, type `shared_pre` to find the `shared_preload_libraries` parameter.
6. Choose **Edit parameters** to access the property values.
7. Add `pglogical` to the list in the **Values** field. Use a comma to separate items in the list of values.



8. Find the `rds.logical_replication` parameter and set it to 1, to turn on logical replication.
9. Reboot the RDS for PostgreSQL DB instance so that your changes take effect.
10. When the instance is available, you can use `psql` (or pgAdmin) to connect to the RDS for PostgreSQL DB instance.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

11. To verify that pglogical is initialized, run the following command.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pglogical
(1 row)
```

12. Verify the setting that enables logical decoding, as follows.

```
SHOW wal_level;
wal_level
-----------
 logical
(1 row)
```

13. Create the extension, as follows.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

14. Choose **Save changes**.
15. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
16. Choose your RDS for PostgreSQL DB instance from the Databases list to select it, and then choose **Reboot** from the Actions menu.

AWS CLI

To setup the pglogical extension

To setup pglogical using the AWS CLI, you call the modify-db-parameter-group operation to modify certain parameters in your custom parameter group as shown in the following procedure.

1. Use the following AWS CLI command to add `pglogical` to the `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pglogical,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Use the following AWS CLI command to set `rds.logical_replication` to 1 to turn on the logical decoding capability for the RDS for PostgreSQL DB instance.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
```

```
   --parameters
 "ParameterName=rds.logical_replication,ParameterValue=1,ApplyMethod=pending-
reboot" \
   --region aws-region
```

3. Use the following AWS CLI command to reboot the RDS for PostgreSQL DB instance so that the pglogical library is initialized.

```
aws rds reboot-db-instance \
    --db-instance-identifier your-instance \
    --region aws-region
```

4. When the instance is available, use `psql` to connect to the RDS for PostgreSQL DB instance.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

5. Create the extension, as follows.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

6. Reboot the RDS for PostgreSQL DB instance using the following AWS CLI command.

```
aws rds reboot-db-instance \
    --db-instance-identifier your-instance \
    --region aws-region
```

Setting up logical replication for RDS for PostgreSQL DB instance

The following procedure shows you how to start logical replication between two RDS for PostgreSQL DB instances. The steps assume that both the source (publisher) and the target (subscriber) have the `pglogical` extension set up as detailed in Setting up the pglogical extension.

> **Note**
>
> The node_name of a subscriber node can't start with `rds`.

To create the publisher node and define the tables to replicate

These steps assume that your RDS for PostgreSQL DB instance has a database that has one or more tables that you want to replicate to another node. You need to recreate the table structure from the publisher on the subscriber, so first, get the table structure if necessary. You can do that by using the `psql` metacommand `\d` *`tablename`* and then creating the same table on the subscriber instance. The following procedure creates an example table on the publisher (source) for demonstration purposes.

1. Use `psql` to connect to the instance that has the table you want to use as a source for subscribers.

   ```
   psql --host=source-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password --dbname=labdb
   ```

   If you don't have an existing table that you want to replicate, you can create a sample table as follows.

   a. Create an example table using the following SQL statement.

      ```
      CREATE TABLE docs_lab_table (a int PRIMARY KEY);
      ```

   b. Populate the table with generated data by using the following SQL statement.

      ```
      INSERT INTO docs_lab_table VALUES (generate_series(1,5000));
      INSERT 0 5000
      ```

   c. Verify that data exists in the table by using the following SQL statement.

      ```
      SELECT count(*) FROM docs_lab_table;
      ```

2. Identify this RDS for PostgreSQL DB instance as the publisher node, as follows.

   ```
   SELECT pglogical.create_node(
       node_name := 'docs_lab_provider',
       dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
    dbname=labdb');
    create_node
   -------------
      3410995529
   ```

```
(1 row)
```

3. Add the table that you want to replicate to the default replication set. For more information about replication sets, see Replication sets in the pglogical documentation.

```
SELECT pglogical.replication_set_add_table('default', 'docs_lab_table', 'true',
 NULL, NULL);
 replication_set_add_table
  --------------------------
  t
  (1 row)
```

The publisher node setup is complete. You can now set up the subscriber node to receive the updates from the publisher.

To set up the subscriber node and create a subscription to receive updates

These steps assume that the RDS for PostgreSQL DB instance has been set up with the `pglogical` extension. For more information, see Setting up the pglogical extension.

1. Use `psql` to connect to the instance that you want to receive updates from the publisher.

```
psql --host=target-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. On the subscriber RDS for PostgreSQL DB instance,create the same table that exists on the publisher. For this example, the table is `docs_lab_table`. You can create the table as follows.

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

3. Verify that this table is empty.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  0
(1 row)
```

4. Identify this RDS for PostgreSQL DB instance as the subscriber node, as follows.

```
SELECT pglogical.create_node(
```

```
    node_name := 'docs_lab_target',
    dsn := 'host=target-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=********');
 create_node
-------------
   2182738256
(1 row)
```

5. Create the subscription.

```
SELECT pglogical.create_subscription(
   subscription_name := 'docs_lab_subscription',
   provider_dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=*******',
   replication_sets := ARRAY['default'],
   synchronize_data := true,
   forward_origins := '{}' );
 create_subscription
---------------------
1038357190
(1 row)
```

When you complete this step, the data from the table on the publisher is created in the table on the subscriber. You can verify that this has occurred by using the following SQL query.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  5000
(1 row)
```

From this point forward, changes made to the table on the publisher are replicated to the table on the subscriber.

Reestablishing logical replication after a major upgrade

Before you can perform a major version upgrade of an RDS for PostgreSQL DB instance that's set up as a publisher node for logical replication, you must drop all replication slots, even those that aren't active. We recommend that you temporarily divert database transactions from the

publisher node, drop the replication slots, upgrade the RDS for PostgreSQL DB instance, and then re-establish and restart replication.

The replication slots are hosted on the publisher node only. The RDS for PostgreSQL subscriber node in a logical replication scenario has no slots to drops, but it can't be upgraded to a major version while it's designated as a subscriber node with a subscription to the publisher. Before upgrading the RDS for PostgreSQL subscriber node, drop the subscription and the node. For more information, see Managing logical replication slots for RDS for PostgreSQL.

Determining that logical replication has been disrupted

You can determine that the replication process has been disrupted by querying either the publisher node or the subscriber node, as follows.

To check the publisher node

- Use `psql` to connect to the publisher node, and then query the `pg_replication_slots` function. Note the value in the active column. Normally, this will return `t` (true), showing that replication is active. If the query returns `f` (false), it's an indication that replication to the subscriber has stopped.

```
SELECT slot_name,plugin,slot_type,active FROM pg_replication_slots;
                    slot_name              |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_labcb4fa94_docs_lab3de412c | pglogical_output | logical   | f
(1 row)
```

To check the subscriber node

On the subscriber node, you can check the status of replication in three different ways.

- Look through the PostgreSQL logs on the subscriber node to find failure messages. The log identifies failure with messages that include exit code 1, as shown following.

```
2022-07-06 16:17:03 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 14610) exited with exit code 1
2022-07-06 16:19:44 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 21783) exited with exit code 1
```

- Query the `pg_replication_origin` function. Connect to the database on the subscriber node using `psql` and query the `pg_replication_origin` function, as follows.

```
SELECT * FROM pg_replication_origin;
 roident | roname
---------+--------
(0 rows)
```

  The empty result set means that replication has been disrupted. Normally, you see output such as the following.

```
   roident |                       roname
  ---------+-------------------------------------------------
         1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
  (1 row)
```

- Query the `pglogical.show_subscription_status` function as shown in the following example.

```
SELECT subscription_name,status,slot_name FROM pglogical.show_subscription_status();
     subscription_name | status |              slot_name
---====----------------+--------+-------------------------------------
 docs_lab_subscription | down   | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

  This output shows that replication has been disrupted. Its status is `down`. Normally, the output shows the status as `replicating`.

If your logical replication process has been disrupted, you can re-establish replication by following these steps.

To reestablish logical replication between publisher and subscriber nodes

To re-establish replication, you first disconnect the subscriber from the publisher node and then re-establish the subscription, as outlined in these steps.

1. Connect to the subscriber node using `psql` as follows.

```
psql --host=222222222222.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. Deactivate the subscription by using the `pglogical.alter_subscription_disable` function.

```
SELECT pglogical.alter_subscription_disable('docs_lab_subscription',true);
 alter_subscription_disable
----------------------------
 t
(1 row)
```

3. Get the publisher node's identifier by querying the `pg_replication_origin`, as follows.

```
SELECT * FROM pg_replication_origin;
 roident |               roname
---------+-------------------------------------
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

4. Use the response from the previous step with the `pg_replication_origin_create` command to assign the identifier that can be used by the subscription when re-established.

```
SELECT pg_replication_origin_create('pgl_labdb_docs_labcb4fa94_docs_lab3de412c');
  pg_replication_origin_create
------------------------------
                             1
(1 row)
```

5. Turn on the subscription by passing its name with a status of `true`, as shown in the following example.

```
SELECT pglogical.alter_subscription_enable('docs_lab_subscription',true);
  alter_subscription_enable
---------------------------
 t
(1 row)
```

Check the status of the node. Its status should be `replicating` as shown in this example.

```
SELECT subscription_name,status,slot_name
  FROM pglogical.show_subscription_status();
            subscription_name |   status    |              slot_name
------------------------------+-------------+-------------------------------------
 docs_lab_subscription         | replicating |
 pgl_labdb_docs_lab98f517b_docs_lab3de412c
(1 row)
```

Check the status of the subscriber's replication slot on the publisher node. The slot's `active` column should return `t` (true), indicating that replication has been re-established.

```
SELECT slot_name,plugin,slot_type,active
  FROM pg_replication_slots;
                   slot_name              |      plugin      | slot_type | active
------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_lab98f517b_docs_lab3de412c | pglogical_output | logical   | t
(1 row)
```

Managing logical replication slots for RDS for PostgreSQL

Before you can perform a major version upgrade on an RDS for PostgreSQL DB instance that's serving as a publisher node in a logical replication scenario, you must drop the replication slots on the instance. The major version upgrade pre-check process notifies you that the upgrade can't proceed until the slots are dropped.

To drop slots from your RDS for PostgreSQL DB instance, first drop the subscription and then drop the slot.

To identify replication slots that were created using the `pglogical` extension, log in to each database and get the name of the nodes. When you query the subscriber node, you get both the publisher and the subscriber nodes in the output, as shown in this example.

```
SELECT * FROM pglogical.node;
node_id   |     node_name
------------+-------------------
 2182738256 | docs_lab_target
 3410995529 | docs_lab_provider
(2 rows)
```

You can get the details about the subscription with the following query.

```
SELECT sub_name,sub_slot_name,sub_target
  FROM pglogical.subscription;
 sub_name |         sub_slot_name          | sub_target
----------+--------------------------------+------------
  docs_lab_subscription     | pgl_labdb_docs_labcb4fa94_docs_lab3de412c | 2182738256
(1 row)
```

You can now drop the subscription, as follows.

```
SELECT pglogical.drop_subscription(subscription_name := 'docs_lab_subscription');
 drop_subscription
-------------------
                 1
(1 row)
```

After dropping the subscription, you can delete the node.

```
SELECT pglogical.drop_node(node_name := 'docs-lab-subscriber');
 drop_node
-----------
 t
(1 row)
```

You can verify that the node no longer exists, as follows.

```
SELECT * FROM pglogical.node;
 node_id | node_name
---------+-----------
(0 rows)
```

Parameter reference for the pglogical extension

In the table you can find parameters associated with the `pglogical` extension. Parameters such as `pglogical.conflict_log_level` and `pglogical.conflict_resolution` are used to handle update conflicts. Conflicts can emerge when changes are made locally to the same tables that are subscribed to changes from the publisher. Conflicts can also occur during various scenarios, such as two-way replication or when multiple subscribers are replicating from the same publisher. For more information, see PostgreSQL bi-directional replication using pglogical.

Parameter	Description
pglogical.batch_inserts	Batch inserts if possible. Not set by default. Change to '1' to turn on, '0' to turn off.
pglogical.conflict_log_level	Sets the log level to use for logging resolved conflicts. Supported string values are debug5, debug4, debug3, debug2, debug1, info, notice, warning, error, log, fatal, panic.
pglogical.conflict_resolution	Sets method to use to resolve conflicts when conflicts are resolvable. Supported string values are error, apply_remote, keep_local, last_update_wins, first_update_wins.
pglogical.extra_connection_options	Connection options to add to all peer node connections.
pglogical.synchronous_commit	pglogical specific synchronous commit value
pglogical.use_spi	Use SPI (server programming interface) instead of low-level API to apply changes. Set to '1' to turn on, '0' to turn off. For more information about SPI, see Server Programming Interface in the PostgreSQL documentation.

Using pgactive to support active-active replication

The `pgactive` extension uses active-active replication to support and coordinate write operations on multiple RDS for PostgreSQL databases. Amazon RDS for PostgreSQL supports the `pgactive` extension on the following versions:

- RDS for PostgreSQL 17.0 and all higher versions
- RDS for PostgreSQL 16.1 and higher 16 versions
- RDS for PostgreSQL 15.4-R2 and higher 15 versions
- RDS for PostgreSQL 14.10 and higher 14 versions
- RDS for PostgreSQL 13.13 and higher 13 versions
- RDS for PostgreSQL 12.17 and higher 12 versions
- RDS for PostgreSQL 11.22

> **Note**
>
> When there are write operations on more than one database in a replication configuration, conflicts are possible. For more information, see Handling conflicts in active-active replication

Topics

- Limitations for the pgactive extension
- Initializing the pgactive extension capability
- Setting up active-active replication for RDS for PostgreSQL DB instances
- Measuring replication lag among pgactive members
- Configuring parameter settings for the pgactive extension
- Understanding active-active conflicts
- Understanding the pgactive schema
- pgactive functions reference
- Handling conflicts in active-active replication
- Handling sequences in active-active replication

Limitations for the pgactive extension

- All tables require a Primary Key, otherwise Update's and Delete's aren't allowed. The values in the Primary Key column shouldn't be updated.
- Sequences may have gaps and sometimes might not follow an order. Sequences are not replicated. For more information, see Handling sequences in active-active replication.
- DDL and large objects are not replicated.
- Secondary unique indexes can cause data divergence.
- Collation needs to be identical on all node in the group.
- Load balancing across nodes is an anti-pattern.
- Large transactions can cause replication lag.

Initializing the pgactive extension capability

To initialize the `pgactive` extension capability on your RDS for PostgreSQL DB instance, set the value of the `rds.enable_pgactive` parameter to `1` and then create the extension in the database. Doing so automatically turns on the parameters `rds.logical_replication` and `track_commit_timestamp` and sets the value of `wal_level` to `logical`.

You must have permissions as the `rds_superuser` role to perform these tasks.

You can use the AWS Management Console or the AWS CLI to create the required RDS for PostgreSQL DB instances. The steps following assume that your RDS for PostgreSQL DB instance is associated with a custom DB parameter group. For information about creating a custom DB parameter group, see Parameter groups for Amazon RDS.

Console

To initialize the pgactive extension capability

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose your RDS for PostgreSQL DB instance.
3. Open the **Configuration** tab for your RDS for PostgreSQL DB instance. In the instance details, find the **DB instance parameter group** link.
4. Choose the link to open the custom parameters associated with your RDS for PostgreSQL DB instance.

5. Find the `rds.enable_pgactive` parameter, and set it to 1 to initialize the `pgactive` capability.
6. Choose **Save changes**.
7. In the navigation pane of the Amazon RDS console, choose **Databases**.
8. Select your RDS for PostgreSQL DB instance, and then choose **Reboot** from the **Actions** menu.
9. Confirm the DB instance reboot so that your changes take effect.
10. When the DB instance is available, you can use `psql` or any other PostgreSQL client to connect to the RDS for PostgreSQL DB instance.

    The following example assumes that your RDS for PostgreSQL DB instance has a default database named *`postgres`*.

    ```
    psql --host=mydb.111122223333.aws-region.rds.amazonaws.com --port=5432 --
    username=postgres --password=PASSWORD --dbname=postgres
    ```

11. To verify that pgactive is initialized, run the following command.

    ```
    postgres=>SELECT setting ~ 'pgactive'
    FROM pg_catalog.pg_settings
    WHERE name = 'shared_preload_libraries';
    ```

    If `pgactive` is in `shared_preload_libraries`, the preceding command will return the following:

    ```
    ?column?
    ----------
     t
    ```

AWS CLI

To initialize the pgactive extension capability

To initialize the `pgactive` using the AWS CLI, call the modify-db-parameter-group operation to modify certain parameters in your custom parameter group as shown in the following procedure.

1. Use the following AWS CLI command to set `rds.enable_pgactive` to 1 to initialize the `pgactive` capability for the RDS for PostgreSQL DB instance.

```
postgres=>aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=rds.enable_pgactive,ParameterValue=1,ApplyMethod=pending-reboot" \
   --region aws-region
```

2. Use the following AWS CLI command to reboot the RDS for PostgreSQL DB instance so that the `pgactive` library is initialized.

```
aws rds reboot-db-instance \
    --db-instance-identifier your-instance \
    --region aws-region
```

3. When the instance is available, use `psql` to connect to the RDS for PostgreSQL DB instance.

```
psql --host=mydb.111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=master user --password=PASSWORD --dbname=postgres
```

4. To verify that pgactive is initialized, run the following command.

```
postgres=>SELECT setting ~ 'pgactive'
FROM pg_catalog.pg_settings
WHERE name = 'shared_preload_libraries';
```

If `pgactive` is in `shared_preload_libraries`, the preceding command will return the following:

```
?column?
----------
 t
```

Setting up active-active replication for RDS for PostgreSQL DB instances

The following procedure shows you how to start active-active replication between two RDS for PostgreSQL DB instances where `pgactive` is available. To run the multi-region high availability example, you need to deploy Amazon RDS for PostgreSQL instances in two different regions and set up VPC Peering. For more information, see VPC peering.

> **Note**
>
> Sending traffic between multiple regions may incur additional costs.

These steps assume that the RDS for PostgreSQL DB instance has been enabled with the `pgactive` extension. For more information, see Initializing the pgactive extension capability.

To configure the first RDS for PostgreSQL DB instance with the `pgactive` extension

The following example illustrates how the `pgactive` group is created, along with other steps required to create the `pgactive` extension on the RDS for PostgreSQL DB instance.

1. Use `psql` or another client tool to connect to your first RDS for PostgreSQL DB instance.

```
psql --host=firstinstance.111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password=PASSWORD --dbname=postgres
```

2. Create a database on the RDS for PostgreSQL instance using the following command:

```
postgres=> CREATE DATABASE app;
```

3. Switch connection to the new database using the following command:

```
\c app
```

4. Create and populate a sample table using the following SQL statements:

   a. Create an example table using the following SQL statement.

```
app=> CREATE SCHEMA inventory;
CREATE TABLE inventory.products (
id int PRIMARY KEY, product_name text NOT NULL,
created_at timestamptz NOT NULL DEFAULT CURRENT_TIMESTAMP);
```

   b. Populate the table with some sample data by using the following SQL statement.

```
app=> INSERT INTO inventory.products (id, product_name)
VALUES (1, 'soap'), (2, 'shampoo'), (3, 'conditioner');
```

   c. Verify that data exists in the table by using the following SQL statement.

```
 app=>SELECT count(*) FROM inventory.products;

 count
-------
 3
```

5. Create `pgactive` extension on the existing database.

```
app=> CREATE EXTENSION pgactive;
```

6. To securely create and initialize the pgactive group use the following commands:

```
app=>
-- connection info for endpoint1
CREATE SERVER pgactive_server_endpoint1
    FOREIGN DATA WRAPPER pgactive_fdw
    OPTIONS (host '<endpoint1>', dbname 'app');
CREATE USER MAPPING FOR postgres
    SERVER pgactive_server_endpoint1
    OPTIONS (user 'postgres', password '<password>');
      -- connection info for endpoint2
CREATE SERVER pgactive_server_endpoint2
    FOREIGN DATA WRAPPER pgactive_fdw
    OPTIONS (host '<endpoint2>', dbname 'app');
CREATE USER MAPPING FOR postgres
    SERVER pgactive_server_endpoint2
    OPTIONS (user 'postgres', password '<password>');
```

Now you can initialize the replication group and add this first instance:

```
SELECT pgactive.pgactive_create_group(
    node_name := 'endpoint1-app',
    node_dsn := 'user_mapping=postgres
 pgactive_foreign_server=pgactive_server_endpoint1'

);
```

Use the following commands as an alternate but less secure method to create and initialize the pgactive group:

```
app=> SELECT pgactive.pgactive_create_group(
    node_name := 'node1-app',
    node_dsn := 'dbname=app host=firstinstance.111122223333.aws-
region.rds.amazonaws.com user=postgres password=PASSWORD');
```

node1-app is the name that you assign to uniquely identify a node in the `pgactive` group.

> **Note**
>
> To perform this step successfully on a DB instance that is publicly accessible, you must turn on the `rds.custom_dns_resolution` parameter by setting it to 1.

7. To check if the DB instance is ready, use the following command:

```
app=> SELECT pgactive.pgactive_wait_for_node_ready();
```

If the command succeeds, you can see the following output:

```
pgactive_wait_for_node_ready
-----------------------------
(1 row)
```

To configure the second RDS for PostgreSQL instance and join it to the `pgactive` group

The following example illustrates how you can join an RDS for PostgreSQL DB instance to the `pgactive` group, along with other steps that are required to create the `pgactive` extension on the DB instance.

These steps assume that another RDS for PostgreSQL DB instances has been set up with the `pgactive` extension. For more information, see Initializing the pgactive extension capability.

1. Use `psql` to connect to the instance that you want to receive updates from the publisher.

```
psql --host=secondinstance.111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password=PASSWORD --dbname=postgres
```

2. Create a database on the second RDS for PostgreSQL DB instance using the following command:

```
postgres=> CREATE DATABASE app;
```

3. Switch connection to the new database using the following command:

```
\c app
```

4. Create the `pgactive` extension on the existing database.

```
app=> CREATE EXTENSION pgactive;
```

5. Join the RDS for PostgreSQL second DB instance to the `pgactive` group in a more secure way using the following commands:

```

-- connection info for endpoint1
CREATE SERVER pgactive_server_endpoint1
    FOREIGN DATA WRAPPER pgactive_fdw
    OPTIONS (host '<endpoint1>', dbname 'app');
CREATE USER MAPPING FOR postgres
    SERVER pgactive_server_endpoint1
    OPTIONS (user 'postgres', password '<password>');

-- connection info for endpoint2
CREATE SERVER pgactive_server_endpoint2
    FOREIGN DATA WRAPPER pgactive_fdw
    OPTIONS (host '<endpoint2>', dbname 'app');
CREATE USER MAPPING FOR postgres
    SERVER pgactive_server_endpoint2
    OPTIONS (user 'postgres', password '<password>');

```

```

SELECT pgactive.pgactive_join_group(
    node_name := 'endpoint2-app',
```

```
    node_dsn := 'user_mapping=postgres
 pgactive_foreign_server=pgactive_server_endpoint2',
    join_using_dsn := 'user_mapping=postgres
 pgactive_foreign_server=pgactive_server_endpoint1'
);
```

Use the following commands as an alternate but less secure method to join the RDS for PostgreSQL second DB instance to the `pgactive` group

```
app=> SELECT pgactive.pgactive_join_group(
node_name := 'node2-app',
node_dsn := 'dbname=app host=secondinstance.111122223333.aws-
region.rds.amazonaws.com user=postgres password=PASSWORD',
join_using_dsn := 'dbname=app host=firstinstance.111122223333.aws-
region.rds.amazonaws.com user=postgres password=PASSWORD');
```

node2-app is the name that you assign to uniquely identify a node in the `pgactive` group.

6. To check if the DB instance is ready, use the following command:

```
app=> SELECT pgactive.pgactive_wait_for_node_ready();
```

If the command succeeds, you can see the following output:

```
pgactive_wait_for_node_ready
-----------------------------
(1 row)
```

If the first RDS for PostgreSQL database is relatively large, you can see `pgactive.pgactive_wait_for_node_ready()` emitting the progress report of the restore operation. The output looks similar to the following:

```
NOTICE:  restoring database 'app', 6% of 7483 MB complete
NOTICE:  restoring database 'app', 42% of 7483 MB complete
NOTICE:  restoring database 'app', 77% of 7483 MB complete
NOTICE:  restoring database 'app', 98% of 7483 MB complete
NOTICE:  successfully restored database 'app' from node node1-app in
 00:04:12.274956
 pgactive_wait_for_node_ready
```

```
------------------------------
(1 row)
```

From this point forward, `pgactive` synchronizes the data between the two DB instances.

7. You can use the following command to verify if the database of the second DB instance has the data:

```
app=> SELECT count(*) FROM inventory.products;
```

If the data is successfully synchronized, you'll see the following output:

```
 count
-------
 3
```

8. Run the following command to insert new values:

```
app=> INSERT INTO inventory.products (id, product_name) VALUES (4, 'lotion');
```

9. Connect to the database of the first DB instance and run the following query:

```
app=> SELECT count(*) FROM inventory.products;
```

If the active-active replication is initialized, the output is similar to the following:

```
count
-------
 4
```

To detach and remove a DB instance from the `pgactive` group

You can detach and remove a DB instance from the `pgactive` group using these steps:

1. You can detach the second DB instance from the first DB instance using the following command:

```
app=> SELECT * FROM pgactive.pgactive_detach_nodes(ARRAY['node2-app']);
```

2. Remove the pgactive extension from the second DB instance using the following command:

```
app=> SELECT * FROM pgactive.pgactive_remove();
```

   To forcefully remove the extension:

```
app=> SELECT * FROM pgactive.pgactive_remove(true);
```

3. Drop the extension using the following command:

```
app=> DROP EXTENSION pgactive;
```

Measuring replication lag among pgactive members

You can use the following query to view the replication lag among the pgactive members. Run this query on every pgactive node to get the full picture.

```
app=> SELECT * FROM pgactive.pgactive_get_replication_lag_info();
#-[ RECORD 1 ]--------+-----------------------------------------
#node_name            | node2-app
#node_sysid           | 7481018224801653637
#application_name     | pgactive:7481018224801653637:send
#slot_name            | pgactive_16385_7481018224801653637_0_16385__
#active               | t
#active_pid           | 783486
#pending_wal_decoding | 0
#pending_wal_to_apply | 0
#restart_lsn          | 0/2108150
#confirmed_flush_lsn  | 0/2154690
#sent_lsn             | 0/2154690
#write_lsn            | 0/2154690
#flush_lsn            | 0/2154690
#replay_lsn           | 0/2154690
#-[ RECORD 2 ]--------+-----------------------------------------
#node_name            | node1-app
#node_sysid           | 7481018033434600853
#application_name     | pgactive:7481018033434600853:send
#slot_name            | pgactive_16385_7481018033434600853_0_16385__
#active               | t
#active_pid           | 783488
```

```
#pending_wal_decoding | 0
#pending_wal_to_apply | 0
#restart_lsn          | 0/20F5AD0
#confirmed_flush_lsn  | 0/214EF68
#sent_lsn             | 0/214EF68
#write_lsn            | 0/214EF68
#flush_lsn            | 0/214EF68
#replay_lsn           | 0/214EF68
```

Monitor the following diagnostics at a minimum:

active

Set up alerts when active is false, which indicates that the slot isn't currently in use (the subscriber instance has disconnected from the publisher).

pending_wal_decoding

In PostgreSQL's logical replication, WAL files are stored in binary format. The publisher must decode these WAL changes and convert them into logical changes (such as insert, update, or delete operations).

The metric pending_wal_decoding shows the number of WAL files waiting to be decoded on the publisher side.

This number can increase due to these factors:

- When the subscriber isn't connected, active status will be false and pending_wal_decoding will increase
- The slot is active, but the publisher can't keep up with the volume of WAL changes

pending_wal_to_apply

The metric pending_wal_apply indicates the number of WAL files waiting to be applied on the subscriber side.

Several factors can prevent the subscriber from applying changes and potentially cause a disk full scenario:

- Schema differences - for example, when you have changes in the WAL stream for a table named sample, but that table doesn't exist on the subscriber side
- Values in the primary key columns were updated
- Secondary unique indexes can cause data divergence

Configuring parameter settings for the pgactive extension

You can use the following query to view all the parameters associated with `pgactive` extension.

```
app=> SELECT * FROM pg_settings WHERE name LIKE 'pgactive.%';
```

You can configure the `pgactive` extension using various parameters. These parameters can be set through either the AWS Management Console or the AWS CLI interface.

Main pgactive extension parameters

The following table provides a reference for the main parameters of the `pgactive` extension:

Parameter	Unit	Default	Description
`pgactive.conflict_logging_include_tuples`	`boolean`	–	Logs complete tuple information for the `pgactive` extension. **Note** A server restart is required for changes to take effect.
`pgactive.log_conflicts_to_table`	`boolean`	–	Determines whether the `pgactive` extension logs the detected conflicts to the `pgactive.`

Parameter	Unit	Default	Description
			`pgactive_conflict_history` table. For more information, see Conflict logging for details. **Note** A server restart is required for changes to take effect.

Parameter	Unit	Default	Description
pgactive.log_conflicts_to_logfile	boolean	–	Determines whether the pgactive extension logs the detected conflicts to the PostgreSQL log file. For more information, see Conflict logging for details. **Note** A server restart is required for changes to take effect.

Parameter	Unit	Default	Description
pgactive.synchronous_commit	boolean	off	Determines the commit behavior for pgactive apply workers. When disabled(off), apply workers perform asynchronous commits, which improves PostgreSQL throughput during apply operations but delays replay confirmations to the upstream. Setting it to off is always safe and won't cause transaction loss or skipping. This setting only affects the timing of disk flushes on the downstream node and when confirmations are sent upstream. The system delays sending replay flush

Parameter	Unit	Default	Description
			confirmations until commits are flushed to disk through unrelated operations like checkpoints or periodic work. However, if the upstream has the downstream listed in `synchronous_standby_names`, setting it to `off` causes synchronous commits on the upstream to take longer to report success to the client. In this case, set the parameter to on. **Note** Even when this parameter is set to on with nodes

Parameter	Unit	Default	Description
			listed in `synchronous_standby_names`, replication conflicts can still occur in active-active configurations. This is because the system lacks inter-node locking and global snapshot management, allowing concurrent transactions on different nodes to modify the same

Parameter	Unit	Default	Description
			tuple. Additionally, transactions only begin replication after committing on the upstream node. Enabling synchronous commit doesn't transform the pgactive extension into an always-consistent system.

Parameter	Unit	Default	Description
pgactive.temp_dump_directory	string	–	Defines the temporary storage path required for database cloning operations during initial setup. This directory must be writable by the postgres user and have sufficient storage space to contain a complete database dump. The system uses this location only during initial database setup with logical copy operations. This parameter isn't used by the pgactive_init_copy command.

Parameter	Unit	Default	Description
pgactive.max_ddl_lock_delay	milliseconds	-1	Specifies the maximum wait time for DDL lock before forcibly aborting concurrent write transactions. The default value is -1, which adopts the value set in max_standby_streaming_delay . This parameter accepts time units. For example, you can set it to 10s for 10 seconds. During this wait period, the system attempts to acquire DDL locks while waiting for ongoing write transactions to either commit or roll back. For more information, see the DDL Locking.

Parameter	Unit	Default	Description
pgactive.ddl_lock_timeout	milliseconds	-1	Specifies how long a DDL lock attempt waits to obtain the lock. The default value is -1, which uses the value specified in lock_timeout. You can set this parameter using time units such as 10s for 10 seconds. This timer only controls the waiting period for obtaining a DDL lock. Once the system obtains the lock and begins the DDL operation, the timer stops. This parameter doesn't limit the total duration a DDL lock can be held or the overall DDL operation time. To control the total duration of the operation,

Parameter	Unit	Default	Description
			use `statement_timeout` instead. For more information, see DDL Locking.

Parameter	Unit	Default	Description
pgactive.debug_trace_ddl_locks_level	boolean	–	Overrides the default debug log level for DDL locking operations in the pgactive extension. When configured, this setting causes DDL lock-related messages to be emitted at the LOG debug level instead of their default level. Use this parameter to monitor DDL locking activity without enabling the verbose DEBUG1 or DEBUG2 log levels across your entire server. Available log levels, in increasing order of verbosity: • **none** - DDL lock messages appear only at

Parameter	Unit	Default	Description
			DEBUG1 and lower server log levels. • **statement** - Adds LOG output for DDL lock acquisition attempts. • **acquire_release** - Records lock acquisition, release, declination events, and peer node applications of remote DDL locks. • **peers** - Provides additional details about DDL lock negotiations between peer nodes. • **debug** - Logs all DDL lock-related activities at LOG level.

Parameter	Unit	Default	Description
			For more information about monitoring options, see Monitoring global DDL locks. **Note** Changes to this setting take effect when you reload the configuration. You don't need to restart the server.

Additional pgactive extension parameters

The following table presents less frequently used and internal configuration options available for the `pgactive` extension.

Parameter	Unit	Default	Description
pgactive.debug_apply_delay	integer	–	Sets an apply delay (in milliseconds) for configured connections that don't have an explicit apply delay in their pgactive.pgactive_connections entry. This delay is set during node creation or join time, and pgactive won't replay a transaction on peer nodes until at least the specified number of milliseconds have elapsed since it was committed. Primarily used to simulate high-latency networks in testing environme

Parameter	Unit	Default	Description
			nts to make it easier to create conflicts. For example, with a 500ms delay on nodes A and B, you have at least 500ms to perform a conflicting insert on node B after inserting a value on node A. **Note** Requires a server reload or restart of apply workers to take effect.

Parameter	Unit	Default	Description
pgactive.connectability_check_duration	integer	–	Specifies the duration (in seconds) that a database worker attempts to establish connections during failed attempts. The worker makes one connection attempt per second until it succeeds or reaches this timeout value. This setting is useful when the database engine starts before the worker is ready to establish connections.

Parameter	Unit	Default	Description
pgactive.skip_ddl_replication	boolean	on	Controls how DDL changes are replicated or handled in Amazon RDS with pgactive enabled. When set to on, the node processes DDL changes like a non-pgcctive node. The following requirements apply when working with this parameter: • New nodes can't join a pgactive group if their skip_ddl_replication value differs from the upstream node. • Existing nodes can't start pgactive workers if their parameter

Parameter	Unit	Default	Description
			value doesn't match the upstream node. • All pgactive members must use the same parameter value. You can modify this parameter in two ways with super user privileges: globally, locally (session level). **Note** Changing this parameter incorrectly can break your replication setups.

Parameter	Unit	Default	Description
pgactive.do_not_replicate	boolean	–	This parameter is for internal use only. When you set this parameter in a transaction, the changes are not replicated to other nodes in your DB cluster. **Note** Changing this parameter incorrectly can break your replication setups.

Parameter	Unit	Default	Description
pgactive.discard_mismatched_row_attributes	boolean	–	This parameter is intended for specialist use only. We recommend using this parameter only when troubleshooting specific replication issues. Use this parameter when: • The incoming replication stream contains rows with more columns than your local table. • These remote rows contain non-null values. This setting overrides the following error message and allows data divergence to arise to

Parameter	Unit	Default	Description
			let replicati on continue: `cannot right-pad mismatched attributes; attno %u is missing in local table and remote row has non-null, non-dropp ed value for this attribute` **Note** Changing this parameter incorrect ly can break your replicati on setups.

Parameter	Unit	Default	Description
pgactive.debug_trace_replay	boolean	–	When set to on, it emits a log message for each remote action that downstream apply workers process. The logs include: • Change type • Affected table name • Number of changes since transaction start • Transaction commit LSN • Commit timestamp • Upstream node identifier • Forwarding node identifier (if applicable) The logs also capture queued DDL commands and table drops. para>

Parameter	Unit	Default	Description
			By default, the logs do not include row field contents. To include row values in the logs, you must recompile with the following flags enabled: • VERBOSE_INSERT • VERBOSE_UPDATE • VERBOSE_DELETE **Note** Enabling this logging setting can impact performance. We recommend enabling it only when needed for

Parameter	Unit	Default	Description
			troubleshooting. Changes to this setting take effect when you reload the configuration. You don't need to restart the server.

Parameter	Unit	Default	Description
pgactive.extra_apply_connection_options		–	You can configure connection parameters for all peer node connections with pgactive nodes. These parameters control settings such as keepalives and SSL modes. By default, pgactive uses the following connection parameters: • connect_timeout=30 • keepalives=1 • keepalives_idle=20 • keepalives_interval=20 • keepalives_count=5 To override the default parameters, use the following similar command:

Parameter	Unit	Default	Description
			`pgactive.extra_apply_connection_options = 'keepalives=0'` Individual node connection strings take precedence over both these settings and pgactive's built-in connection options. For more information about connection string formats, see libpq connection strings. We recommend keeping the default keepalive settings enabled. Only disable keepalives if you experience issues with

Parameter	Unit	Default	Description
			large transactions completing over unreliable networks. **Note** We recommend keeping the default keepalive settings enabled. Only disable keepalives if you experience issues with large transactions completing over unreliable networks. Changes to this setting take effect

Parameter	Unit	Default	Description
			when you reload the configuration. You don't need to restart the server.
`pgactive.init_node_parallel_jobs (int)`		–	Specifies the number of parallel jobs that `pg_dump` and `pg_restore` can use during logical node joins with the `pgactive.pgactive_join_group` function. Changes to this setting take effect when you reload the configuration. You don't need to restart the server.

Parameter	Unit	Default	Description
pgactive.max_nodes	int	4	Specifies the maximum number of nodes allowed in a pgactive extension group. The default value is 4 nodes. You must consider the following when setting the value of this parameter: • All nodes in a pgactive extension group must use the same parameter value. • A new node can't join if its parameter value differs from the upstream node. • Existing nodes can't start pgactive extension workers

Parameter	Unit	Default	Description
			if their parameter value differs from the upstream node. • Larger groups require additional monitoring and maintenance effort, so the value of this parameter wisely. You can set this parameter in two ways: in the configuration file, using the `ALTER SYSTEM SET` command Default value for this parameter is 4, meaning, there can be maximum of 4 nodes allowed in the `pgactive` extension group at any point of time.

Parameter	Unit	Default	Description
			Note The change takes effect after you restart the server.
`pgactive.permit_node_identifier_getter_function_creation`	`boolean`	–	This parameter is intended for internal use only. When enabled, `pgactive` extension allows creation of pgactive node identifier getter function.

Understanding active-active conflicts

When you use pgactive in active-active mode, writing to the same tables from multiple nodes can create data conflicts. While some clustering systems use distributed locks to prevent concurrent access, pgactive takes an optimistic approach that's better suited for geographically distributed applications.

Some database clustering systems prevent concurrent data access by using distributed locks. While this approach works when servers are in close proximity, it doesn't support geographically distributed applications because it requires extremely low latency for good performance. Instead

of using distributed locks (a pessimistic approach), the pgactive extension uses an optimistic approach. This means it:

- Helps you avoid conflicts when possible.
- Allows certain types of conflicts to occur.
- Provides conflict resolution when conflicts happen.

This approach gives you more flexibility when building distributed applications.

How conflicts happen

Inter-node conflicts arise from sequences of events that could not happen if all the involved transactions occurred concurrently on the same node. Because the nodes only exchange changes after transactions commit, each transaction is individually valid on the node it committed on but would not be valid if run on another node that has done other work in the meantime. Since pgactive apply essentially replays the transaction on the other nodes, the replay operation can fail if there is a conflict between a transaction being applied and a transaction that was committed on the receiving node.

The reason most conflicts can't happen when all transactions run on a single node is that PostgreSQL has inter-transaction communication mechanisms to prevent it, including:

- UNIQUE indexes
- SEQUENCEs
- Row and relation locking
- SERIALIZABLE dependency tracking

All of these mechanisms are ways to communicate between transactions to prevent undesirable concurrency issues

pgactive achieves low latency and handles network partitions well because it doesn't use a distributed transaction manager or lock manager. However, this means transactions on different nodes run in complete isolation from each other. While isolation typically improves database consistency, in this case, you need to reduce isolation to prevent conflicts.

Types of conflicts

Conflicts that can occur include:

Topics

- PRIMARY KEY or UNIQUE conflicts
- INSERT/INSERT conflicts
- INSERTs that violate multiple UNIQUE constraints
- UPDATE/UPDATE conflicts
- UPDATE conflicts on the PRIMARY KEY
- UPDATEs that violate multiple UNIQUE constraints
- UPDATE/DELETE conflicts
- INSERT/UPDATE conflicts
- DELETE/DELETE conflicts
- Foreign Key Constraint conflicts
- Exclusion constraint conflicts
- Global data conflicts
- Lock conflicts and deadlock aborts
- Divergent conflicts

PRIMARY KEY or UNIQUE conflicts

Row conflicts occur when multiple operations attempt to modify the same row key in ways not possible on a single node. These conflicts represent the most common type of data conflicts.

pgactive resolves detected conflicts through last-update-wins handling or your custom conflict handler.

Row conflicts include:

- INSERT vs INSERT
- INSERT vs UPDATE
- UPDATE vs DELETE
- INSERT vs DELETE
- DELETE vs DELETE
- INSERT vs DELETE

INSERT/INSERT conflicts

This most common conflict occurs when INSERTs on two different nodes create a tuple with the same PRIMARY KEY values (or identical UNIQUE constraint values when no PRIMARY KEY exists).

pgactivelink resolves INSERT conflicts by using the timestamp from the originating host to keep the most recent tuple. You can override this default behavior with your custom conflict handler. While this process requires no special administrator action, be aware that pgactivelink discards one of the INSERT operations across all nodes. No automatic data merging occurs unless your custom handler implements it.

The pgactivelink can only resolve conflicts involving a single constraint violation. If an INSERT violates multiple UNIQUE constraints, you must implement additional conflict resolution strategies.

INSERTs that violate multiple UNIQUE constraints

An INSERT/INSERT conflict can violate multiple UNIQUE constraints, including the PRIMARY KEY. pgactivelink can only handle conflicts that involve a single UNIQUE constraint. When conflicts violate multiple UNIQUE constraints, the apply worker fails and returns the following error:

```
multiple unique constraints violated by remotely INSERTed tuple.
```

In older versions, this situation generated a 'diverging uniqueness conflict' error instead.

To resolve these conflicts, you must take manual action. Either DELETE the conflicting local tuples or UPDATE them to remove conflicts with the new remote tuple. Be aware that you might need to address multiple conflicting tuples. Currently, pgactivelink provides no built-in functionality to ignore, discard, or merge tuples that violate multiple unique constraints.

> **Note**
>
> For more information, see UPDATEs that violate multiple UNIQUE constraints.

UPDATE/UPDATE conflicts

This conflict occurs when two nodes concurrently modify the same tuple without changing its PRIMARY KEY. pgactivelink resolves these conflicts using last-update-wins logic or your custom conflict handler, if defined. A PRIMARY KEY is essential for tuple matching and conflict resolution.

For tables without a PRIMARY KEY, pgactivelink rejects UPDATE operations with the following error:

```
Cannot run UPDATE or DELETE on table (tablename) because it does not have a primary key.
```

UPDATE conflicts on the PRIMARY KEY

pgactive has limitations when handling PRIMARY KEY updates. While you can perform UPDATE operation on a PRIMARY KEY, pgactive can't automatically resolve conflicts using last-update-wins logic for these operations. You must ensure that your PRIMARY KEY updates don't conflict with existing values. If conflicts occur during PRIMARY KEY updates, they become divergent conflicts that require your manual intervention. For more information about handling these situations, see Divergent conflicts.

UPDATEs that violate multiple UNIQUE constraints

pgactivelink cannot apply last-update-wins conflict resolution when an incoming UPDATE violates multiple UNIQUE constraints or PRIMARY KEY values. This behavior is similar to INSERT operations with multiple constraint violations. These situations create divergent conflicts that require your manual intervention. For more information, see Divergent conflicts.

UPDATE/DELETE conflicts

This conflicts occur when one node UPDATEs a row while another node simultaneously DELETEs it. In this case a UPDATE/DELETE conflict occurs on replay. The resolution is to discard any UPDATE that arrives after a DELETE, unless your custom conflict handler specifies otherwise.

pgactivelink requires a PRIMARY KEY to match tuples and resolve conflicts. For tables without a PRIMARY KEY, it rejects DELETE operations with the following error:

```
Cannot run UPDATE or DELETE on table (tablename) because it does not have a primary key.
```

> **Note**
>
> pgactivelink cannot distinguish between UPDATE/DELETE and INSERT/UPDATE conflicts. In both cases, an UPDATE affects a nonexistent row. Due to asynchronous replication and lack of replay ordering between nodes, pgactivelink can't determine if the UPDATE is for a new

row (INSERT not yet received) or a deleted row. In both scenarios, pgactivelink discards the UPDATE.

INSERT/UPDATE conflicts

This conflict can occur in multi-node environments. It happens when one node INSERTs a row, a second node UPDATEs it, and a third node receives the UPDATE before the original INSERT. By default, pgactivelink resolves these conflicts by discarding the UPDATE, unless your custom conflict trigger specifies otherwise. Be aware that this resolution method can result in data inconsistencies across nodes. For more information about similar scenarios and their handling, see UPDATE/DELETE conflicts.

DELETE/DELETE conflicts

This conflict occurs when two different nodes concurrently delete the same tuple. pgactivelink considers these conflicts harmless because both DELETE operations have the same end result. In this scenario, pgactivelink safely ignores one of the DELETE operations without affecting data consistency.

Foreign Key Constraint conflicts

FOREIGN KEY constraints can cause conflicts when applying remote transactions to existing local data. These conflicts typically occur when transactions are applied in a different sequence than their logical order on the originating nodes.

By default, pgactive applies changes with session_replication_role as `replica`, which bypasses foreign key checks during replication. In active-active configurations, this can lead to foreign key violations. Most violations are temporary and resolve once replication catches up. However, dangling foreign keys can occur because pgactive doesn't support cross-node row locking.

This behavior is inherent to partition-tolerant asynchronous active-active systems. For example, node A might insert a new child row while node B simultaneously deletes its parent row. The system can't prevent this type of concurrent modification across nodes.

To minimize foreign key conflicts, we recommend the following:

- Limit foreign key relationships to closely related entities.
- Modify related entities from a single node when possible.
- Choose entities that rarely require modification.

- Implement application-level concurrency control for modifications.

Exclusion constraint conflicts

pgactive link doesn't support exclusion constraints and restricts their creation.

> **Note**
>
> If you convert an existing standalone database to a pgactivelink database, manually drop all exclusion constraints.

In a distributed asynchronous system, it's not possible to guarantee that no set of rows violates the constraint. This is because all transactions on different nodes are fully isolated. Exclusion constraints can lead to replay deadlocks, where replay can't progress from any node to another due to exclusion constraint violations.

If you force pgactive Link to create an exclusion constraint, or if you don't drop existing ones when converting a standalone database to pgactive Link, replication is likely to break. To restore replication progress, remove or alter the local tuples that conflict with an incoming remote tuple so that the remote transaction can be applied.

Global data conflicts

When using pgactivelink, conflicts can occur when nodes have different global PostgreSQL system-wide data, such as roles. These conflicts can cause operations—primarily DDL—to succeed and commit on one node but fail to apply to other nodes.

If a user exists on one node but not another, replication issues can occur:

- Node1 has a user named `fred`, but this user doesn't exist on Node2
- When `fred` creates a table on Node1, the table is replicated with `fred` as the owner
- When this DDL command is applied to Node2, it fails because user `fred` doesn't exist
- This failure generates an ERROR in the PostgreSQL logs on Node2 and increments the `pgactive.pgactive_stats.nr_rollbacks` counter

Resolution: Create the user `fred` on Node2. The user doesn't need identical permissions but must exist on both nodes.

If a table exists on one node but not another, data modification operations will fail:

- Node1 has a table named `foo` that doesn't exist on Node2
- Any DML operations on the `foo` table on Node1 will fail when replicated to Node2

Resolution: Create the table `foo` on Node2 with the same structure.

> **Note**
>
> pgactivelink doesn't currently replicate CREATE USER commands or DDL operations. DDL replication is planned for a future release.

Lock conflicts and deadlock aborts

Because pgactive apply processes operate like normal user sessions, they follow standard row and table locking rules. This can result in pgactivelink apply processes waiting on locks held by user transactions or by other apply processes.

The following types of locks can affect apply processes:

- Explicit table-level locking (LOCK TABLE ...) by user sessions
- Explicit row-level locking (SELECT ... FOR UPDATE/FOR SHARE) by user sessions
- Locking from foreign keys
- Implicit locking due to row UPDATEs, INSERTs, or DELETEs, either from local activity or apply from other servers

Deadlocks can occur between:

- A pgactivelink apply process and a user transaction
- Two apply processes

When deadlocks occur, PostgreSQL's deadlock detector terminates one of the problem transactions. If the pgactivelink apply worker's process is terminated, it automatically retries and typically succeeds.

Note

- These issues are temporary and generally don't require administrator intervention. If an apply process is blocked for an extended period by a lock on an idle user session, you can terminate the user session to resume replication. This situation is similar to when a user holds a long lock that affects another user session.
- To identify locking-related replay delays, enable the `log_lock_waits` facility in PostgreSQL.

Divergent conflicts

Divergent conflicts occur when data that should be identical across nodes differs unexpectedly. While these conflicts shouldn't happen, not all can be reliably prevented in the current implementation.

Note

Modifying a row's PRIMARY KEY can cause divergent conflicts if another node changes the same row's key before all nodes process the change. Avoid changing primary keys, or restrict changes to one designated node. For more information, see the section called "UPDATE conflicts on the PRIMARY KEY".

Divergent conflicts involving row data typically require administrator intervention. To resolve these conflicts, you must manually adjust data on one node to match another while temporarily disabling replication using `pgactive.pgactive_do_not_replicate`. These conflicts shouldn't occur when you use pgactive as documented and avoid settings or functions marked as unsafe.

As an administrator, you must manually resolve these conflicts. Depending on the conflict type, you'll need to use advanced options like `pgactive.pgactive_do_not_replicate`. Use these options with caution, as improper use can worsen the situation. Due to the variety of possible conflicts, we can't provide universal resolution instructions.

Divergent conflicts occur when data that should be identical across different nodes unexpectedly differs. While these conflicts shouldn't happen, not all such conflicts can be reliably prevented in the current implementation.

Avoiding or tolerating conflicts

In most cases, you can use appropriate application design to avoid conflicts or make your application tolerant of conflicts.

Conflicts only occur when simultaneous operations happen on multiple nodes. To avoid conflicts:

- Write to only one node
- Write to independent database subsets on each node (for example, assign each node a separate schema)

For INSERT vs INSERT conflicts, use Global sequences to prevent conflicts entirely.

If conflicts are not acceptable for your use case, consider implementing distributed locking at the application level. Often, the best approach is to design your application to work with pgactive's conflict resolution mechanisms rather than trying to prevent all conflicts. For more information, see the section called “Types of conflicts”.

Conflict logging

pgactivelink logs conflict incidents in the `pgactive.pgactive_conflict_history` table to help you diagnose and handle active-active conflicts. Conflict logging to this table only occurs when you set `pgactive.log_conflicts_to_table` to true. The pgactive extension also logs conflicts to the PostgreSQL log file when log_min_messages is set to `LOG` or `lower`, regardless of the `pgactive.log_conflicts_to_table` setting.

Use the conflict history table to:

- Measure how frequently your application creates conflicts
- Identify where conflicts occur
- Improve your application to reduce conflict rates
- Detect cases where conflict resolutions don't produce desired results
- Determine where you need user-defined conflict triggers or application design changes

For row conflicts, you can optionally log row values. This is controlled by the `pgactive.log_conflicts_to_table` setting. Note that:

- This is a global database-wide option

- There is no per-table control over row value logging
- No limits are applied to field numbers, array elements, or field lengths
- Enabling this feature may not be advisable if you work with multi-megabyte rows that might trigger conflicts

Since the conflict history table contains data from every table in the database (each with potentially different schemas), logged row values are stored as JSON fields. The JSON is created using `row_to_json`, similar to calling it directly from SQL. PostgreSQL doesn't provide a `json_to_row` function, so you'll need table-specific code (in PL/pgSQL, PL/Python, PL/Perl, etc.) to reconstruct a composite-typed tuple from the logged JSON.

> **Note**
>
> Support for user-defined conflicts is planned as a future extension feature.

Understanding the pgactive schema

The pgactive schema manages active-active replication in RDS for PostgreSQL. This schema contains tables that store replication configuration and status information.

> **Note**
>
> The pgactive schema is evolving and subject to change. Don't modify the data in these tables directly.

The key tables in the pgactive schema include:

- `pgactive_nodes` – Stores information about nodes in the active-active replication group.
- `pgactive_connections` – Stores connection details for each node.

pgactive_nodes

The pgactive_nodes stores information about the nodes participating in the active-active replication group.

Column	Type	Collation	Nullable	Default
node_sysid	text	–	not null	–
node_time line	oid	–	not null	–
node_dboid	oid	–	not null	–
node_status	char	–	not null	–
node_name	text	–	not null	–
node_dsn	text	–	not null	–
node_init _from_dsn	text	–	not null	–
node_read _only	boolean	–	–	false
node_seq_id	smallint	–	not null	–

node_sysid

Unique ID for a node, generated during `pgactive_create_group` or `pgactive_join_group`

node_status

Readiness of the node:

- **b** - beginning setup
- **i** - initializing
- **c** - catchup
- **o** - creating outbound slots
- **r** - ready
- **k** - killed

> **Note**
>
> This column doesn't indicate if a node is connected or disconnected.

node_name

User-provided unique node name.

node_dsn

Connection string or user mapping name

node_init_from_dsn

DSN from which this node was created.

pgactive_connection

The pgactive_connections stores connection details for each node.

Column	Type	Collation	Nullable	Default
conn_sysid	text	none	not null	none
conn_timeline	oid	none	not null	none
conn_dboid	oid	none	not null	none
conn_dsn	text	none	not null	none
conn_apply_delay	integer	none	none	none
conn_replication_sets	text	none	none	none

conn_sysid

Node identifier for the node this entry refers to.

conn_dsn

Same as pgactive.pgactive_nodes node_dsn.

conn_apply_delay

If set, milliseconds to wait before applying each transaction from the remote node. Mainly for debugging. If null, the global default applies.

Working with replication sets

Replication sets determine which tables to include or exclude from replication operations. By default, all tables are replicated unless you specify otherwise using the following functions:

- `pgactive_exclude_table_replication_set()` - Excludes specified tables from replication
- `pgactive_include_table_replication_set()` - Includes specified tables in replication

> **Note**
>
> Before you configure replication sets, consider the following:
>
> - You can configure table inclusion or exclusion only after running `pgactive_create_group()` but before `pgactive_join_group()`.
> - After you use `pgactive_exclude_table_replication_set()`, you can't use `pgactive_include_table_replication_set()`.
> - After you use `pgactive_include_table_replication_set()`, you can't use `pgactive_exclude_table_replication_set()`.

The system handles newly created tables differently based on your initial configuration:

- If you excluded tables: Any new tables created after `pgactive_join_group()` are automatically included in replication
- If you included tables: Any new tables created after `pgactive_join_group()` are automatically excluded from replication.

To view the replication set configuration for a specific table, use the `pgactive.pgactive_get_table_replication_sets()` function.

pgactive functions reference

Following, you can find a list of pgactive functions with their parameters, return values, and practical usage notes to help you effectively use them:

get_last_applied_xact_info

Retrieves the last applied transaction information for a specified node.

Arguments

- sysid (text) - timeline OID
- dboid (OID)

Return type

It records the following:

- last_applied_xact_id (OID)
- last_applied_xact_committs (timestamp with time zone)
- last_applied_xact_at (timestamp with time zone)

Usage notes

Use this function to retrieve the last applied transaction information for a specified node.

pgactive_apply_pause

Pauses the replication apply process.

Arguments

None

Return type

boolean

Usage notes

Call this function to pause the replication apply process.

pgactive_apply_resume

Resumes the replication apply process.

Arguments

None

Return type

void

Usage notes

Call this function to resume the replication apply process.

pgactive_is_apply_paused

Checks if replication apply is currently paused.

Arguments

None

Return type

boolean

Usage notes

Use this function to check if replication apply is currently paused.

pgactive_create_group

Creates a pgactive group by converting a standalone database into the initial node.

Arguments

- node_name (text)
- node_dsn (text)
- apply_delay integer DEFAULT NULL::integer - replication_sets text[] DEFAULT ARRAY['default'::text]

Return type

void

Usage notes

Creates a pgactive group by converting a standalone database into the initial node. The function performs sanity checks before transforming the node into a pgactive node. Before using this function, ensure that your PostgreSQL cluster has sufficient `max_worker_processes` available to support pgactive background workers.

pgactive_detach_nodes

Removes specified nodes from the pgactive group.

Arguments

- p_nodes (text[])

Return type

void

Usage notes

Use this function to remove specified nodes from the pgactive group.

pgactive_exclude_table_replication_set

Excludes a specific table from replication.

Arguments

- p_relation (regclass)

Return type

void

Usage notes

Use this function to exclude a specific table from replication.

pgactive_get_replication_lag_info

Retrieves detailed replication lag information, including node details, WAL status, and LSN values.

Arguments

None

Return type

SETOF record - node_name text - node_sysid text - application_name text - slot_name text - active boolean - active_pid integer - pending_wal_decoding bigint - Approximate size of WAL in bytes to be decoded on the sender node - pending_wal_to_apply bigint - Approximate size of WAL in bytes to be applied on receiving node - restart_lsn pg_lsn - confirmed_flush_lsn pg_lsn - sent_lsn pg_lsn - write_lsn pg_lsn - flush_lsn pg_lsn - replay_lsn pg_lsn

Usage notes

Call this function to retrieve replication lag information, including node details, WAL status, and LSN values.

pgactive_get_stats

Retrieves pgactive replication statistics.

Arguments

None

Return type

SETOF record - rep_node_id oid - rilocalid oid - riremoteid text - nr_commit bigint - nr_rollback bigint - nr_insert bigint - nr_insert_conflict bigint - nr_update bigint - nr_update_conflict bigint - nr_delete bigint - nr_delete_conflict bigint - nr_disconnect bigint

Usage notes

Use this function to retrieve pgactive replication statistics.

pgactive_get_table_replication_sets

Gets replication set configuration for a specific relation.

Arguments

- relation (regclass)

Return type

SETOF record

Usage notes

Call this function to get replication set configuration for a specific relation.

pgactive_include_table_replication_set

Includes a specific table in replication.

Arguments

- p_relation (regclass)

Return type

void

Usage notes

Use this function to include a specific table in replication.

pgactive_join_group

Adds a node to an existing pgactive group.

Arguments

- node_name (text)
- node_dsn (text)
- join_using_dsn (text)
- apply_delay (integer, optional)
- replication_sets (text[], default: ['default'])
- bypass_collation_check (boolean, default: false)
- bypass_node_identifier_creation (boolean, default: false)
- bypass_user_tables_check (boolean, default: false)

Return type

void

Usage notes

Call this function to add a node to an existing pgactive group. Ensure your PostgreSQL cluster has sufficient max_worker_processes for pgactive background workers.

pgactive_remove

Removes all pgactive components from the local node.

Arguments

- force (boolean, default: false)

Return type

void

Usage notes

Call this function to remove all pgactive components from the local node.

pgactive_snowflake_id_nextval

Generates node-specific unique sequence values.

Arguments

- regclass

Return type

bigint

Usage notes

Use this function to generate node-specific unique sequence values.

pgactive_update_node_conninfo

Updates connection information for a pgactive node.

Arguments

- node_name_to_update (text)
- node_dsn_to_update (text)

Return type

void

Usage notes

Use this function to update connection information for a pgactive node.

pgactive_wait_for_node_ready

Monitors the progress of group creation or joining operations.

Arguments

- timeout (integer, default: 0)
- progress_interval (integer, default: 60)

Return type

void

Usage notes

Call this function to monitor the progress of group creation or joining operations.

Handling conflicts in active-active replication

The `pgactive` extension works per database and not per cluster. Each DB instance that uses `pgactive` is an independent instance and can accept data changes from any source. When a change is sent to a DB instance, PostgreSQL commits it locally and then uses `pgactive` to replicate the change asynchronously to other DB instances. When two PostgreSQL DB instances update the same record at nearly the same time, a conflict can occur.

The `pgactive` extension provides mechanisms for conflict detection and automatic resolution. It tracks the time stamp when the transaction was committed on both the DB instances and automatically applies the change with the latest time stamp. The `pgactive` extension also logs when a conflict occurs in the `pgactive.pgactive_conflict_history` table.

The `pgactive.pgactive_conflict_history` will keep growing. You may want to define a purging policy. This can be done by deleting some records on a regular basis or defining a partitioning scheme for this relation (and later detach, drop, truncate partitions of interest). To implement the purging policy on a regular basis, one option is to use the `pg_cron` extension. See

the following information of an example for the pg_cron history table, Scheduling maintenance with the PostgreSQL pg_cron extension.

Handling sequences in active-active replication

An RDS for PostgreSQL DB instance with the pgactive extension uses two different sequence mechanisms to generate unique values.

Global Sequences

To use a global sequence, create a local sequence with the CREATE SEQUENCE statement. Use pgactive.pgactive_snowflake_id_nextval(seqname) instead of usingnextval(seqname) to get the next unique value of the sequence.

The following example creates a global sequence:

```
app=> CREATE TABLE gstest (
      id bigint primary key,
      parrot text
    );
```

```
app=>CREATE SEQUENCE gstest_id_seq OWNED BY gstest.id;
```

```
app=> ALTER TABLE gstest \
      ALTER COLUMN id SET DEFAULT \
      pgactive.pgactive_snowflake_id_nextval('gstest_id_seq');
```

Partitioned sequences

In split-step or partitioned sequences, a normal PostgreSQL sequence is used on each node. Each sequence increments by the same amount and starts at different offsets. For example, with step 100, the node 1 generates sequence as 101, 201, 301, and so on and the node 2 generates sequence as 102, 202, 302, and so on. This scheme works well even if the nodes can't communicate for extended periods, but requires that the designer specify a maximum number of nodes when establishing the schema and requires per-node configuration. Mistakes can easily lead to overlapping sequences.

It is relatively simple to configure this approach with pgactive by creating the desired sequence on a node as follows:

```
CREATE TABLE some_table (generated_value bigint primary key);
```

```
app=> CREATE SEQUENCE some_seq INCREMENT 100 OWNED BY some_table.generated_value;
```

```
app=> ALTER TABLE some_table ALTER COLUMN generated_value SET DEFAULT
 nextval('some_seq');
```

Then call `setval` on each node to give a different offset starting value as follows.

```
app=>
-- On node 1
SELECT setval('some_seq', 1);

-- On node 2
SELECT setval('some_seq', 2);
```

Reducing bloat in tables and indexes with the pg_repack extension

You can use the pg_repack extension to remove bloat from tables and indexes as an alternative to VACUUM FULL. This extension is supported on RDS for PostgreSQL versions 9.6.3 and higher. For more information on the pg_repack extension and the full table repack, see the GitHub project documentation.

Unlike VACUUM FULL, the pg_repack extension requires an exclusive lock (AccessExclusiveLock) only for a short period of time during the table rebuild operation in the following cases:

- Initial creation of log table – A log table is created to record changes that occur during initial copy of the data, as shown in the following example:

```
postgres=>\dt+ repack.log_*
List of relations
-[ RECORD 1 ]-+----------
Schema        | repack
Name          | log_16490
Type          | table
Owner         | postgres
Persistence   | permanent
Access method | heap
Size          | 65 MB
Description   |
```

- Final swap-and-drop phase.

For the rest of the rebuild operation, it only needs an ACCESS SHARE lock on the original table to copy rows from it to the new table. This helps the INSERT, UPDATE, and DELETE operations to proceed as usual.

Recommendations

The following recommendations apply when you remove bloat from the tables and indexes using the pg_repack extension:

- Perform repack during non-business hours or over a maintenance window to minimize its impact on performance of other database activities.
- Closely monitor blocking sessions during the rebuild activity and ensure that there is no activity on the original table that could potentially block pg_repack, specifically during the final swap-

and-drop phase when it needs an exclusive lock on the original table. For more information, see Identifying what is blocking a query.

When you see a blocking session, you can terminate it using the following command after careful consideration. This helps in the continuation of pg_repack to finish the rebuild:

```
SELECT pg_terminate_backend(pid);
```

- While applying the accrued changes from the pg_repack's log table on systems with a very high transaction rate, the apply process might not be able to keep up with the rate of changes. In such cases, pg_repack would not be able to complete the apply process. For more information, see Monitoring the new table during the repack. If indexes are severely bloated, an alternative solution is to perform an index only repack. This also helps VACUUM's index cleanup cycles to finish faster.

  You can skip the index cleanup phase using manual VACUUM from PostgreSQL version 12, and it is skipped automatically during emergency autovacuum from PostgreSQL version 14. This helps VACUUM complete faster without removing the index bloat and is only meant for emergency situations such as preventing wraparound VACUUM. For more information, see Avoiding bloat in indexes in the Amazon Aurora User Guide.

Pre-requisites

- The table must have PRIMARY KEY or not-null UNIQUE constraint.
- The extension version must be the same for both the client and the server.
- Ensure that the RDS instance has more FreeStorageSpace than the total size of the table without the bloat. As an example, consider the total size of the table including TOAST and indexes as 2TB, and total bloat in the table as 1TB. The required FreeStorageSpace must be more than value returned by the following calculation:

  2TB (Table size) - 1TB (Table bloat) = 1TB

  You can use the following query to check the total size of the table and use pgstattuple to derive bloat. For more information, see Diagnosing table and index bloat in the Amazon Aurora User Guide

```
SELECT pg_size_pretty(pg_total_relation_size('table_name')) AS total_table_size;
```

This space is reclaimed after the completion of the activity.

- Ensure that the RDS instance has enough compute and IO capacity to handle the repack operation. You might consider to scale up the instance class for optimal balance of performance.

To use the pg_repack extension

1. Install the pg_repack extension on your RDS for PostgreSQL DB instance by running the following command.

```
CREATE EXTENSION pg_repack;
```

2. Run the following commands to grant write access to temporary log tables created by pg_repack.

```
ALTER DEFAULT PRIVILEGES IN SCHEMA repack GRANT INSERT ON TABLES TO PUBLIC;
ALTER DEFAULT PRIVILEGES IN SCHEMA repack GRANT USAGE, SELECT ON SEQUENCES TO
 PUBLIC;
```

3. Connect to the database using the pg_repack client utility. Use an account that has rds_superuser privileges. As an example, assume that rds_test role has rds_superuser privileges. The following syntax performs pg_repack for full tables including all the table indexes in the postgres database.

```
pg_repack -h db-instance-name.111122223333.aws-region.rds.amazonaws.com -U rds_test
 -k postgres
```

> **Note**
>
> You must connect using the -k option. The -a option is not supported.

The response from the pg_repack client provides information on the tables on the DB instance that are repacked.

```
INFO: repacking table "pgbench_tellers"
INFO: repacking table "pgbench_accounts"
INFO: repacking table "pgbench_branches"
```

4. The following syntax repacks a single table `orders` including indexes in `postgres` database.

```
pg_repack -h db-instance-name.111122223333.aws-region.rds.amazonaws.com -U rds_test
 --table orders -k postgres
```

The following syntax repacks only indexes for `orders` table in `postgres` database.

```
pg_repack -h db-instance-name.111122223333.aws-region.rds.amazonaws.com -U rds_test
 --table orders --only-indexes -k postgres
```

Monitoring the new table during the repack

- The size of the database is increased by the total size of the table minus bloat, until swap-and-drop phase of repack. You can monitor the growth rate of the database size, calculate the speed of the repack, and roughly estimate the time it takes to complete initial data transfer.

  As an example, consider the total size of the table as 2TB, the size of the database as 4TB, and total bloat in the table as 1TB. The database total size value returned by the calculation at the end of the repack operation is the following:

  `2TB (Table size) + 4 TB (Database size) - 1TB (Table bloat) = 5TB`

  You can roughly estimate the speed of the repack operation by sampling the growth rate in bytes between two points in time. If the growth rate is 1GB per minute, it can take 1000 minutes or 16.6 hours approximately to complete the initial table build operation. In addition to the initial table build, `pg_repack` also needs to apply accrued changes. The time it takes depends on the rate of applying ongoing changes plus accrued changes.

  > **Note**
  >
  > You can use `pgstattuple` extension to calculate the bloat in the table. For more information, see pgstattuple .

- The number of rows in the `pg_repack's` log table, under the repack schema represents the volume of changes pending to be applied to the new table after the initial load.

You can check the pg_repack's log table in pg_stat_all_tables to monitor the changes applied to the new table. pg_stat_all_tables.n_live_tup indicates the number of records that are pending to be applied to the new table. For more information, see pg_stat_all_tables.

```
postgres=>SELECT relname,n_live_tup FROM pg_stat_all_tables WHERE schemaname =
 'repack' AND relname ILIKE '%log%';

-[ RECORD 1 ]---------
relname    | log_16490
n_live_tup | 2000000
```

- You can use the pg_stat_statements extension to find out the time taken by each step in the repack operation. This is helpful in preparation for applying the same repack operation in a production environment. You may adjust the LIMIT clause for extending the output further.

```
postgres=>SELECT
     SUBSTR(query, 1, 100) query,
     round((round(total_exec_time::numeric, 6) / 1000 / 60),4)
 total_exec_time_in_minutes
 FROM
     pg_stat_statements
 WHERE
     query ILIKE '%repack%'
 ORDER BY
     total_exec_time DESC LIMIT 5;

 query                                                                |
 total_exec_time_in_minutes
----------------------------------------------------------------------
+---------------------------
 CREATE UNIQUE INDEX index_16493 ON repack.table_16490 USING btree (a) |
        6.8627
 INSERT INTO repack.table_16490 SELECT a FROM ONLY public.t1           |
        6.4150
 SELECT repack.repack_apply($1, $2, $3, $4, $5, $6)                    |
        0.5395
 SELECT repack.repack_drop($1, $2)                                     |
        0.0004
 SELECT repack.repack_swap($1)                                         |
        0.0004
(5 rows)
```

Repacking is completely an out-of-place operation so the original table is not impacted and we do not anticipate any unexpected challenges that require recovery of the original table. If repack fails unexpectedly, you must inspect the cause of the error and resolve it.

After the issue is resolved, drop and recreate the `pg_repack` extension in the database where the table exists, and retry the `pg_repack` step. In addition, the availability of compute resources and concurrent accessibility of the table plays a crucial role in the timely completion of the repack operation.

Upgrading and using the PLV8 extension

PLV8 is a trusted Javascript language extension for PostgreSQL. You can use it for stored procedures, triggers, and other procedural code that's callable from SQL. This language extension is supported by all current releases of PostgreSQL.

If you use PLV8 and upgrade PostgreSQL to a new PLV8 version, you immediately take advantage of the new extension. Take the following steps to synchronize your catalog metadata with the new version of PLV8. These steps are optional, but we highly recommend that you complete them to avoid metadata mismatch warnings.

The upgrade process drops all your existing PLV8 functions. Thus, we recommend that you create a snapshot of your RDS for PostgreSQL DB instance before upgrading. For more information, see Creating a DB snapshot for a Single-AZ DB instance for Amazon RDS.

Important

Starting with PostgreSQL version 18, Amazon RDS for PostgreSQL will deprecate the `plcoffee` and `plls` PostgreSQL extensions. We recommend that you stop using CoffeeScript and LiveScript in your applications to ensure you have an upgrade path for future engine version upgrades.

To synchronize your catalog metadata with a new version of PLV8

1. Verify that you need to update. To do this, run the following command while connected to your instance.

   ```
   SELECT * FROM pg_available_extensions WHERE name IN ('plv8','plls','plcoffee');
   ```

   If your results contain values for an installed version that is a lower number than the default version, continue with this procedure to update your extensions. For example, the following result set indicates that you should update.

   ```
   name    | default_version | installed_version |                      comment
   --------+-----------------+-------------------
   +-------------------------------------------------
   plls    | 2.1.0           | 1.5.3             | PL/LiveScript (v8) trusted
    procedural language
   ```

```
plcoffee| 2.1.0           | 1.5.3             | PL/CoffeeScript (v8) trusted
 procedural language
plv8    | 2.1.0           | 1.5.3             | PL/JavaScript (v8) trusted
 procedural language
(3 rows)
```

2. Create a snapshot of your RDS for PostgreSQL DB instance if you haven't done so yet. You can continue with the following steps while the snapshot is being created.
3. Get a count of the number of PLV8 functions in your DB instance so you can validate that they are all in place after the upgrade. For example, the following SQL query returns the number of functions written in plv8, plcoffee, and plls.

```
SELECT proname, nspname, lanname
FROM pg_proc p, pg_language l, pg_namespace n
WHERE p.prolang = l.oid
AND n.oid = p.pronamespace
AND lanname IN ('plv8','plcoffee','plls');
```

4. Use pg_dump to create a schema-only dump file. For example, create a file on your client machine in the `/tmp` directory.

```
./pg_dump -Fc --schema-only -U master postgres >/tmp/test.dmp
```

This example uses the following options:

- `-Fc` – Custom format
- --schema-only – Dump only the commands necessary to create schema (functions in this case)
- `-U` – The RDS master user name
- `database` – The database name for our DB instance

For more information on pg_dump, see pg_dump in the PostgreSQL documentation.

5. Extract the "CREATE FUNCTION" DDL statement that is present in the dump file. The following example uses the `grep` command to extract the DDL statement that creates the functions and save them to a file. You use this in subsequent steps to recreate the functions.

```
./pg_restore -l /tmp/test.dmp | grep FUNCTION > /tmp/function_list
```

For more information on pg_restore, see pg_restore in the PostgreSQL documentation.

6. Drop the functions and extensions. The following example drops any PLV8 based objects. The cascade option ensures that any dependent are dropped.

```
DROP EXTENSION plv8 CASCADE;
```

If your PostgreSQL instance contains objects based on plcoffee or plls, repeat this step for those extensions.

7. Create the extensions. The following example creates the plv8, plcoffee, and plls extensions.

```
CREATE EXTENSION plv8;
CREATE EXTENSION plcoffee;
CREATE EXTENSION plls;
```

8. Create the functions using the dump file and "driver" file.

The following example recreates the functions that you extracted previously.

```
./pg_restore -U master -d postgres -Fc -L /tmp/function_list /tmp/test.dmp
```

9. Verify that all your functions have been recreated by using the following query.

```
SELECT * FROM pg_available_extensions WHERE name IN ('plv8','plls','plcoffee');
```

The PLV8 version 2 adds the following extra row to your result set:

```
    proname    |  nspname   | lanname
---------------+------------+----------
 plv8_version  | pg_catalog | plv8
```

Using PL/Rust to write PostgreSQL functions in the Rust language

PL/Rust is a trusted Rust language extension for PostgreSQL. You can use it for stored procedures, functions, and other procedural code that's callable from SQL. The PL/Rust language extension is available in the following versions:

- RDS for PostgreSQL 17.1 and higher 17 versions

- RDS for PostgreSQL 16.1 and higher 16 versions
- RDS for PostgreSQL 15.2-R2 and higher 15 versions
- RDS for PostgreSQL 14.9 and higher 14 versions
- RDS for PostgreSQL 13.12 and higher 13 versions

For more information, see PL/Rust on GitHub.

Topics

- Setting up PL/Rust
- Creating functions with PL/Rust
- Using crates with PL/Rust
- PL/Rust limitations

Setting up PL/Rust

To install the plrust extension on your DB instance, add plrust to the `shared_preload_libraries` parameter in the DB parameter group associated with your DB instance. With the plrust extension installed, you can create functions.

To modify the `shared_preload_libraries` parameter, your DB instance must be associated with a custom parameter group. For information about creating a custom DB parameter group, see Parameter groups for Amazon RDS.

You can install the plrust extension using the AWS Management Console or the AWS CLI.

The following steps assume that your DB instance is associated with a custom DB parameter group.

Console

Install the plrust extension in the `shared_preload_libraries` parameter

Complete the following steps using an account that is a member of the `rds_superuser` group (role).

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.

3. Choose the name of your DB instance to display its details.
4. Open the **Configuration** tab for your DB instance and find the DB instance parameter group link.
5. Choose the link to open the custom parameters associated with your DB instance.
6. In the **Parameters** search field, type `shared_pre` to find the **`shared_preload_libraries`** parameter.
7. Choose **Edit parameters** to access the property values.
8. Add plrust to the list in the **Values** field. Use a comma to separate items in the list of values.
9. Reboot the DB instance so that your change to the `shared_preload_libraries` parameter takes effect. The initial reboot may require additional time to complete.
10. When the instance is available, verify that plrust has been initialized. Use `psql` to connect to the DB instance, and then run the following command.

```
SHOW shared_preload_libraries;
```

Your output should look similar to the following:

```
shared_preload_libraries
--------------------------
rdsutils,plrust
(1 row)
```

AWS CLI

Install the plrust extension in the shared_preload_libraries parameter

Complete the following steps using an account that is a member of the `rds_superuser` group (role).

1. Use the modify-db-parameter-group AWS CLI command to add plrust to the `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=plrust,ApplyMethod=pending-
reboot" \
```

```
    --region aws-region
```

2. Use the reboot-db-instance AWS CLI command to reboot the DB instance and initialize the plrust library. The initial reboot may require additional time to complete.

```
aws rds reboot-db-instance \
    --db-instance-identifier your-instance \
    --region aws-region
```

3. When the instance is available, you can verify that plrust has been initialized. Use `psql` to connect to the DB instance, and then run the following command.

```
SHOW shared_preload_libraries;
```

Your output should look similar to the following:

```
shared_preload_libraries
--------------------------
rdsutils,plrust
(1 row)
```

Creating functions with PL/Rust

PL/Rust will compile the function as a dynamic library, load it, and execute it.

The following Rust function filters multiples out of an array.

```
postgres=> CREATE LANGUAGE plrust;
CREATE EXTENSION
```

```
CREATE OR REPLACE FUNCTION filter_multiples(a BIGINT[], multiple BIGINT) RETURNS
 BIGINT[]
    IMMUTABLE STRICT
    LANGUAGE PLRUST AS
$$
    Ok(Some(a.into_iter().filter(|x| x.unwrap() % multiple != 0).collect()))
$$;

WITH gen_values AS (
SELECT ARRAY(SELECT * FROM generate_series(1,100)) as arr)
```

```
SELECT filter_multiples(arr, 3)
from gen_values;
```

Using crates with PL/Rust

In RDS for PostgreSQL versions 16.3-R2 and higher, 15.7-R2 and higher 15 versions, 14.12-R2 and higher 14 versions, and 13.15-R2 and higher 13 versions, PL/Rust supports additional crates:

- `url`
- `regex`
- `serde`
- `serde_json`

In RDS for PostgreSQL versions 15.5-R2 and higher, 14.10-R2 and higher 14 versions, and 13.13-R2 and higher 13 versions, PL/Rust supports two additional crates:

- `croaring-rs`
- `num-bigint`

Starting with Amazon RDS for PostgreSQL versions 15.4, 14.9, and 13.12, PL/Rust supports the following crates:

- `aes`
- `ctr`
- `rand`

Only the default features are supported for these crates. New RDS for PostgreSQL versions might contain updated versions of crates, and older versions of crates may no longer be supported.

Follow the best practices for performing a major version upgrade to test whether your PL/Rust functions are compatible with the new major version. For more information, see the blog Best practices for upgrading Amazon RDS to major and minor versions of PostgreSQL and Upgrading the PostgreSQL DB engine for Amazon RDS in the Amazon RDS User Guide.

Examples of using dependencies when creating a PL/Rust function are available at Use dependencies.

PL/Rust limitations

By default, database users can't use PL/Rust. To provide access to PL/Rust, connect as a user with rds_superuser privilege, and run the following command:

```
postgres=> GRANT USAGE ON LANGUAGE PLRUST TO user;
```

Managing spatial data with the PostGIS extension

PostGIS is an extension to PostgreSQL for storing and managing spatial information. To learn more about PostGIS, see PostGIS.net.

Starting with version 10.5, PostgreSQL supports the libprotobuf 1.3.0 library used by PostGIS for working with map box vector tile data.

Setting up the PostGIS extension requires `rds_superuser` privileges. We recommend that you create a user (role) to manage the PostGIS extension and your spatial data. The PostGIS extension and its related components add thousands of functions to PostgreSQL. Consider creating the PostGIS extension in its own schema if that makes sense for your use case. The following example shows how to install the extension in its own database, but this isn't required.

Topics

- Step 1: Create a user (role) to manage the PostGIS extension
- Step 2: Load the PostGIS extensions
- Step 3: Transfer ownership of the extension schemas
- Step 4: Transfer ownership of the PostGIS tables
- Step 5: Test the extensions
- Step 6: Upgrade the PostGIS extension
- PostGIS extension versions
- Upgrading PostGIS 2 to PostGIS 3

Step 1: Create a user (role) to manage the PostGIS extension

First, connect to your RDS for PostgreSQL DB instance as a user that has `rds_superuser` privileges. If you kept the default name when you set up your instance, you connect as `postgres`.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres
 --password
```

Create a separate role (user) to administer the PostGIS extension.

```
postgres=>  CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

Grant this role `rds_superuser` privileges, to allow the role to install the extension.

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

Create a database to use for your PostGIS artifacts. This step is optional. Or you can create a schema in your user database for the PostGIS extensions, but this also isn't required.

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

Give the `gis_admin` all privileges on the `lab_gis` database.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

Exit the session and reconnect to your RDS for PostgreSQL DB instance as `gis_admin`.

```
postgres=> psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
Password for user gis_admin:...
lab_gis=>
```

Continue setting up the extension as detailed in the next steps.

Step 2: Load the PostGIS extensions

The PostGIS extension includes several related extensions that work together to provide geospatial functionality. Depending on your use case, you might not need all the extensions created in this step.

Use `CREATE EXTENSION` statements to load the PostGIS extensions.

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
```

```
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

You can verify the results by running the SQL query shown in the following example, which lists the extensions and their owners.

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;
List of schemas
     Name     |   Owner
--------------+-----------
 public       | postgres
 tiger        | rdsadmin
 tiger_data   | rdsadmin
 topology     | rdsadmin
(4 rows)
```

Step 3: Transfer ownership of the extension schemas

Use the ALTER SCHEMA statements to transfer ownership of the schemas to the `gis_admin` role.

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

You can confirm the ownership change by running the following SQL query. Or you can use the \dn metacommand from the psql command line.

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
```

```
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;

      List of schemas
    Name     |     Owner
-------------+--------------
 public      | postgres
 tiger       | gis_admin
 tiger_data  | gis_admin
 topology    | gis_admin
(4 rows)
```

Step 4: Transfer ownership of the PostGIS tables

> **Note**
>
> Do not change ownership of the PostGIS functions. Proper operation and future upgrades of PostGIS require these functions to retain original ownership. For more information about PostGIS permissions, see PostgreSQL Security.

Use the following function to transfer ownership of the PostGIS tables to the `gis_admin` role. Run the following statement from the psql prompt to create the function.

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE
 $1; RETURN $1; END; $f$;
CREATE FUNCTION
```

Next, run the following query to run the `exec` function that in turn runs the statements and alters the permissions.

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname)
 || ' OWNER TO gis_admin;')
  FROM (
    SELECT nspname, relname
    FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
    WHERE nspname in ('tiger','topology') AND
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

Step 5: Test the extensions

To avoid needing to specify the schema name, add the `tiger` schema to your search path using the following command.

```
SET search_path=public,tiger;
SET
```

Test the `tiger` schema by using the following SELECT statement.

```
SELECT address, streetname, streettypeabbrev, zip
 FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
       1 | Devonshire | Pl               | 02109
(1 row)
```

To learn more about this extension, see Tiger Geocoder in the PostGIS documentation.

Test access to the `topology` schema by using the following `SELECT` statement. This calls the `createtopology` function to register a new topology object (my_new_topo) with the specified spatial reference identifier (26986) and default tolerance (0.5). To learn more, see CreateTopology in the PostGIS documentation.

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
 createtopology
----------------
              1
(1 row)
```

Step 6: Upgrade the PostGIS extension

Each new release of PostgreSQL supports one or more versions of the PostGIS extension compatible with that release. Upgrading the PostgreSQL engine to a new version doesn't automatically upgrade the PostGIS extension. Before upgrading the PostgreSQL engine, you typically upgrade PostGIS to the newest available version for the current PostgreSQL version. For details, see PostGIS extension versions.

After the PostgreSQL engine upgrade, you then upgrade the PostGIS extension again, to the version supported for the newly upgraded PostgreSQL engine version. For more information

about upgrading the PostgreSQL engine, see How to perform a major version upgrade for RDS for PostgreSQL.

You can check for available PostGIS extension version updates on your RDS for PostgreSQL DB instance at any time. To do so, run the following command. This function is available with PostGIS 2.5.0 and higher versions.

```
SELECT postGIS_extensions_upgrade();
```

If your application doesn't support the latest PostGIS version, you can install an older version of PostGIS that's available in your major version as follows.

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

If you want to upgrade to a specific PostGIS version from an older version, you can also use the following command.

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

Depending on the version that you're upgrading from, you might need to use this function again. The result of the first run of the function determines if an additional upgrade function is needed. For example, this is the case for upgrading from PostGIS 2 to PostGIS 3. For more information, see Upgrading PostGIS 2 to PostGIS 3.

If you upgraded this extension to prepare for a major version upgrade of the PostgreSQL engine, you can continue with other preliminary tasks. For more information, see How to perform a major version upgrade for RDS for PostgreSQL.

PostGIS extension versions

We recommend that you install the versions of all extensions such as PostGIS as listed in Extension versions for Amazon RDS for PostgreSQL in the *Amazon RDS for PostgreSQL Release Notes.* To get a list of versions that are available in your release, use the following command.

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

You can find version information in the following sections in the *Amazon RDS for PostgreSQL Release Notes*:

- PostgreSQL version 16 extensions supported on Amazon RDS

- PostgreSQL version 15 extensions supported on Amazon RDS
- PostgreSQL version 14 extensions supported on Amazon RDS
- PostgreSQL version 13 extensions supported on Amazon RDS
- PostgreSQL version 12 extensions supported on Amazon RDS
- PostgreSQL version 11 extensions supported on Amazon RDS
- PostgreSQL version 10 extensions supported on Amazon RDS
- PostgreSQL version 9.6.x extensions supported on Amazon RDS

Upgrading PostGIS 2 to PostGIS 3

Starting with version 3.0, the PostGIS raster functionality is now a separate extension, `postgis_raster`. This extension has its own installation and upgrade path. This removes dozens of functions, data types, and other artifacts required for raster image processing from the core `postgis` extension. That means that if your use case doesn't require raster processing, you don't need to install the `postgis_raster` extension.

In the following upgrade example, the first upgrade command extracts raster functionality into the `postgis_raster` extension. A second upgrade command is then required to upgrade `postgis_raster` to the new version.

To upgrade from PostGIS 2 to PostGIS 3

1. Identify the default version of PostGIS that's available to the PostgreSQL version on your RDS for PostgreSQL DB instance. To do so, run the following query.

```
SELECT * FROM pg_available_extensions
    WHERE default_version > installed_version;
  name   | default_version | installed_version |                          comment
---------+-----------------+-------------------
+------------------------------------------------------------
 postgis | 3.1.4           | 2.3.7             | PostGIS geometry and geography
 spatial types and functions
(1 row)
```

2. Identify the versions of PostGIS installed in each database on your RDS for PostgreSQL DB instance. In other words, query each user database as follows.

```
SELECT
```

```
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
    AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
  Name   | Version | Schema |                            Description
---------+---------+--------
+------------------------------------------------------------------------
 postgis | 2.3.7   | public | PostGIS geometry, geography, and raster spatial types
 and functions
(1 row)
```

This mismatch between the default version (PostGIS 3.1.4) and the installed version (PostGIS 2.3.7) means that you need to upgrade the PostGIS extension.

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. Run the following query to verify that the raster functionality is now in its own package.

```
SELECT
    probin,
    count(*)
FROM
    pg_proc
WHERE
    probin LIKE '%postgis%'
GROUP BY
    probin;
          probin          | count
--------------------------+-------
 $libdir/rtpostgis-2.3    | 107
 $libdir/postgis-3        | 487
```

```
(2 rows)
```

The output shows that there's still a difference between versions. The PostGIS functions are version 3 (postgis-3), while the raster functions (rtpostgis) are version 2 (rtpostgis-2.3). To complete the upgrade, you run the upgrade command again, as follows.

```
postgres=> SELECT postgis_extensions_upgrade();
```

You can safely ignore the warning messages. Run the following query again to verify that the upgrade is complete. The upgrade is complete when PostGIS and all related extensions aren't marked as needing upgrade.

```
SELECT postgis_full_version();
```

4. Use the following query to see the completed upgrade process and the separately packaged extensions, and verify that their versions match.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
      Name      | Version | Schema |                            Description
----------------+---------+--------
+------------------------------------------------------------------------
 postgis        | 3.1.5   | public | PostGIS geometry, geography, and raster
 spatial types and functions
 postgis_raster | 3.1.5   | public | PostGIS raster types and functions
(2 rows)
```

The output shows that the PostGIS 2 extension was upgraded to PostGIS 3, and both `postgis` and the now separate `postgis_raster` extension are version 3.1.5.

After this upgrade completes, if you don't plan to use the raster functionality, you can drop the extension as follows.

```
DROP EXTENSION postgis_raster;
```

Working with the supported foreign data wrappers for Amazon RDS for PostgreSQL

A foreign data wrapper (FDW) is a specific type of extension that provides access to external data. For example, the `oracle_fdw` extension allows your RDS for PostgreSQL DB cluster to work with Oracle databases. As another example, by using the PostgreSQL native `postgres_fdw` extension you can access data stored in PostgreSQL DB instances external to your RDS for PostgreSQL DB instance.

Following, you can find information about several supported PostgreSQL foreign data wrappers.

Topics

- Using the log_fdw extension to access the DB log using SQL
- Using the postgres_fdw extension to access external data
- Working with MySQL databases by using the mysql_fdw extension
- Working with Oracle databases by using the oracle_fdw extension
- Working with SQL Server databases by using the tds_fdw extension

Using the log_fdw extension to access the DB log using SQL

RDS for PostgreSQL DB instance supports the `log_fdw` extension, which you can use to access your database engine log using a SQL interface. The `log_fdw` extension provides two functions that make it easy to create foreign tables for database logs:

- `list_postgres_log_files` – Lists the files in the database log directory and the file size in bytes.
- `create_foreign_table_for_log_file(table_name text, server_name text, log_file_name text)` – Builds a foreign table for the specified file in the current database.

All functions created by `log_fdw` are owned by `rds_superuser`. Members of the `rds_superuser` role can grant access to these functions to other database users.

By default, the log files are generated by Amazon RDS in `stderr` (standard error) format, as specified in `log_destination` parameter. There are only two options for this parameter, `stderr` and `csvlog` (comma-separated values, CSV). If you add the `csvlog` option to the parameter, Amazon RDS generates both `stderr` and `csvlog` logs. This can affect the storage capacity on

your DB cluster, so you need to be aware of the other parameters that affect log handling. For more information, see Setting the log destination (`stderr, csvlog`).

One benefit of generating `csvlog` logs is that the `log_fdw` extension lets you build foreign tables with the data neatly split into several columns. To do this, your instance needs to be associated with a custom DB parameter group so that you can change the setting for `log_destination`. For more information about how to do so, see Working with parameters on your RDS for PostgreSQL DB instance.

The following example assumes that the `log_destination` parameter includes `cvslog`.

To use the log_fdw extension

1. Install the `log_fdw` extension.

```
postgres=> CREATE EXTENSION log_fdw;
CREATE EXTENSION
```

2. Create the log server as a foreign data wrapper.

```
postgres=> CREATE SERVER log_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER
```

3. Select all from a list of log files.

```
postgres=> SELECT * FROM list_postgres_log_files() ORDER BY 1;
```

   A sample response is as follows.

```
          file_name           | file_size_bytes
------------------------------+-----------------
 postgresql.log.2023-08-09-22.csv |            1111
 postgresql.log.2023-08-09-23.csv |            1172
 postgresql.log.2023-08-10-00.csv |            1744
 postgresql.log.2023-08-10-01.csv |            1102
(4 rows)
```

4. Create a table with a single 'log_entry' column for the selected file.

```
postgres=> SELECT create_foreign_table_for_log_file('my_postgres_error_log',
     'log_server', 'postgresql.log.2023-08-09-22.csv');
```

The response provides no detail other than that the table now exists.

```
----------------------------------
(1 row)
```

5. Select a sample of the log file. The following code retrieves the log time and error message description.

```
postgres=> SELECT log_time, message FROM my_postgres_error_log ORDER BY 1;
```

A sample response is as follows.

```
            log_time             |                                  message
---------------------------------
+-------------------------------------------------------------------------------
Tue Aug 09 15:45:18.172 2023 PDT | ending log output to stderr
Tue Aug 09 15:45:18.175 2023 PDT | database system was interrupted; last known up
 at 2023-08-09 22:43:34 UTC
Tue Aug 09 15:45:18.223 2023 PDT | checkpoint record is at 0/90002E0
Tue Aug 09 15:45:18.223 2023 PDT | redo record is at 0/90002A8; shutdown FALSE
Tue Aug 09 15:45:18.223 2023 PDT | next transaction ID: 0/1879; next OID: 24578
Tue Aug 09 15:45:18.223 2023 PDT | next MultiXactId: 1; next MultiXactOffset: 0
Tue Aug 09 15:45:18.223 2023 PDT | oldest unfrozen transaction ID: 1822, in
 database 1
(7 rows)
```

Using the postgres_fdw extension to access external data

You can access data in a table on a remote database server with the postgres_fdw extension. If you set up a remote connection from your PostgreSQL DB instance, access is also available to your read replica.

To use postgres_fdw to access a remote database server

1. Install the postgres_fdw extension.

```
CREATE EXTENSION postgres_fdw;
```

2. Create a foreign data server using CREATE SERVER.

```
CREATE SERVER foreign_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'xxx.xx.xxx.xx', port '5432', dbname 'foreign_db');
```

3. Create a user mapping to identify the role to be used on the remote server.

> **Important**
>
> To redact the password so that it doesn't appear in the logs, set `log_statement=none` at the session level. Setting at the parameter level doesn't redact the password.

```
CREATE USER MAPPING FOR local_user
SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. Create a table that maps to the table on the remote server.

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
        data text)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

Working with MySQL databases by using the mysql_fdw extension

To access a MySQL-compatible database from your RDS for PostgreSQL DB instance, you can install and use the `mysql_fdw` extension. This foreign data wrapper lets you work with RDS for MySQL, Aurora MySQL, MariaDB, and other MySQL-compatible databases. The connection from RDS for PostgreSQL DB instance to the MySQL database is encrypted on a best-effort basis, depending on the client and server configurations. However, you can enforce encryption if you like. For more information, see Using encryption in transit with the extension.

The `mysql_fdw` extension is supported on Amazon RDS for PostgreSQL version 14.2, 13.6, and higher releases. It supports selects, inserts, updates, and deletes from an RDS for PostgreSQL DB to tables on a MySQL-compatible database instance.

Topics

- Setting up your RDS for PostgreSQL DB to use the mysql_fdw extension
- Example: Working with an RDS for MySQL database from RDS for PostgreSQL
- Using encryption in transit with the extension

Setting up your RDS for PostgreSQL DB to use the mysql_fdw extension

Setting up the `mysql_fdw` extension on your RDS for PostgreSQL DB instance involves loading the extension in your DB instance and then creating the connection point to the MySQL DB instance. For that task, you need to have the following details about the MySQL DB instance:

- Hostname or endpoint. For an RDS for MySQL DB instance, you can find the endpoint by using the Console. Choose the Connectivity & security tab and look in the "Endpoint and port" section.
- Port number. The default port number for MySQL is 3306.
- Name of the database. The DB identifier.

You also need to provide access on the security group or the access control list (ACL) for the MySQL port, 3306. Both the RDS for PostgreSQL DB instance and the RDS for MySQL DB instance need access to port 3306. If access isn't configured correctly, when you try to connect to MySQL-compatible table you see an error message similar to the following:

```
ERROR: failed to connect to MySQL: Can't connect to MySQL server on 'hostname.aws-
region.rds.amazonaws.com:3306' (110)
```

In the following procedure, you (as the `rds_superuser` account) create the foreign server. You then grant access to the foreign server to specific users. These users then create their own mappings to the appropriate MySQL user accounts to work with the MySQL DB instance.

To use mysql_fdw to access a MySQL database server

1. Connect to your PostgreSQL DB instance using an account that has the `rds_superuser` role. If you accepted the defaults when you created your RDS for PostgreSQL DB instance , the user name is `postgres`, and you can connect using the `psql` command line tool as follows:

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Install the `mysql_fdw` extension as follows:

```
postgres=> CREATE EXTENSION mysql_fdw;
CREATE EXTENSION
```

After the extension is installed on your RDS for PostgreSQL DB instance , you set up the foreign server that provides the connection to a MySQL database.

To create the foreign server

Perform these tasks on the RDS for PostgreSQL DB instance . The steps assume that you're connected as a user with `rds_superuser` privileges, such as `postgres`.

1. Create a foreign server in the RDS for PostgreSQL DB instance :

```
postgres=> CREATE SERVER mysql-db FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'db-
name.111122223333.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. Grant the appropriate users access to the foreign server. These should be non-administrator users, that is, users without the `rds_superuser` role.

```
postgres=> GRANT USAGE ON FOREIGN SERVER mysql-db to user1;
GRANT
```

PostgreSQL users create and manage their own connections to the MySQL database through the foreign server.

Example: Working with an RDS for MySQL database from RDS for PostgreSQL

Suppose that you have a simple table on an RDS for PostgreSQL DB instance . Your RDS for PostgreSQL users want to query (`SELECT`), `INSERT`, `UPDATE`, and `DELETE` items on that table. Assume that the `mysql_fdw` extension was created on your RDS for PostgreSQL DB instance, as detailed in the preceding procedure. After you connect to the RDS for PostgreSQL DB instance as a user that has `rds_superuser` privileges, you can proceed with the following steps.

1. On the RDS for PostgreSQL DB instance, create a foreign server:

```
test=> CREATE SERVER mysqldb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'your-
DB.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. Grant usage to a user who doesn't have `rds_superuser` permissions, for example, `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER mysqldb TO user1;
GRANT
```

3. Connect as *user1*, and then create a mapping to the MySQL user:

```
test=> CREATE USER MAPPING FOR user1 SERVER mysqldb OPTIONS (username 'myuser',
 password 'mypassword');
CREATE USER MAPPING
```

4. Create a foreign table linked to the MySQL table:

```
test=> CREATE FOREIGN TABLE mytab (a int, b text) SERVER mysqldb OPTIONS (dbname
 'test', table_name '');
CREATE FOREIGN TABLE
```

5. Run a simple query against the foreign table:

```
test=> SELECT * FROM mytab;
a |   b
---+-------
1 | apple
(1 row)
```

6. You can add, change, and remove data from the MySQL table. For example:

```
test=> INSERT INTO mytab values (2, 'mango');
INSERT 0 1
```

Run the SELECT query again to see the results:

```
test=> SELECT * FROM mytab ORDER BY 1;
 a |   b
---+-------
1 | apple
2 | mango
```

```
(2 rows)
```

Using encryption in transit with the extension

The connection to MySQL from RDS for PostgreSQL uses encryption in transit (TLS/SSL) by default. However, the connection falls back to non-encrypted when the client and server configuration differ. You can enforce encryption for all outgoing connections by specifying the `REQUIRE SSL` option on the RDS for MySQL user accounts. This same approach also works for MariaDB and Aurora MySQL user accounts.

For MySQL user accounts configured to `REQUIRE SSL`, the connection attempt fails if a secure connection can't be established.

To enforce encryption for existing MySQL database user accounts, you can use the `ALTER USER` command. The syntax varies, depending on the MySQL version, as shown in the following table. For more information, see ALTER USER in *MySQL Reference Manual*.

MySQL 5.7, MySQL 8.0	MySQL 5.6
`ALTER USER 'user'@'%' REQUIRE SSL;`	`GRANT USAGE ON *.* to 'user'@'%' REQUIRE SSL;`

For more information about the `mysql_fdw` extension, see the mysql_fdw documentation.

Working with Oracle databases by using the oracle_fdw extension

To access an Oracle database from your RDS for PostgreSQL DB instance you can install and use the `oracle_fdw` extension. This extension is a foreign data wrapper for Oracle databases. To learn more about this extension, see the oracle_fdw documentation.

The `oracle_fdw` extension is supported on RDS for PostgreSQL 12.7, 13.3, and higher versions.

Topics

- Turning on the oracle_fdw extension
- Example: Using a foreign server linked to an Amazon RDS for Oracle database
- Working with encryption in transit
- Understanding the pg_user_mappings view and permissions

Turning on the oracle_fdw extension

To use the oracle_fdw extension, perform the following procedure.

To turn on the oracle_fdw extension

- Run the following command using an account that has `rds_superuser` permissions.

```
CREATE EXTENSION oracle_fdw;
```

Example: Using a foreign server linked to an Amazon RDS for Oracle database

The following example shows the use of a foreign server linked to an Amazon RDS for Oracle database.

To create a foreign server linked to an RDS for Oracle database

1. Note the following on the RDS for Oracle DB instance:
   - Endpoint
   - Port
   - Database name
2. Create a foreign server.

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
 '//endpoint:port/DB_name');
CREATE SERVER
```

3. Grant usage to a user who doesn't have `rds_superuser` privileges, for example `user1`.

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. Connect as `user1`, and create a mapping to an Oracle user.

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser',
 password 'mypassword');
CREATE USER MAPPING
```

5. Create a foreign table linked to an Oracle table.

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
CREATE FOREIGN TABLE
```

6. Query the foreign table.

```
test=>  SELECT * FROM mytab;
a
---
1
(1 row)
```

If the query reports the following error, check your security group and access control list (ACL) to make sure that both instances can communicate.

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

Working with encryption in transit

PostgreSQL-to-Oracle encryption in transit is based on a combination of client and server configuration parameters. For an example using Oracle 21c, see About the Values for Negotiating Encryption and Integrity in the Oracle documentation. The client used for oracle_fdw on Amazon RDS is configured with ACCEPTED, meaning that the encryption depends on the Oracle database server configuration and it uses Oracle Security Library (libnnz) for encryption.

If your database is on RDS for Oracle, see Oracle native network encryption to configure the encryption.

Understanding the pg_user_mappings view and permissions

The PostgreSQL catalog `pg_user_mapping` stores the mapping from an RDS for PostgreSQL user to the user on a foreign data (remote) server. Access to the catalog is restricted, but you use the `pg_user_mappings` view to see the mappings. In the following, you can find an example that shows how permissions apply with an example Oracle database, but this information applies more generally to any foreign data wrapper.

In the following output, you can find roles and permissions mapped to three different example users. Users `rdssu1` and `rdssu2` are members of the `rds_superuser` role, and `user1` isn't. The example uses the `psql` metacommand `\du` to list existing roles.

```
test=>  \du
                                                              List of roles
    Role name    |                          Attributes                          |
                 Member of
-----------------+------------------------------------------------------------
+-----------------------------------------------------------
 rdssu1          |                                                              |
 {rds_superuser}
 rdssu2          |                                                              |
 {rds_superuser}
 user1           |                                                              | {}
```

All users, including users that have `rds_superuser` privileges, are allowed to view their own user mappings (`umoptions`) in the `pg_user_mappings` table. As shown in the following example, when `rdssu1` tries to obtain all user mappings, an error is raised even though `rdssu1rds_superuser` privileges:

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

Following are some examples.

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     | {user=oracleuser,password=mypwd}
 (3 rows)
```

```
test=> SET SESSION AUTHORIZATION user1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |           umoptions
-------+-------+---------+--------+------------+-------------------------------
 16414 | 16411 | oradb   |  16412 | user1      | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)
```

Because of implementation differences between `information_schema._pg_user_mappings` and `pg_catalog.pg_user_mappings`, a manually created `rds_superuser` requires additional permissions to view passwords in `pg_catalog.pg_user_mappings`.

No additional permissions are required for an `rds_superuser` to view passwords in `information_schema._pg_user_mappings`.

Users who don't have the `rds_superuser` role can view passwords in `pg_user_mappings` only under the following conditions:

- The current user is the user being mapped and owns the server or holds the `USAGE` privilege on it.
- The current user is the server owner and the mapping is for `PUBLIC`.

Working with SQL Server databases by using the tds_fdw extension

You can use the PostgreSQL `tds_fdw` extension to access databases that support the tabular data stream (TDS) protocol, such as Sybase and Microsoft SQL Server databases. This foreign data wrapper lets you connect from your RDS for PostgreSQL DB instance to databases that use the TDS protocol, including Amazon RDS for Microsoft SQL Server. For more information, see tds-fdw/tds_fdw documentation on GitHub.

The `tds_fdw` extension is supported on Amazon RDS for PostgreSQL version 14.2, 13.6, and higher releases.

Setting up your Aurora PostgreSQL DB to use the tds_fdw extension

In the following procedures, you can find an example of setting up and using the `tds_fdw` with an RDS for PostgreSQL DB instance. Before you can connect to a SQL Server database using `tds_fdw`, you need to get the following details for the instance:

- Hostname or endpoint. For an RDS for SQL Server DB instance, you can find the endpoint by using the Console. Choose the Connectivity & security tab and look in the "Endpoint and port" section.
- Port number. The default port number for Microsoft SQL Server is 1433.
- Name of the database. The DB identifier.

You also need to provide access on the security group or the access control list (ACL) for the SQL Server port, 1433. Both the RDS for PostgreSQL DB instance and the RDS for SQL Server DB instance need access to port 1433. If access isn't configured correctly, when you try to query the Microsoft SQL Server you see the following error message:

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:
Adaptive Server is unavailable or does not exist (mssql2019.aws-
region.rds.amazonaws.com), OS #: 0, OS Msg: Success, Level: 9
```

To use tds_fdw to connect to a SQL Server database

1. Connect to your PostgreSQL DB instance using an account that has the `rds_superuser` role:

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=test --password
   ```

2. Install the `tds_fdw` extension:

   ```
   test=> CREATE EXTENSION tds_fdw;
   CREATE EXTENSION
   ```

After the extension is installed on your RDS for PostgreSQL DB instance, you set up the foreign server.

To create the foreign server

Perform these tasks on the RDS for PostgreSQL DB instance using an account that has `rds_superuser` privileges.

1. Create a foreign server in the RDS for PostgreSQL DB instance:

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS
 (servername 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database
 'tds_fdw_testing');
CREATE SERVER
```

To access non-ASCII data on the SQLServer side, create a server link with the character_set option in the RDS for PostgreSQL DB instance:

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername
 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds_fdw_testing',
 character_set 'UTF-8');
CREATE SERVER
```

2. Grant permissions to a user who doesn't have `rds_superuser` role privileges, for example, `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. Connect as user1 and create a mapping to a SQL Server user:

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username
 'sqlserveruser', password 'password');
CREATE USER MAPPING
```

4. Create a foreign table linked to a SQL Server table:

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table
 'MYTABLE');
CREATE FOREIGN TABLE
```

5. Query the foreign table:

```
test=> SELECT * FROM mytab;
 a
---
 1
(1 row)
```

Using encryption in transit for the connection

The connection from RDS for PostgreSQL to SQL Server uses encryption in transit (TLS/SSL) depending on the SQL Server database configuration. If the SQL Server isn't configured for encryption, the RDS for PostgreSQL client making the request to the SQL Server database falls back to unencrypted.

You can enforce encryption for the connection to RDS for SQL Server DB instances by setting the `rds.force_ssl` parameter. To learn how, see Forcing connections to your DB instance to use SSL. For more information about SSL/TLS configuration for RDS for SQL Server, see Using SSL with a Microsoft SQL Server DB instance.

Working with Trusted Language Extensions for PostgreSQL

Trusted Language Extensions for PostgreSQL is an open source development kit for building PostgreSQL extensions. It allows you to build high performance PostgreSQL extensions and safely run them on your RDS for PostgreSQL DB instance. By using Trusted Language Extensions (TLE) for PostgreSQL, you can create PostgreSQL extensions that follow the documented approach for extending PostgreSQL functionality. For more information, see Packaging Related Objects into an Extension in the PostgreSQL documentation.

One key benefit of TLE is that you can use it in environments that don't provide access to the file system underlying the PostgreSQL instance. Previously, installing a new extension required access to the file system. TLE removes this constraint. It provides a development environment for creating new extensions for any PostgreSQL database, including those running on your RDS for PostgreSQL DB instances.

TLE is designed to prevent access to unsafe resources for the extensions that you create using TLE. Its runtime environment limits the impact of any extension defect to a single database connection. TLE also gives database administrators fine-grained control over who can install extensions, and it provides a permissions model for running them.

TLE is supported on the following RDS for PostgreSQL versions:

- Version 18.1 and higher 18 versions
- Version 17.1 and higher 17 versions
- Version 16.1 and higher 16 versions
- Version 15.2 and higher 15 versions
- Version 14.5 and higher 14 versions
- Version 13.12 and higher 13 versions

The Trusted Language Extensions development environment and runtime are packaged as the `pg_tle` PostgreSQL extension, version 1.0.1. It supports creating extensions in JavaScript, Perl, Tcl, PL/pgSQL, and SQL. You install the `pg_tle` extension in your RDS for PostgreSQL DB instance in the same way that you install other PostgreSQL extensions. After the `pg_tle` is set up, developers can use it to create new PostgreSQL extensions, known as *TLE extensions*.

In the following topics, you can find information about how to set up Trusted Language Extensions and how to get started creating your own TLE extensions.

Topics

- Terminology
- Requirements for using Trusted Language Extensions for PostgreSQL
- Setting up Trusted Language Extensions in your RDS for PostgreSQL DB instance
- Overview of Trusted Language Extensions for PostgreSQL
- Creating TLE extensions for RDS for PostgreSQL
- Dropping your TLE extensions from a database
- Uninstalling Trusted Language Extensions for PostgreSQL
- Using PostgreSQL hooks with your TLE extensions
- Using Custom Data Types in TLE
- Function reference for Trusted Language Extensions for PostgreSQL
- Hooks reference for Trusted Language Extensions for PostgreSQL

Terminology

To help you better understand Trusted Language Extensions, view the following glossary for terms used in this topic.

Trusted Language Extensions for PostgreSQL

Trusted Language Extensions for PostgreSQL is the official name of the open source development kit that's packaged as the `pg_tle` extension. It's available for use on any PostgreSQL system. For more information, see aws/pg_tle on GitHub.

Trusted Language Extensions

Trusted Language Extensions is the short name for Trusted Language Extensions for PostgreSQL. This shortened name and its abbreviation (TLE) are also used in this documentation.

trusted language

A *trusted language* is a programming or scripting language that has specific security attributes. For example, trusted languages typically restrict access to the file system, and they limit use of specified networking properties. The TLE development kit is designed to support trusted languages. PostgreSQL supports several different languages that are used to create

trusted or untrusted extensions. For an example, see Trusted and Untrusted PL/Perl in the PostgreSQL documentation. When you create an extension using Trusted Language Extensions, the extension inherently uses trusted language mechanisms.

TLE extension

A *TLE extension* is a PostgreSQL extension that's been created by using the Trusted Language Extensions (TLE) development kit.

Requirements for using Trusted Language Extensions for PostgreSQL

The following are requirements for setting up and using the TLE development kit.

- **RDS for PostgreSQL versions** – Trusted Language Extensions is supported on RDS for PostgreSQL versions 13.12 and higher 13 versions, 14.5 and higher 14 versions, and 15.2 and higher versions only.
  - If you need to upgrade your RDS for PostgreSQL instance, see Upgrades of the RDS for PostgreSQL DB engine.
  - If you don't yet have an Amazon RDS DB instance running PostgreSQL, you can create one. For more information, see RDS for PostgreSQL DB instance, see Creating and connecting to a PostgreSQL DB instance.
- **Requires `rds_superuser` privileges** – To set up and configure the `pg_tle` extension, your database user role must have the permissions of the `rds_superuser` role. By default, this role is granted to the `postgres` user that creates the RDS for PostgreSQL DB instance.
- **Requires a custom DB parameter group** – Your RDS for PostgreSQL DB instance must be configured with a custom DB parameter group.
  - If your RDS for PostgreSQL DB instance isn't configured with a custom DB parameter group, you should create one and associate it with your RDS for PostgreSQL DB instance. For a short summary of steps, see Creating and applying a custom DB parameter group.
  - If your RDS for PostgreSQL DB instance is already configured using a custom DB parameter group, you can set up Trusted Language Extensions. For details, see Setting up Trusted Language Extensions in your RDS for PostgreSQL DB instance.

Creating and applying a custom DB parameter group

Use the following steps to create a custom DB parameter group and configure your RDS for PostgreSQL DB instance to use it.

Console

To create a custom DB parameter group and use it with your RDS for PostgreSQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. Choose Parameter groups from the Amazon RDS menu.
3. Choose **Create parameter group**.
4. In the **Parameter group details** page, enter the following information.
   - For **Parameter group family**, choose postgres14.
   - For **Type**, choose DB Parameter Group.
   - For **Group name**, give your parameter group a meaningful name in the context of your operations.
   - For **Description**, enter a useful description so that others on your team can easily find it.
5. Choose **Create**. Your custom DB parameter group is created in your AWS Region. You can now modify your RDS for PostgreSQL DB instance to use it by following the next steps.
6. Choose **Databases** from the Amazon RDS menu.
7. Choose the RDS for PostgreSQL DB instance that you want to use with TLE from among those listed, and then choose **Modify.**
8. In the Modify DB instance settings page, find **Database options** in the Additional configuration section and choose your custom DB parameter group from the selector.
9. Choose **Continue** to save the change.
10. Choose **Apply immediately** so that you can continue setting up the RDS for PostgreSQL DB instance to use TLE.

To continue setting up your system for Trusted Language Extensions, see Setting up Trusted Language Extensions in your RDS for PostgreSQL DB instance.

For more information working with DB parameter groups, see DB parameter groups for Amazon RDS DB instances.

AWS CLI

You can avoid specifying the `--region` argument when you use CLI commands by configuring your AWS CLI with your default AWS Region. For more information, see Configuration basics in the *AWS Command Line Interface User Guide*.

To create a custom DB parameter group and use it with your RDS for PostgreSQL DB instance

1. Use the create-db-parameter-group AWS CLI command to create a custom DB parameter group based on postgres14 for your AWS Region.

   For Linux, macOS, or Unix:

   ```
   aws rds create-db-parameter-group \
     --region aws-region \
     --db-parameter-group-name custom-params-for-pg-tle \
     --db-parameter-group-family postgres14 \
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   For Windows:

   ```
   aws rds create-db-parameter-group ^
     --region aws-region ^
     --db-parameter-group-name custom-params-for-pg-tle ^
     --db-parameter-group-family postgres14 ^
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   Your custom DB parameter group is available in your AWS Region, so you can modify RDS for PostgreSQL DB instance to use it.

2. Use the modify-db-instance AWS CLI command to apply your custom DB parameter group to your RDS for PostgreSQL DB instance. This command immediately reboots the active instance.

   For Linux, macOS, or Unix:

   ```
   aws rds modify-db-instance \
     --region aws-region \
     --db-instance-identifier your-instance-name \
     --db-parameter-group-name custom-params-for-pg-tle \
     --apply-immediately
   ```

For Windows:

```
aws rds modify-db-instance ^
  --region aws-region ^
  --db-instance-identifier your-instance-name ^
  --db-parameter-group-name custom-params-for-pg-tle ^
  --apply-immediately
```

To continue setting up your system for Trusted Language Extensions, see Setting up Trusted Language Extensions in your RDS for PostgreSQL DB instance.

For more information, see Parameter groups for Amazon RDS.

Setting up Trusted Language Extensions in your RDS for PostgreSQL DB instance

The following steps assume that your RDS for PostgreSQL DB instance is associated with a custom DB parameter group. You can use the AWS Management Console or the AWS CLI for these steps.

When you set up Trusted Language Extensions in your RDS for PostgreSQL DB instance, you install it in a specific database for use by the database users who have permissions on that database.

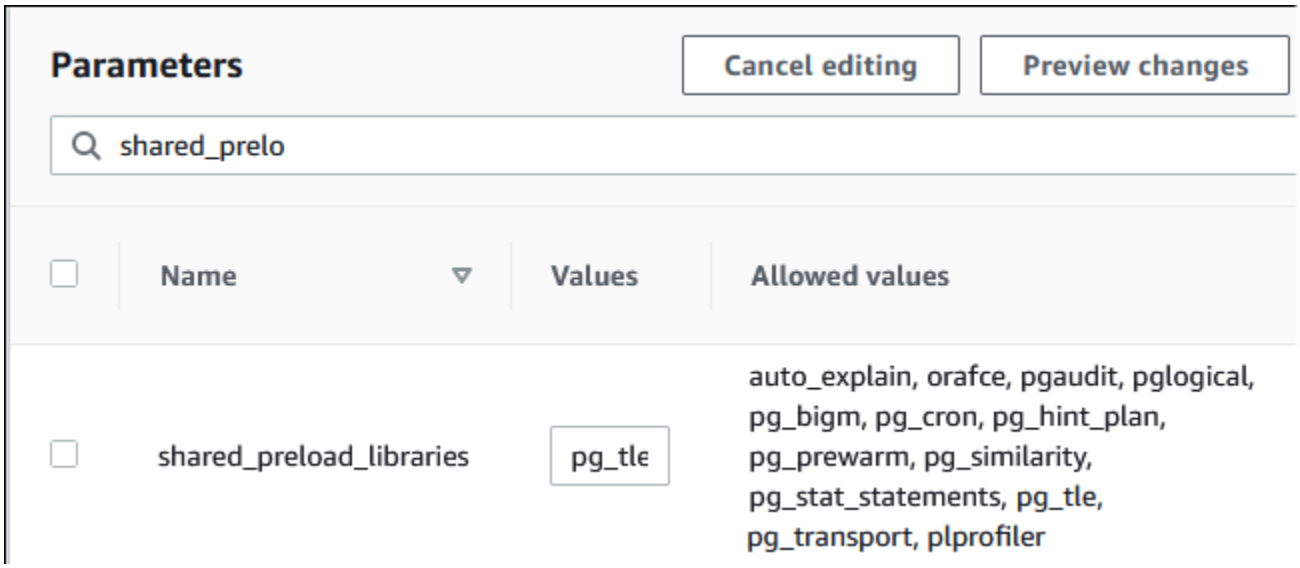
Console

To set up Trusted Language Extensions

Perform the following steps using an account that's a member of the `rds_superuser` group (role).

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose your RDS for PostgreSQL DB instance.
3. Open the **Configuration** tab for your RDS for PostgreSQL DB instance. Among the Instance details, find the **Parameter group** link.
4. Choose the link to open the custom parameters associated with your RDS for PostgreSQL DB instance.
5. In the **Parameters** search field, type `shared_pre` to find the `shared_preload_libraries` parameter.

6. Choose **Edit parameters** to access the property values.
7. Add `pg_tle` to the list in the **Values** field. Use a comma to separate items in the list of values.



8. Reboot the RDS for PostgreSQL DB instance so that your change to the `shared_preload_libraries` parameter takes effect.
9. When the instance is available, verify that `pg_tle` has been initialized. Use `psql` to connect to the RDS for PostgreSQL DB instance, and then run the following command.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

10. With the `pg_tle` extension initialized, you can now create the extension.

```
CREATE EXTENSION pg_tle;
```

You can verify that the extension is installed by using the following `psql` metacommand.

```
labdb=> \dx
                          List of installed extensions
  Name   | Version |   Schema   |                Description
---------+---------+------------+-------------------------------------------
 pg_tle  | 1.0.1   | pgtle      | Trusted-Language Extensions for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

11. Grant the `pgtle_admin` role to the primary user name that you created for your RDS for PostgreSQL DB instance when you set it up. If you accepted the default, it's `postgres`.

```
labdb=> GRANT pgtle_admin TO postgres;
GRANT ROLE
```

You can verify that the grant has occurred by using the `psql` metacommand as shown in the following example. Only the `pgtle_admin` and `postgres` roles are shown in the output. For more information, see Understanding the rds_superuser role.

```
labdb=> \du
                          List of roles
    Role name    |           Attributes            |               Member of
-----------------+---------------------------------
+-----------------------------------
pgtle_admin     | Cannot login                     | {}
postgres        | Create role, Create DB          +| {rds_superuser,pgtle_admin}
                | Password valid until infinity    |...
```

12. Close the `psql` session using the `\q` metacommand.

```
\q
```

To get started creating TLE extensions, see Example: Creating a trusted language extension using SQL.

AWS CLI

You can avoid specifying the `--region` argument when you use CLI commands by configuring your AWS CLI with your default AWS Region. For more information, see Configuration basics in the *AWS Command Line Interface User Guide*.

To set up Trusted Language Extensions

1. Use the modify-db-parameter-group AWS CLI command to add `pg_tle` to the `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
```

```
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_tle,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Use the reboot-db-instance AWS CLI command to reboot the RDS for PostgreSQL DB instance and initialize the `pg_tle` library.

```
aws rds reboot-db-instance \
    --db-instance-identifier your-instance \
    --region aws-region
```

3. When the instance is available, you can verify that `pg_tle` has been initialized. Use `psql` to connect to the RDS for PostgreSQL DB instance, and then run the following command.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

   With `pg_tle` initialized, you can now create the extension.

```
CREATE EXTENSION pg_tle;
```

4. Grant the `pgtle_admin` role to the primary user name that you created for your RDS for PostgreSQL DB instance when you set it up. If you accepted the default, it's `postgres`.

```
GRANT pgtle_admin TO postgres;
GRANT ROLE
```

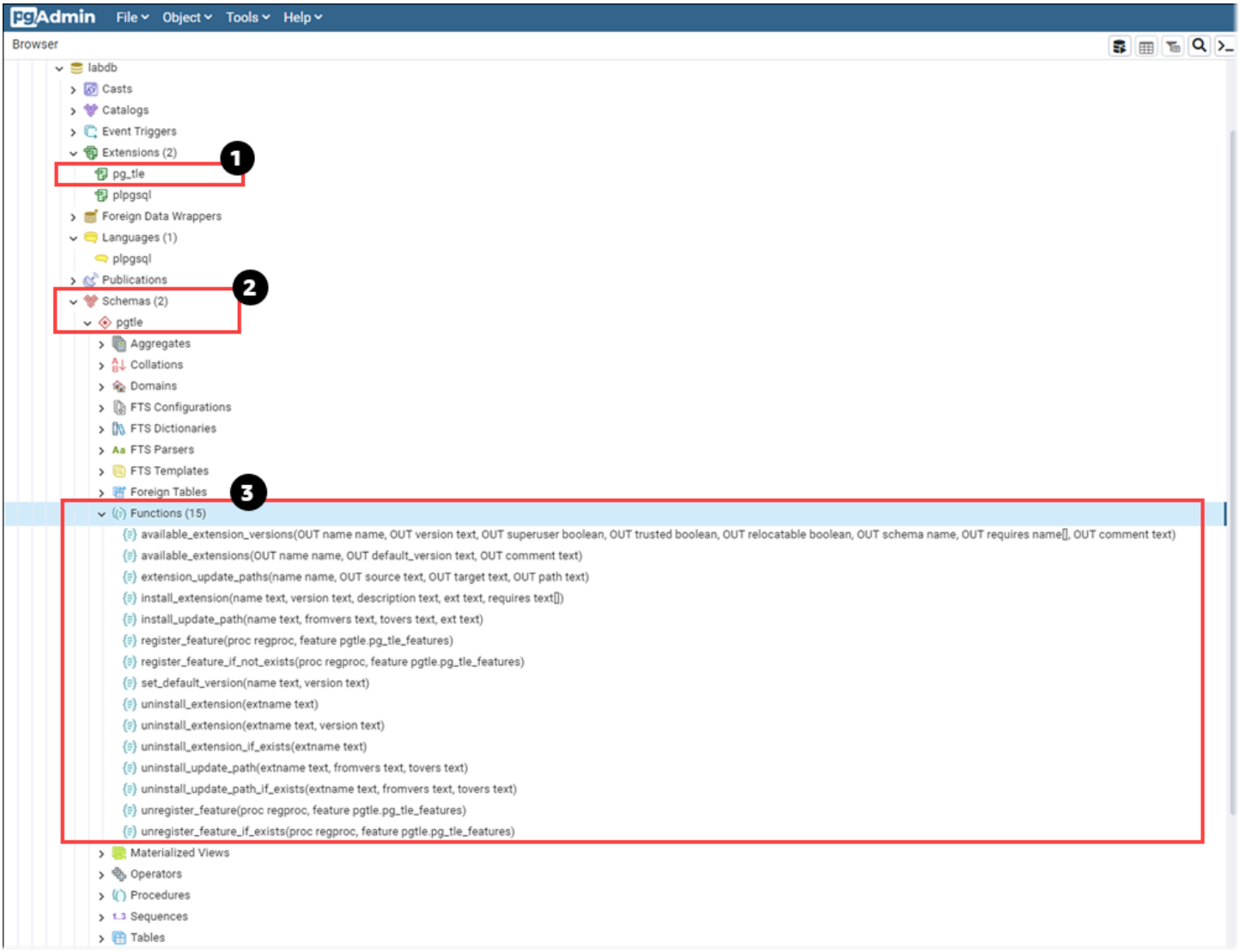
5. Close the `psql` session as follows.

```
labdb=> \q
```

To get started creating TLE extensions, see Example: Creating a trusted language extension using SQL.

Overview of Trusted Language Extensions for PostgreSQL

Trusted Language Extensions for PostgreSQL is a PostgreSQL extension that you install in your RDS for PostgreSQL DB instance in the same way that you set up other PostgreSQL extensions. In the following image of an example database in the pgAdmin client tool, you can view some of the components that comprise the `pg_tle` extension.



You can see the following details.

1. The Trusted Language Extensions (TLE) for PostgreSQL development kit is packaged as the `pg_tle` extension. As such, `pg_tle` is added to the available extensions for the database in which it's installed.
2. TLE has its own schema, `pgtle`. This schema contains helper functions (3) for installing and managing the extensions that you create.

3. TLE provides over a dozen helper functions for installing, registering, and managing your extensions. To learn more about these functions, see Function reference for Trusted Language Extensions for PostgreSQL.

Other components of the `pg_tle` extension include the following:

- **The `pgtle_admin` role** – The `pgtle_admin` role is created when the `pg_tle` extension is installed. This role is privileged and should be treated as such. We strongly recommend that you follow the principle of *least privilege* when granting the `pgtle_admin` role to database users. In other words, grant the `pgtle_admin` role only to database users that are allowed to create, install, and manage new TLE extensions, such as `postgres`.
- **The `pgtle.feature_info` table** – The `pgtle.feature_info` table is a protected table that contains information about your TLEs, hooks, and the custom stored procedures and functions that they use. If you have `pgtle_admin` privileges, you use the following Trusted Language Extensions functions to add and update that information in the table.
  - pgtle.register_feature
  - pgtle.register_feature_if_not_exists
  - pgtle.unregister_feature
  - pgtle.unregister_feature_if_exists

Creating TLE extensions for RDS for PostgreSQL

You can install any extensions that you create with TLE in any RDS for PostgreSQL DB instance that has the `pg_tle` extension installed. The `pg_tle` extension is scoped to the PostgreSQL database in which it's installed. The extensions that you create using TLE are scoped to the same database.

Use the various `pgtle` functions to install the code that makes up your TLE extension. The following Trusted Language Extensions functions all require the `pgtle_admin` role.

- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)

- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

Example: Creating a trusted language extension using SQL

The following example shows you how to create a TLE extension named `pg_distance` that contains a few SQL functions for calculating distances using different formulas. In the listing, you can find the function for calculating the Manhattan distance and the function for calculating the Euclidean distance. For more information about the difference between these formulas, see Taxicab geometry and Euclidean geometry in Wikipedia.

You can use this example in your own RDS for PostgreSQL DB instance if you have the `pg_tle` extension set up as detailed in Setting up Trusted Language Extensions in your RDS for PostgreSQL DB instance.

> **Note**
>
> You need to have the privileges of the `pgtle_admin` role to follow this procedure.

To create the example TLE extension

The following steps use an example database named `labdb`. This database is owned by the `postgres` primary user. The `postgres` role also has the permissions of the `pgtle_admin` role.

1. Use `psql` to connect to RDS for PostgreSQL DB instance.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Create a TLE extension named `pg_distance` by copying the following code and pasting it into your `psql` session console.

```
SELECT pgtle.install_extension
```

```
(
 'pg_distance',
 '0.1',
  'Distance functions for two points',
$_pg_tle_$
    CREATE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8, norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL;

    CREATE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL;

    CREATE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL;
$_pg_tle_$
);
```

You see the output, such as the following.

```
install_extension
---------------
 t
(1 row)
```

The artifacts that make up the `pg_distance` extension are now installed in your database. These artifacts include the control file and the code for the extension, which are items that need to be present so that the extension can be created using the `CREATE EXTENSION` command. In other words, you still need to create the extension to make its functions available to database users.

3. To create the extension, use the `CREATE EXTENSION` command as you do for any other extension. As with other extensions, the database user needs to have the `CREATE` permissions in the database.

```
CREATE EXTENSION pg_distance;
```

4. To test the pg_distance TLE extension, you can use it to calculate the Manhattan distance between four points.

```
labdb=> SELECT manhattan_dist(1, 1, 5, 5);
8
```

To calculate the Euclidean distance between the same set of points, you can use the following.

```
labdb=> SELECT euclidean_dist(1, 1, 5, 5);
5.656854249492381
```

The pg_distance extension loads the functions in the database and makes them available to any users with permissions on the database.

Modifying your TLE extension

To improve query performance for the functions packaged in this TLE extension, add the following two PostgreSQL attributes to their specifications.

- IMMUTABLE – The IMMUTABLE attribute ensures that the query optimizer can use optimizations to improve query response times. For more information, see Function Volatility Categories in the PostgreSQL documentation.
- PARALLEL SAFE – The PARALLEL SAFE attribute is another attribute that allows PostgreSQL to run the function in parallel mode. For more information, see CREATE FUNCTION in the PostgreSQL documentation.

In the following example, you can see how the pgtle.install_update_path function is used to add these attributes to each function to create a version 0.2 of the pg_distance TLE extension. For more information about this function, see pgtle.install_update_path. You need to have the pgtle_admin role to perform this task.

To update an existing TLE extension and specify the default version

1. Connect to RDS for PostgreSQL DB instance using psql or another client tool, such as pgAdmin.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Modify the existing TLE extension by copying the following code and pasting it into your `psql` session console.

```
SELECT pgtle.install_update_path
(
 'pg_distance',
 '0.1',
 '0.2',
$_pg_tle_$
    CREATE OR REPLACE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8,
 norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;
$_pg_tle_$
);
```

You see a response similar to the following.

```
install_update_path
---------------------
 t
(1 row)
```

You can make this version of the extension the default version, so that database users don't have to specify a version when they create or update the extension in their database.

3. To specify that the modified version (version 0.2) of your TLE extension is the default version, use the `pgtle.set_default_version` function as shown in the following example.

```
SELECT pgtle.set_default_version('pg_distance', '0.2');
```

For more information about this function, see pgtle.set_default_version.

4. With the code in place, you can update the installed TLE extension in the usual way, by using `ALTER EXTENSION ... UPDATE` command, as shown here:

```
ALTER EXTENSION pg_distance UPDATE;
```

Dropping your TLE extensions from a database

You can drop your TLE extensions by using the `DROP EXTENSION` command in the same way that you do for other PostgreSQL extensions. Dropping the extension doesn't remove the installation files that make up the extension, which allows users to re-create the extension. To remove the extension and its installation files, do the following two-step process.

To drop the TLE extension and remove its installation files

1. Use `psql` or another client tool to connect to the RDS for PostgreSQL DB instance.

```
psql --host=.111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=dbname
```

2. Drop the extension as you would any PostgreSQL extension.

```
DROP EXTENSION your-TLE-extension
```

For example, if you create the `pg_distance` extension as detailed in Example: Creating a trusted language extension using SQL, you can drop the extension as follows.

```
DROP EXTENSION pg_distance;
```

You see output confirming that the extension has been dropped, as follows.

```
DROP EXTENSION
```

At this point, the extension is no longer active in the database. However, its installation files and control file are still available in the database, so database users can create the extension again if they like.

- If you want to leave the extension files intact so that database users can create your TLE extension, you can stop here.
- If you want to remove all files that make up the extension, continue to the next step.

3. To remove all installation files for your extension, use the `pgtle.uninstall_extension` function. This function removes all the code and control files for your extension.

```
SELECT pgtle.uninstall_extension('your-tle-extension-name');
```

For example, to remove all `pg_distance` installation files, use the following command.

```
SELECT pgtle.uninstall_extension('pg_distance');
 uninstall_extension
---------------------
 t
(1 row)
```

Uninstalling Trusted Language Extensions for PostgreSQL

If you no longer want to create your own TLE extensions using TLE, you can drop the `pg_tle` extension and remove all artifacts. This action includes dropping any TLE extensions in the database and dropping the `pgtle` schema.

To drop the `pg_tle` extension and its schema from a database

1. Use `psql` or another client tool to connect to the RDS for PostgreSQL DB instance.

```
psql --host=.111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=dbname
```

2. Drop the pg_tle extension from the database. If the database has your own TLE extensions still running in the database, you need to also drop those extensions. To do so, you can use the CASCADE keyword, as shown in the following.

```
DROP EXTENSION pg_tle CASCADE;
```

   If the pg_tle extension isn't still active in the database, you don't need to use the CASCADE keyword.

3. Drop the pgtle schema. This action removes all the management functions from the database.

```
DROP SCHEMA pgtle CASCADE;
```

   The command returns the following when the process completes.

```
DROP SCHEMA
```

   The pg_tle extension, its schema and functions, and all artifacts are removed. To create new extensions using TLE, go through the setup process again. For more information, see Setting up Trusted Language Extensions in your RDS for PostgreSQL DB instance.

Using PostgreSQL hooks with your TLE extensions

A *hook* is a callback mechanism available in PostgreSQL that allows developers to call custom functions or other routines during regular database operations. The TLE development kit supports PostgreSQL hooks so that you can integrate custom functions with PostgreSQL behavior at runtime. For example, you can use a hook to associate the authentication process with your own custom code, or to modify the query planning and execution process for your specific needs.

Your TLE extensions can use hooks. If a hook is global in scope, it applies across all databases. Therefore, if your TLE extension uses a global hook, then you need to create your TLE extension in all databases that your users can access.

When you use the pg_tle extension to build your own Trusted Language Extensions, you can use the available hooks from a SQL API to build out the functions of your extension. You should register any hooks with pg_tle. For some hooks, you might also need to set various configuration parameters. For example, the passcode check hook can be set to on, off, or require. For more

information about specific requirements for available pg_tle hooks, see Hooks reference for Trusted Language Extensions for PostgreSQL.

Example: Creating an extension that uses a PostgreSQL hook

The example discussed in this section uses a PostgreSQL hook to check the password provided during specific SQL operations and prevents database users from setting their passwords to any of those contained in the password_check.bad_passwords table. The table contains the top-ten most commonly used, but easily breakable choices for passwords.

To set up this example in your RDS for PostgreSQL DB instance, you must have already installed Trusted Language Extensions. For details, see Setting up Trusted Language Extensions in your RDS for PostgreSQL DB instance.

To set up the password-check hook example

1. Use psql to connect to RDS for PostgreSQL DB instance.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Copy the code from the Password-check hook code listing and paste it into your database.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
```

```
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

When the extension has been loaded into your database, you see the output such as the following.

```
 install_extension
-------------------
 t
(1 row)
```

3. While still connected to the database, you can now create the extension.

```
CREATE EXTENSION my_password_check_rules;
```

4. You can confirm that the extension has been created in the database by using the following `psql` metacommand.

```
\dx
                        List of installed extensions
          Name           | Version |   Schema   |
 Description
------------------------+---------+------------
+-------------------------------------------------------------
 my_password_check_rules | 1.0     | public     | Prevent use of any of the top-ten
 most common bad passwords
 pg_tle                  | 1.0.1   | pgtle      | Trusted-Language Extensions for
 PostgreSQL
 plpgsql                 | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

5. Open another terminal session to work with the AWS CLI. You need to modify your custom DB parameter group to turn on the password-check hook. To do so, use the modify-db-parameter-group CLI command as shown in the following example.

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

When the parameter is successfully turned on, you see the output such as the following.

```
(
    "DBParameterGroupName": "docs-lab-parameters-for-tle"
}
```

It might take a few minutes for the change to the parameter group setting to take effect. This parameter is dynamic, however, so you don't need to restart the RDS for PostgreSQL DB instance for the setting to take effect.

6. Open the `psql` session and query the database to verify that the password_check hook has been turned on.

```
labdb=> SHOW pgtle.enable_password_check;
pgtle.enable_password_check
----------------------------
on
(1 row)
```

The password-check hook is now active. You can test it by creating a new role and using one of the bad passwords, as shown in the following example.

```
CREATE ROLE test_role PASSWORD 'password';
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 21 at RAISE
SQL statement "SELECT password_check.passcheck_hook(
    $1::pg_catalog.text,
    $2::pg_catalog.text,
    $3::pgtle.password_types,
    $4::pg_catalog.timestamptz,
    $5::pg_catalog.bool)"
```

The output has been formatted for readability.

The following example shows that `pgsql` interactive metacommand `\password` behavior is also affected by the password_check hook.

```
postgres=> SET password_encryption TO 'md5';
SET
postgres=> \password
Enter new password for user "postgres":*****
Enter it again:*****
ERROR:  Cannot use passwords from the common password dictionary
```

```
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 12 at RAISE
SQL statement "SELECT password_check.passcheck_hook($1::pg_catalog.text,
 $2::pg_catalog.text, $3::pgtle.password_types, $4::pg_catalog.timestamptz,
 $5::pg_catalog.bool)"
```

You can drop this TLE extension and uninstall its source files if you want. For more information, see Dropping your TLE extensions from a database.

Password-check hook code listing

The example code shown here defines the specification for the `my_password_check_rules` TLE extension. When you copy this code and paste it into your database, the code for the `my_password_check_rules` extension is loaded into the database, and the `password_check` hook is registered for use by the extension.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
```

```
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

Using Custom Data Types in TLE

PostgreSQL supports commands to register new base types (also known as scalar types) for efficiently handling complex data structures in your database. A base type allows you to customize how the data is stored internally, and how to convert it to and from an external textual representation. These custom data types are helpful when extending PostgreSQL to support functional domains where a built-in type such as number or text can't provide sufficient search semantics.

RDS for PostgreSQL enables you to create custom data types in your trusted language extension and define functions that support SQL and index operations for these new data types. Custom data types are available for the following versions:

- RDS for PostgreSQL 15.4 and higher 15 versions
- RDS for PostgreSQL 14.9 and higher 14 versions
- RDS for PostgreSQL 13.12 and higher 13 versions

For more information, see Trusted Language Base types.

Function reference for Trusted Language Extensions for PostgreSQL

View the following reference documentation about functions available in Trusted Language Extensions for PostgreSQL. Use these functions to install, register, update, and manage your *TLE extensions*, that is, the PostgreSQL extensions that you develop using the Trusted Language Extensions development kit.

Functions

- pgtle.available_extensions
- pgtle.available_extension_versions
- pgtle.extension_update_paths
- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

pgtle.available_extensions

The `pgtle.available_extensions` function is a set-returning function. It returns all available TLE extensions in the database. Each returned row contains information about a single TLE extension.

Function prototype

```
pgtle.available_extensions()
```

Role

None.

Arguments

None.

Output

- `name` – The name of the TLE extension.
- `default_version` – The version of the TLE extension to use when `CREATE EXTENSION` is called without a version specified.
- `description` – A more detailed description about the TLE extension.

Usage example

```
SELECT * FROM pgtle.available_extensions();
```

pgtle.available_extension_versions

The `available_extension_versions` function is a set-returning function. It returns a list of all available TLE extensions and their versions. Each row contains information about a specific version of the given TLE extension, including whether it requires a specific role.

Function prototype

```
pgtle.available_extension_versions()
```

Role

None.

Arguments

None.

Output

- `name` – The name of the TLE extension.
- `version` – The version of the TLE extension.
- `superuser` – This value is always `false` for your TLE extensions. The permissions needed to create the TLE extension or update it are the same as for creating other objects in the given database.
- `trusted` – This value is always `false` for a TLE extension.
- `relocatable` – This value is always `false` for a TLE extension.
- `schema` – Specifies the name of the schema in which the TLE extension is installed.
- `requires` – An array containing the names of other extensions needed by this TLE extension.
- `description` – A detailed description of the TLE extension.

For more information about output values, see Packaging Related Objects into an Extension > Extension Files in the PostgreSQL documentation.

Usage example

```
SELECT * FROM pgtle.available_extension_versions();
```

pgtle.extension_update_paths

The `extension_update_paths` function is a set-returning function. It returns a list of all the possible update paths for a TLE extension. Each row includes the available upgrades or downgrades for that TLE extension.

Function prototype

```
pgtle.extension_update_paths(name)
```

Role

None.

Arguments

name – The name of the TLE extension from which to get upgrade paths.

Output

- source – The source version for an update.
- target – The target version for an update.
- path – The upgrade path used to update a TLE extension from source version to target version, for example, 0.1--0.2.

Usage example

```
SELECT * FROM pgtle.extension_update_paths('your-TLE');
```

pgtle.install_extension

The install_extension function lets you install the artifacts that make up your TLE extension in the database, after which it can be created using the CREATE EXTENSION command.

Function prototype

```
pgtle.install_extension(name text, version text, description text, ext text, requires
 text[] DEFAULT NULL::text[])
```

Role

None.

Arguments

- name – The name of the TLE extension. This value is used when calling CREATE EXTENSION.
- version – The version of the TLE extension.
- description – A detailed description about the TLE extension. This description is displayed in the comment field in pgtle.available_extensions().
- ext – The contents of the TLE extension. This value contains objects such as functions.

- `requires` – An optional parameter that specifies dependencies for this TLE extension. The `pg_tle` extension is automatically added as a dependency.

Many of these arguments are the same as those that are included in an extension control file for installing a PostgreSQL extension on the file system of a PostgreSQL instance. For more information, see the Extension Files in Packaging Related Objects into an Extension in the PostgreSQL documentation.

Output

This functions returns `OK` on success and `NULL` on error.

- `OK` – The TLE extension has been successfully installed in the database.
- `NULL` – The TLE extension hasn't been successfully installed in the database.

Usage example

```
SELECT pgtle.install_extension(
 'pg_tle_test',
 '0.1',
 'My first pg_tle extension',
$_pgtle_$
  CREATE FUNCTION my_test()
  RETURNS INT
  AS $$
    SELECT 42;
  $$ LANGUAGE SQL IMMUTABLE;
$_pgtle_$
);
```

pgtle.install_update_path

The `install_update_path` function provides an update path between two different versions of a TLE extension. This function allows users of your TLE extension to update its version by using the `ALTER EXTENSION ... UPDATE` syntax.

Function prototype

```
pgtle.install_update_path(name text, fromvers text, tovers text, ext text)
```

Role

`pgtle_admin`

Arguments

- `name` – The name of the TLE extension. This value is used when calling `CREATE EXTENSION`.
- `fromvers` – The source version of the TLE extension for the upgrade.
- `tovers` – The destination version of the TLE extension for the upgrade.
- `ext` – The contents of the update. This value contains objects such as functions.

Output

None.

Usage example

```
SELECT pgtle.install_update_path('pg_tle_test', '0.1', '0.2',
  $_pgtle_$
    CREATE OR REPLACE FUNCTION my_test()
    RETURNS INT
    AS $$
      SELECT 21;
    $$ LANGUAGE SQL IMMUTABLE;
  $_pgtle_$
);
```

pgtle.register_feature

The `register_feature` function adds the specified internal PostgreSQL feature to the `pgtle.feature_info` table. PostgreSQL hooks are an example of an internal PostgreSQL feature. The Trusted Language Extensions development kit supports the use of PostgreSQL hooks. Currently, this function supports the following feature.

- `passcheck` – Registers the password-check hook with your procedure or function that customizes PostgreSQL's password-check behavior.

Function prototype

```
pgtle.register_feature(proc regproc, feature pg_tle_feature)
```

Role

pgtle_admin

Arguments

- proc – The name of a stored procedure or function to use for the feature.
- feature – The name of the pg_tle feature (such as passcheck) to register with the function.

Output

None.

Usage example

```
SELECT pgtle.register_feature('pw_hook', 'passcheck');
```

pgtle.register_feature_if_not_exists

The pgtle.register_feature_if_not_exists function adds the specified PostgreSQL feature to the pgtle.feature_info table and identifies the TLE extension or other procedure or function that uses the feature. For more information about hooks and Trusted Language Extensions, see Using PostgreSQL hooks with your TLE extensions.

Function prototype

```
pgtle.register_feature_if_not_exists(proc regproc, feature pg_tle_feature)
```

Role

pgtle_admin

Arguments

- proc – The name of a stored procedure or function that contains the logic (code) to use as a feature for your TLE extension. For example, the pw_hook code.

- `feature` – The name of the PostgreSQL feature to register for the TLE function. Currently, the only available feature is the `passcheck` hook. For more information, see Password-check hook (passcheck).

Output

Returns `true` after registering the feature for the specified extension. Returns `false` if the feature is already registered.

Usage example

```
SELECT pgtle.register_feature_if_not_exists('pw_hook', 'passcheck');
```

pgtle.set_default_version

The `set_default_version` function lets you specify a `default_version` for your TLE extension. You can use this function to define an upgrade path and designate the version as the default for your TLE extension. When database users specify your TLE extension in the `CREATE EXTENSION` and `ALTER EXTENSION ... UPDATE` commands, that version of your TLE extension is created in the database for that user.

This function returns `true` on success. If the TLE extension specified in the `name` argument doesn't exist, the function returns an error. Similarly, if the `version` of the TLE extension doesn't exist, it returns an error.

Function prototype

```
pgtle.set_default_version(name text, version text)
```

Role

`pgtle_admin`

Arguments

- `name` – The name of the TLE extension. This value is used when calling `CREATE EXTENSION`.
- `version` – The version of the TLE extension to set the default.

Output

- `true` – When setting default version succeeds, the function returns `true`.
- ERROR – Returns an error message if a TLE extension with the specified name or version doesn't exist.

Usage example

```
SELECT * FROM pgtle.set_default_version('my-extension', '1.1');
```

pgtle.uninstall_extension(name)

The `uninstall_extension` function removes all versions of a TLE extension from a database. This function prevents future calls of `CREATE EXTENSION` from installing the TLE extension. If the TLE extension doesn't exist in the database, an error is raised.

The `uninstall_extension` function won't drop a TLE extension that's currently active in the database. To remove a TLE extension that's currently active, you need to explicitly call `DROP EXTENSION` to remove it.

Function prototype

```
pgtle.uninstall_extension(extname text)
```

Role

`pgtle_admin`

Arguments

- `extname` – The name of the TLE extension to uninstall. This name is the same as the one used with `CREATE EXTENSION` to load the TLE extension for use in a given database.

Output

None.

Usage example

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test');
```

pgtle.uninstall_extension(name, version)

The `uninstall_extension(name, version)` function removes the specified version of the TLE extension from the database. This function prevents `CREATE EXTENSION` and `ALTER EXTENSION` from installing or updating a TLE extension to the specified version. This function also removes all update paths for the specified version of the TLE extension. This function won't uninstall the TLE extension if it's currently active in the database. You must explicitly call `DROP EXTENSION` to remove the TLE extension. To uninstall all versions of a TLE extension, see pgtle.uninstall_extension(name).

Function prototype

```
pgtle.uninstall_extension(extname text, version text)
```

Role

`pgtle_admin`

Arguments

- `extname` – The name of the TLE extension. This value is used when calling `CREATE EXTENSION`.
- `version` – The version of the TLE extension to uninstall from the database.

Output

None.

Usage example

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test', '0.2');
```

pgtle.uninstall_extension_if_exists

The `uninstall_extension_if_exists` function removes all versions of a TLE extension from a given database. If the TLE extension doesn't exist, the function returns silently (no error message is raised). If the specified extension is currently active within a database, this function doesn't drop it. You must explicitly call `DROP EXTENSION` to remove the TLE extension before using this function to uninstall its artifacts.

Function prototype

```
pgtle.uninstall_extension_if_exists(extname text)
```

Role

pgtle_admin

Arguments

- extname – The name of the TLE extension. This value is used when calling CREATE EXTENSION.

Output

The uninstall_extension_if_exists function returns true after uninstalling the specified extension. If the specified extension doesn't exist, the function returns false.

- true – Returns true after uninstalling the TLE extension.
- false – Returns false when the TLE extension doesn't exist in the database.

Usage example

```
SELECT * FROM pgtle.uninstall_extension_if_exists('pg_tle_test');
```

pgtle.uninstall_update_path

The uninstall_update_path function removes the specific update path from a TLE extension. This prevents ALTER EXTENSION ... UPDATE TO from using this as an update path.

If the TLE extension is currently being used by one of the versions on this update path, it remains in the database.

If the update path specified doesn't exist, this function raises an error.

Function prototype

```
pgtle.uninstall_update_path(extname text, fromvers text, tovers text)
```

Role

pgtle_admin

Arguments

- `extname` – The name of the TLE extension. This value is used when calling `CREATE EXTENSION`.
- `fromvers` – The source version of the TLE extension used on the update path.
- `tovers` – The destination version of the TLE extension used on the update path.

Output

None.

Usage example

```
SELECT * FROM pgtle.uninstall_update_path('pg_tle_test', '0.1', '0.2');
```

pgtle.uninstall_update_path_if_exists

The `uninstall_update_path_if_exists` function is similar to `uninstall_update_path` in that it removes the specified update path from a TLE extension. However, if the update path doesn't exist, this function doesn't raise an error message. Instead, the function returns `false`.

Function prototype

```
pgtle.uninstall_update_path_if_exists(extname text, fromvers text, tovers text)
```

Role

`pgtle_admin`

Arguments

- `extname` – The name of the TLE extension. This value is used when calling `CREATE EXTENSION`.
- `fromvers` – The source version of the TLE extension used on the update path.
- `tovers` – The destination version of the TLE extension used on the update path.

Output

- `true` – The function has successfully updated the path for the TLE extension.

- `false` – The function wasn't able to update the path for the TLE extension.

Usage example

```
SELECT * FROM pgtle.uninstall_update_path_if_exists('pg_tle_test', '0.1', '0.2');
```

pgtle.unregister_feature

The `unregister_feature` function provides a way to remove functions that were registered to use `pg_tle` features, such as hooks. For information about registering a feature, see pgtle.register_feature.

Function prototype

```
pgtle.unregister_feature(proc regproc, feature pg_tle_features)
```

Role

`pgtle_admin`

Arguments

- `proc` – The name of a stored function to register with a `pg_tle` feature.
- `feature` – The name of the `pg_tle` feature to register with the function. For example, `passcheck` is a feature that can be registered for use by the trusted language extensions that you develop. For more information, see Password-check hook (passcheck).

Output

None.

Usage example

```
SELECT * FROM pgtle.unregister_feature('pw_hook', 'passcheck');
```

pgtle.unregister_feature_if_exists

The `unregister_feature` function provides a way to remove functions that were registered to use `pg_tle` features, such as hooks. For more information, see Using PostgreSQL hooks with your

TLE extensions. Returns `true` after successfully unregistering the feature. Returns `false` if the feature wasn't registered.

For information about registering `pg_tle` features for your TLE extensions, see pgtle.register_feature.

Function prototype

```
pgtle.unregister_feature_if_exists('proc regproc', 'feature pg_tle_features')
```

Role

`pgtle_admin`

Arguments

- `proc` – The name of the stored function that was registered to include a `pg_tle` feature.
- `feature` – The name of the `pg_tle` feature that was registered with the trusted language extension.

Output

Returns `true` or `false`, as follows.

- `true` – The function has successfully unregistered the feature from extension.
- `false` – The function wasn't able to unregister the feature from the TLE extension.

Usage example

```
SELECT * FROM pgtle.unregister_feature_if_exists('pw_hook', 'passcheck');
```

Hooks reference for Trusted Language Extensions for PostgreSQL

Trusted Language Extensions for PostgreSQL supports PostgreSQL hooks. A *hook* is an internal callback mechanism available to developers for extending PostgreSQL's core functionality. By using hooks, developers can implement their own functions or procedures for use during various database operations, thereby modifying PostgreSQL's behavior in some way. For example, you

can use a passcheck hook to customize how PostgreSQL handles the passwords supplied when creating or changing passwords for users (roles).

View the following documentation to learn about the passcheck hook available for your TLE extensions. To learn more about the available hooks including the client authentication hook, see Trusted Language Extensions hooks.

Password-check hook (passcheck)

The passcheck hook is used to customize PostgreSQL behavior during the password-checking process for the following SQL commands and psql metacommand.

- CREATE ROLE *username* ...PASSWORD – For more information, see CREATE ROLE in the PostgreSQL documentation.
- ALTER ROLE *username*...PASSWORD – For more information, see ALTER ROLE in the PostgreSQL documentation.
- \password *username* – This interactive psql metacommand securely changes the password for the specified user by hashing the password before transparently using the ALTER ROLE ... PASSWORD syntax. The metacommand is a secure wrapper for the ALTER ROLE ... PASSWORD command, thus the hook applies to the behavior of the psql metacommand.

For an example, see Password-check hook code listing.

Contents

- Function prototype
- Arguments
- Configuration
- Usage notes

Function prototype

```
passcheck_hook(username text, password text, password_type pgtle.password_types,
 valid_until timestamptz, valid_null boolean)
```

Arguments

A passcheck hook function takes the following arguments.

- `username` – The name (as text) of the role (username) that's setting a password.
- `password` – The plaintext or hashed password. The password entered should match the type specified in `password_type`.
- `password_type` – Specify the `pgtle.password_type` format of the password. This format can be one of the following options.
  - `PASSWORD_TYPE_PLAINTEXT` – A plaintext password.
  - `PASSWORD_TYPE_MD5` – A password that's been hashed using MD5 (message digest 5) algorithm.
  - `PASSWORD_TYPE_SCRAM_SHA_256` – A password that's been hashed using SCRAM-SHA-256 algorithm.
- `valid_until` – Specify the time when the password becomes invalid. This argument is optional. If you use this argument, specify the time as a `timestamptz` value.
- `valid_null` – If this Boolean is set to `true`, the `valid_until` option is set to `NULL`.

Configuration

The function `pgtle.enable_password_check` controls whether the passcheck hook is active. The passcheck hook has three possible settings.

- `off` – Turns off the `passcheck` password-check hook. This is the default value.
- on – Turns on the `passcode` password-check hook so that passwords are checked against the table.
- `require` – Requires a password check hook to be defined.

Usage notes

To turn the `passcheck` hook on or off, you need to modify the custom DB parameter group for your RDS for PostgreSQL DB instance.

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
    --region aws-region ^
    --db-parameter-group-name your-custom-parameter-group ^
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Code examples for Amazon RDS using AWS SDKs

The following code examples show how to use Amazon RDS with an AWS software development kit (SDK).

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Basic examples for Amazon RDS using AWS SDKs
  - Hello Amazon RDS
  - Learn the basics of Amazon RDS with an AWS SDK
  - Actions for Amazon RDS using AWS SDKs
    - Use CreateDBInstance with an AWS SDK or CLI
    - Use CreateDBParameterGroup with an AWS SDK or CLI
    - Use CreateDBSnapshot with an AWS SDK or CLI
    - Use DeleteDBInstance with an AWS SDK or CLI
    - Use DeleteDBParameterGroup with an AWS SDK or CLI
    - Use DescribeAccountAttributes with an AWS SDK or CLI
    - Use DescribeDBEngineVersions with an AWS SDK or CLI
    - Use DescribeDBInstances with an AWS SDK or CLI
    - Use DescribeDBParameterGroups with an AWS SDK or CLI
    - Use DescribeDBParameters with an AWS SDK or CLI
    - Use DescribeDBSnapshots with an AWS SDK or CLI
    - Use DescribeOrderableDBInstanceOptions with an AWS SDK or CLI

    - Use GenerateRDSAuthToken with an AWS SDK
    - Use ModifyDBInstance with an AWS SDK or CLI
    - Use ModifyDBParameterGroup with an AWS SDK or CLI
    - Use RebootDBInstance with an AWS SDK or CLI
- Scenarios for Amazon RDS using AWS SDKs
    - Create an Aurora Serverless work item tracker
- Serverless examples for Amazon RDS
    - Connecting to an Amazon RDS database in a Lambda function

Basic examples for Amazon RDS using AWS SDKs

The following code examples show how to use the basics of Amazon Relational Database Service with AWS SDKs.

Examples

- Hello Amazon RDS
- Learn the basics of Amazon RDS with an AWS SDK
- Actions for Amazon RDS using AWS SDKs
    - Use CreateDBInstance with an AWS SDK or CLI
    - Use CreateDBParameterGroup with an AWS SDK or CLI
    - Use CreateDBSnapshot with an AWS SDK or CLI
    - Use DeleteDBInstance with an AWS SDK or CLI
    - Use DeleteDBParameterGroup with an AWS SDK or CLI
    - Use DescribeAccountAttributes with an AWS SDK or CLI
    - Use DescribeDBEngineVersions with an AWS SDK or CLI
    - Use DescribeDBInstances with an AWS SDK or CLI
    - Use DescribeDBParameterGroups with an AWS SDK or CLI
    - Use DescribeDBParameters with an AWS SDK or CLI
    - Use DescribeDBSnapshots with an AWS SDK or CLI
    - Use DescribeOrderableDBInstanceOptions with an AWS SDK or CLI
    - Use GenerateRDSAuthToken with an AWS SDK

- Use ModifyDBInstance with an AWS SDK or CLI
- Use ModifyDBParameterGroup with an AWS SDK or CLI
- Use RebootDBInstance with an AWS SDK or CLI

Hello Amazon RDS

The following code examples show how to get started using Amazon RDS.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using System;
using System.Threading.Tasks;
using Amazon.RDS;
using Amazon.RDS.Model;

namespace RDSActions;

public static class HelloRds
{
    static async Task Main(string[] args)
    {
        var rdsClient = new AmazonRDSClient();

        Console.WriteLine($"Hello Amazon RDS! Following are some of your DB
 instances:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first twenty DB instances.
        var response = await rdsClient.DescribeDBInstancesAsync(
            new DescribeDBInstancesRequest()
            {
```

```
                MaxRecords = 20 // Must be between 20 and 100.
            });

        foreach (var instance in response.DBInstances)
        {
            Console.WriteLine($"\tDB name: {instance.DBName}");
            Console.WriteLine($"\tArn: {instance.DBInstanceArn}");
            Console.WriteLine($"\tIdentifier: {instance.DBInstanceIdentifier}");
            Console.WriteLine();
        }
    }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_rds")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
```

```
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_rds.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code for the hello_rds.cpp source file.

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBInstancesRequest.h>
#include <iostream>

/*
 *  A "Hello Rds" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client and
 *  describes the Amazon RDS instances.
```

```
 *
 *  main function
 *
 *  Usage: 'hello_rds'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::RDS::RDSClient rdsClient(clientConfig);
        Aws::String marker;
        std::vector<Aws::String> instanceDBIDs;

        do {
            Aws::RDS::Model::DescribeDBInstancesRequest request;

            if (!marker.empty()) {
                request.SetMarker(marker);
            }

            Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
                    rdsClient.DescribeDBInstances(request);

            if (outcome.IsSuccess()) {
                for (auto &instance: outcome.GetResult().GetDBInstances()) {
                    instanceDBIDs.push_back(instance.GetDBInstanceIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with RDS::DescribeDBInstances. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
```

```
        } while (!marker.empty());

        std::cout << instanceDBIDs.size() << " RDS instances found." <<
 std::endl;
        for (auto &instanceDBID: instanceDBIDs) {
            std::cout << "   Instance: " << instanceDBID << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- For API details, see DescribeDBInstances in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Relational Database
 Service (Amazon RDS)
// client and list up to 20 DB instances in your account.
// This example uses the default settings specified in your shared credentials
```

```
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 rdsClient := rds.NewFromConfig(sdkConfig)
 const maxInstances = 20
 fmt.Printf("Let's list up to %v DB instances.\n", maxInstances)
 output, err := rdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{MaxRecords: aws.Int32(maxInstances)})
 if err != nil {
  fmt.Printf("Couldn't list DB instances: %v\n", err)
  return
 }
 if len(output.DBInstances) == 0 {
  fmt.Println("No DB instances found.")
 } else {
  for _, instance := range output.DBInstances {
   fmt.Printf("DB instance %v has database %v.\n",
 *instance.DBInstanceIdentifier,
    *instance.DBName)
  }
 }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.model.DescribeDbInstancesResponse;
import software.amazon.awssdk.services.rds.model.DBInstance;
import software.amazon.awssdk.services.rds.model.RdsException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeDBInstances {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeInstances(rdsClient);
        rdsClient.close();
    }

    public static void describeInstances(RdsClient rdsClient) {
        try {
            DescribeDbInstancesResponse response =
 rdsClient.describeDBInstances();
            List<DBInstance> instanceList = response.dbInstances();
            for (DBInstance instance : instanceList) {
                System.out.println("Instance ARN is: " +
 instance.dbInstanceArn());
                System.out.println("The Engine is " + instance.engine());
                System.out.println("Connection endpoint is" +
 instance.endpoint().address());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
```

```
            System.exit(1);
        }
    }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
"""
Purpose

Shows how to use the AWS SDK for Python (Boto3) with the Amazon Relational
 Database Service
(Amazon RDS) to list the databases in your account.
"""

import boto3
from botocore.exceptions import ClientError

# Create an RDS client
rds_client = boto3.client("rds")

# Create a paginator for the describe_db_instances operation
paginator = rds_client.get_paginator("describe_db_instances")

try:
    # Use the paginator to get a list of DB instances
    response_iterator = paginator.paginate(
        PaginationConfig={
            "MaxItems": 123,
            "PageSize": 50,  # Adjust PageSize as needed
            "StartingToken": None,
```

```
        }
    )

    # Iterate through the pages of the response
    instances_found = False
    for page in response_iterator:
        if "DBInstances" in page and page["DBInstances"]:
            instances_found = True
            print("Your RDS instances are:")
            for db in page["DBInstances"]:
                print(db["DBInstanceIdentifier"])

    if not instances_found:
        print("No RDS instances found!")

except ClientError as e:
    print(f"Couldn't list RDS instances. Here's why: {e.response['Error']
['Message']}")
```

- For API details, see DescribeDBInstances in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-rds'
require 'logger'

# RDSManager is a class responsible for managing RDS operations
# such as listing all RDS DB instances in the current AWS account.
class RDSManager
  def initialize(client)
    @client = client
```

```
    @logger = Logger.new($stdout)
  end

  # Lists and prints all RDS DB instances in the current AWS account.
  def list_db_instances
    @logger.info('Listing RDS DB instances')

    paginator = @client.describe_db_instances
    instances = []

    paginator.each_page do |page|
      instances.concat(page.db_instances)
    end

    if instances.empty?
      @logger.info('No instances found.')
    else
      @logger.info("Found #{instances.count} instance(s):")
      instances.each do |instance|
        @logger.info(" * #{instance.db_instance_identifier}
 (#{instance.db_instance_status})")
      end
    end
  end
end

if $PROGRAM_NAME == __FILE__
  rds_client = Aws::RDS::Client.new(region: 'us-west-2')
  manager = RDSManager.new(rds_client)
  manager.list_db_instances
end
```

- For API details, see DescribeDBInstances in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Learn the basics of Amazon RDS with an AWS SDK

The following code examples show how to:

- Create a custom DB parameter group and set parameter values.
- Create a DB instance that's configured to use the parameter group. The DB instance also contains a database.
- Take a snapshot of the instance.
- Delete the instance and parameter group.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
/// <summary>
/// Scenario for RDS DB instance example.
/// </summary>
public class RDSInstanceScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Returns a list of the available DB engine families using the
 DescribeDBEngineVersionsAsync method.
    2.  Selects an engine family and creates a custom DB parameter group using
 the CreateDBParameterGroupAsync method.
    3.  Gets the parameter groups using the DescribeDBParameterGroupsAsync
 method.
    4.  Gets parameters in the group using the DescribeDBParameters method.
```

```
   5.  Parses and displays parameters in the group.
   6.  Modifies both the auto_increment_offset and auto_increment_increment
parameters
       using the ModifyDBParameterGroupAsync method.
   7.  Gets and displays the updated parameters using the DescribeDBParameters
method with a source of "user".
   8.  Gets a list of allowed engine versions using the
DescribeDBEngineVersionsAsync method.
   9.  Displays and selects from a list of micro instance classes available for
the selected engine and version.
   10. Creates an RDS DB instance that contains a MySql database and uses the
parameter group
       using the CreateDBInstanceAsync method.
   11. Waits for DB instance to be ready using the DescribeDBInstancesAsync
method.
   12. Prints out the connection endpoint string for the new DB instance.
   13. Creates a snapshot of the DB instance using the CreateDBSnapshotAsync
method.
   14. Waits for DB snapshot to be ready using the DescribeDBSnapshots method.
   15. Deletes the DB instance using the DeleteDBInstanceAsync method.
   16. Waits for DB instance to be deleted using the DescribeDbInstances method.
   17. Deletes the parameter group using the DeleteDBParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static RDSWrapper rdsWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon RDS service.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<RDSWrapper>()
           )
           .Build();
```

```
        logger = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        }).CreateLogger<RDSInstanceScenario>();

        rdsWrapper = host.Services.GetRequiredService<RDSWrapper>();

        Console.WriteLine(sepBar);
        Console.WriteLine(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) DB
instance scenario example.");
        Console.WriteLine(sepBar);

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamily();

            var parameterGroup = await
CreateDbParameterGroup(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroup(parameterGroup.DBParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParameters(parameterGroup.DBParameterGroupName,
parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBParameterGroupName);

            var engineVersionChoice = await
ChooseDbEngineVersion(parameterGroupFamily);

            var instanceChoice = await ChooseDbInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            var newInstance = await CreateRdsNewInstance(parameterGroup, engine,
engineVersionChoice.EngineVersion,
                instanceChoice.DBInstanceClass, newInstanceIdentifier);
            if (newInstance != null)
            {
```

```
                DisplayConnectionString(newInstance);

                await CreateSnapshot(newInstance);

                await DeleteRdsInstance(newInstance);
            }

            await DeleteParameterGroup(parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the RDS DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamily()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await rdsWrapper.DescribeDBEngineVersions(engine);

        Console.WriteLine("1. The following is a list of available DB parameter
group families:");
        int i = 1;
        var parameterGroupFamilies = engines.GroupBy(e =>
e.DBParameterGroupFamily).ToList();
        foreach (var parameterGroupFamily in parameterGroupFamilies)
        {
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
```

```
        {
            Console.WriteLine("Select an available DB parameter group family by
entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber -
1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the
new DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBParameterGroup> CreateDbParameterGroup(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
{dbParameterGroupFamily}:");

        var parameterGroup = await rdsWrapper.CreateDBParameterGroup(
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            dbParameterGroupFamily, "New example parameter group");

        var groupInfo =
            await rdsWrapper.DescribeDBParameterGroups(parameterGroup
                .DBParameterGroupName);

        Console.WriteLine(
            $"3. New DB parameter group: \n\t{groupInfo[0].Description}, \n\tARN
{groupInfo[0].DBParameterGroupArn}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
```

```
    /// <param name="parameterNames">Optional specific names of parameters to
describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>> DescribeParametersInGroup(string
parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await rdsWrapper.DescribeDBParameters(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParameters(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        foreach (var p in parameters)
        {
```

```
            if (p.IsModifiable && p.DataType == "integer")
            {
                int newValue = 0;
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    Int32.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        await rdsWrapper.ModifyDBParameterGroup(parameterGroupName, parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string
parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe user source parameters in the group.");

        var parameters =
            await rdsWrapper.DescribeDBParameters(parameterGroupName, "user");


        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));
```

```
        Console.WriteLine(sepBar);
    }


    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDbEngineVersion(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await rdsWrapper.DescribeDBEngineVersions(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group
family {dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. Engine: {version.Engine} Version
{version.EngineVersion}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by
entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }
```

```
    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption>
 ChooseDbInstanceClass(string engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await rdsWrapper.DescribeOrderableDBInstanceOptions(engine,
 engineVersion);

        Console.WriteLine($"8. Available micro DB instance classes for engine
 {engine} and version {engineVersion}:");
        int i = 1;

        // Filter to micro instances for this example.
        allowedInstances = allowedInstances
            .Where(i => i.DBInstanceClass.Contains("micro")).ToList();

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("9. Select an available DB instance class by
 entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
```

```
    }

    /// <summary>
    /// Create a new RDS DB instance.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB
 instance.</param>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateRdsNewInstance(DBParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string instanceClass, string
 instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"10. Create a new DB instance with identifier
 {instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await rdsWrapper.DescribeDBInstances();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
 "available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
 instanceIdentifier);
        }
        else
        {
            Console.WriteLine("Please enter an admin user name:");
            var username = Console.ReadLine();

            Console.WriteLine("Please enter an admin password:");
            var password = Console.ReadLine();
```

```
            newInstance = await rdsWrapper.CreateDBInstance(
                "ExampleInstance",
                instanceIdentifier,
                parameterGroup.DBParameterGroupName,
                engineName,
                engineVersion,
                instanceClass,
                20,
                username,
                password
            );

            // 11. Wait for the DB instance to be ready.

            Console.WriteLine("11. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                instances = await
rdsWrapper.DescribeDBInstances(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
                newInstance = instances.First();
                Thread.Sleep(30000);
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to use to get a connection
string.</param>
    public static void DisplayConnectionString(DBInstance instance)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("12. New DB instance connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {instance.Endpoint.Address} -P
{instance.Endpoint.Port} "
            + $"-u {instance.MasterUsername} -p [YOUR PASSWORD]\n");
```

```
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an RDS DB instance.
    /// </summary>
    /// <param name="instance">DB instance to use when creating a snapshot.</
param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBSnapshot> CreateSnapshot(DBInstance instance)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"13. Creating snapshot from DB instance
 {instance.DBInstanceIdentifier}.");
        var snapshot = await
 rdsWrapper.CreateDBSnapshot(instance.DBInstanceIdentifier, "ExampleSnapshot-" +
 DateTime.Now.Ticks);

        // Wait for the snapshot to be available
        bool isSnapshotReady = false;

        Console.WriteLine($"14. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            var snapshots = await
 rdsWrapper.DescribeDBSnapshots(instance.DBInstanceIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
            Thread.Sleep(30000);
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBSnapshotIdentifier} status is
 {snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Delete an RDS DB instance.
    /// </summary>
    /// <param name="instance">The DB instance to delete.</param>
```

```
    /// <returns>Async task.</returns>
    public static async Task DeleteRdsInstance(DBInstance newInstance)
    {
        Console.WriteLine(sepBar);
        // Delete the DB instance.
        Console.WriteLine($"15. Delete the DB instance
{newInstance.DBInstanceIdentifier}.");
        await rdsWrapper.DeleteDBInstance(newInstance.DBInstanceIdentifier);

        // Wait for the DB instance to delete.
        Console.WriteLine($"16. Waiting for the DB instance to delete...");
        bool isInstanceDeleted = false;

        while (!isInstanceDeleted)
        {
            var instance = await rdsWrapper.DescribeDBInstances();
            isInstanceDeleted = instance.All(i => i.DBInstanceIdentifier !=
newInstance.DBInstanceIdentifier);
            Thread.Sleep(30000);
        }

        Console.WriteLine("DB instance deleted.");
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Delete a DB parameter group.
    /// </summary>
    /// <param name="parameterGroup">The parameter group to delete.</param>
    /// <returns>Async task.</returns>
    public static async Task DeleteParameterGroup(DBParameterGroup
parameterGroup)
    {
        Console.WriteLine(sepBar);
        // Delete the parameter group.
        Console.WriteLine($"17. Delete the DB parameter group
{parameterGroup.DBParameterGroupName}.");
        await
rdsWrapper.DeleteDBParameterGroup(parameterGroup.DBParameterGroupName);

        Console.WriteLine(sepBar);
    }
```

Wrapper methods used by the scenario for DB instance actions.

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
 DB instance operations.
/// </summary>
public partial class RDSWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public RDSWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }


    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="dbParameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>List of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string
 engine,
        string dbParameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = dbParameterGroupFamily
            });
        return response.DBEngineVersions;
    }



    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
```

```
   /// <returns>List of OrderableDBInstanceOptions.</returns>
   public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptions(string engine, string engineVersion)
   {
       // Use a paginator to get a list of DB instance options.
       var results = new List<OrderableDBInstanceOption>();
       var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
           new DescribeOrderableDBInstanceOptionsRequest()
           {
               Engine = engine,
               EngineVersion = engineVersion,
           });
       // Get the entire list using the paginator.
       await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
       {
           results.Add(instanceOptions);
       }
       return results;
   }



   /// <summary>
   /// Returns a list of DB instances.
   /// </summary>
   /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
   /// <returns>List of DB instances.</returns>
   public async Task<List<DBInstance>> DescribeDBInstances(string
dbInstanceIdentifier = null)
   {
       var results = new List<DBInstance>();
       var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
           new DescribeDBInstancesRequest
           {
               DBInstanceIdentifier = dbInstanceIdentifier
           });
       // Get the entire list using the paginator.
       await foreach (var instances in instancesPaginator.DBInstances)
       {
           results.Add(instances);
       }
```

```
        return results;
    }



    /// <summary>
    /// Create an RDS DB instance with a particular set of properties. Use the
action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbName">Name for the DB instance.</param>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="parameterGroupName">DB parameter group to associate with the
instance.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <param name="allocatedStorage">The amount of storage in gibibytes (GiB)
to allocate to the DB instance.</param>
    /// <param name="adminName">Admin user name.</param>
    /// <param name="adminPassword">Admin user password.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceIdentifier,
        string parameterGroupName, string dbEngine, string dbEngineVersion,
        string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
    {
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBName = dbName,
                DBInstanceIdentifier = dbInstanceIdentifier,
                DBParameterGroupName = parameterGroupName,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass,
                AllocatedStorage = allocatedStorage,
                MasterUsername = adminName,
                MasterUserPassword = adminPassword
            });

        return response.DBInstance;
    }
```

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstance(string dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

Wrapper methods used by the scenario for DB parameter groups.

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
 parameter groups.
/// </summary>
public partial class RDSWrapper
{

    /// <summary>
    /// Get descriptions of DB parameter groups.
    /// </summary>
    /// <param name="name">Optional name of the DB parameter group to describe.</
param>
    /// <returns>The list of DB parameter group descriptions.</returns>
    public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string
 name = null)
    {
        var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
```

```
            new DescribeDBParameterGroupsRequest()
            {
                DBParameterGroupName = name
            });
        return response.DBParameterGroups;
    }



    /// <summary>
    /// Create a new DB parameter group. Use the action
 DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="family">Family of the DB parameter group.</param>
    /// <param name="description">Description of the DB parameter group.</param>
    /// <returns>The new DB parameter group.</returns>
    public async Task<DBParameterGroup> CreateDBParameterGroup(
        string name, string family, string description)
    {
        var response = await _amazonRDS.CreateDBParameterGroupAsync(
            new CreateDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                DBParameterGroupFamily = family,
                Description = description
            });
        return response.DBParameterGroup;
    }



    /// <summary>
    /// Update a DB parameter group. Use the action
 DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
    /// <returns>The updated DB parameter group name.</returns>
    public async Task<string> ModifyDBParameterGroup(
        string name, List<Parameter> parameters)
```

```
    {
        var response = await _amazonRDS.ModifyDBParameterGroupAsync(
            new ModifyDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                Parameters = parameters,
            });
        return response.DBParameterGroupName;
    }



    /// <summary>
    /// Delete a DB parameter group. The group cannot be a default DB parameter
group
    /// or be associated with any DB instances.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDBParameterGroup(string name)
    {
        var response = await _amazonRDS.DeleteDBParameterGroupAsync(
            new DeleteDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }



    /// <summary>
    /// Get a list of DB parameters from a specific parameter group.
    /// </summary>
    /// <param name="dbParameterGroupName">Name of a specific DB parameter
group.</param>
    /// <param name="source">Optional source for selecting parameters.</param>
    /// <returns>List of parameter values.</returns>
    public async Task<List<Parameter>> DescribeDBParameters(string
dbParameterGroupName, string source = null)
    {
        var results = new List<Parameter>();
        var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
            new DescribeDBParametersRequest()
```

```
            {
                DBParameterGroupName = dbParameterGroupName,
                Source = source
            });
        // Get the entire list using the paginator.
        await foreach (var parameters in paginateParameters.Parameters)
        {
            results.Add(parameters);
        }
        return results;
    }
```

Wrapper methods used by the scenario for DB snapshot actions.

```
/// <summary>
/// Wrapper methods to use Amazon Relational Database Service (Amazon RDS) with
 snapshots.
/// </summary>
public partial class RDSWrapper
{

    /// <summary>
    /// Create a snapshot of a DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
 string snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBSnapshotAsync(
            new CreateDBSnapshotRequest()
            {
                DBSnapshotIdentifier = snapshotIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier
            });

        return response.DBSnapshot;
    }
```

```
    /// <summary>
    /// Return a list of DB snapshots for a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdentifier)
    {
        var results = new List<DBSnapshot>();
        var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
            new DescribeDBSnapshotsRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });

        // Get the entire list using the paginator.
        await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
        {
            results.Add(snapshots);
        }
        return results;
    }
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - CreateDBInstance
  - CreateDBParameterGroup
  - CreateDBSnapshot
  - DeleteDBInstance
  - DeleteDBParameterGroup
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeDBParameterGroups
  - DescribeDBParameters
  - DescribeDBSnapshots
  - DescribeOrderableDBInstanceOptions

- ModifyDBParameterGroup

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Routine which creates an Amazon RDS instance and demonstrates several
 operations
//! on that instance.
/*!
 \sa gettingStartedWithDBInstances()
 \param clientConfiguration: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::gettingStartedWithDBInstances(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::RDS::RDSClient client(clientConfig);

    printAsterisksLine();
    std::cout << "Welcome to the Amazon Relational Database Service (Amazon RDS)"
              << std::endl;
    std::cout << "get started with DB instances demo." << std::endl;
    printAsterisksLine();

    std::cout << "Checking for an existing DB parameter group named '" <<
              PARAMETER_GROUP_NAME << "'." << std::endl;
    Aws::String dbParameterGroupFamily("Undefined");
    bool parameterGroupFound = true;
    {
        // 1. Check if the DB parameter group already exists.
        Aws::RDS::Model::DescribeDBParameterGroupsRequest request;
```

```
        request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBParameterGroupsOutcome outcome =
                client.DescribeDBParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB parameter group named '" <<
                      PARAMETER_GROUP_NAME << "' already exists." << std::endl;
            dbParameterGroupFamily = outcome.GetResult().GetDBParameterGroups()
[0].GetDBParameterGroupFamily();
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::RDS::RDSErrors::D_B_PARAMETER_GROUP_NOT_FOUND_FAULT) {
            std::cout << "DB parameter group named '" <<
                      PARAMETER_GROUP_NAME << "' does not exist." << std::endl;
            parameterGroupFound = false;
        }
        else {
            std::cerr << "Error with RDS::DescribeDBParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!parameterGroupFound) {
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 2. Get available engine versions for the specified engine.
        if (!getDBEngineVersions(DB_ENGINE, NO_PARAMETER_GROUP_FAMILY,
                                 engineVersions, client)) {
            return false;
        }

        std::cout << "Getting available database engine versions for " <<
 DB_ENGINE
                  << "."
                  << std::endl;
        std::vector<Aws::String> families;
        for (const Aws::RDS::Model::DBEngineVersion &version: engineVersions) {
            Aws::String family = version.GetDBParameterGroupFamily();
            if (std::find(families.begin(), families.end(), family) ==
                families.end()) {
                families.push_back(family);
```

```
                std::cout << "  " << families.size() << ": " << family <<
std::endl;
            }
        }

        int choice = askQuestionForIntRange("Which family do you want to use? ",
1,
                                            static_cast<int>(families.size()));
        dbParameterGroupFamily = families[choice - 1];
    }
    if (!parameterGroupFound) {
        // 3.  Create a DB parameter group.
        Aws::RDS::Model::CreateDBParameterGroupRequest request;
        request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example parameter group.");

        Aws::RDS::Model::CreateDBParameterGroupOutcome outcome =
                client.CreateDBParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    printAsterisksLine();
    std::cout << "Let's set some parameter values in your parameter group."
              << std::endl;

    Aws::String marker;
    Aws::Vector<Aws::RDS::Model::Parameter> autoIncrementParameters;
    // 4.  Get the parameters in the DB parameter group.
    if (!getDBParameters(PARAMETER_GROUP_NAME, AUTO_INCREMENT_PREFIX, NO_SOURCE,
                         autoIncrementParameters,
                         client)) {
        cleanUpResources(PARAMETER_GROUP_NAME, "", client);
        return false;
```

```
    }

    Aws::Vector<Aws::RDS::Model::Parameter> updateParameters;

    for (Aws::RDS::Model::Parameter &autoIncParameter: autoIncrementParameters) {
        if (autoIncParameter.GetIsModifiable() &&
            (autoIncParameter.GetDataType() == "integer")) {
            std::cout << "The " << autoIncParameter.GetParameterName()
                      << " is described as: " <<
                      autoIncParameter.GetDescription() << "." << std::endl;
            if (autoIncParameter.ParameterValueHasBeenSet()) {
                std::cout << "The current value is "
                          << autoIncParameter.GetParameterValue()
                          << "." << std::endl;
            }
            std::vector<int> splitValues = splitToInts(
                    autoIncParameter.GetAllowedValues(), '-');
            if (splitValues.size() == 2) {
                int newValue = askQuestionForIntRange(
                        Aws::String("Enter a new value in the range ") +
                        autoIncParameter.GetAllowedValues() + ": ",
                        splitValues[0], splitValues[1]);
                autoIncParameter.SetParameterValue(std::to_string(newValue));
                updateParameters.push_back(autoIncParameter);

            }
            else {
                std::cerr << "Error parsing " <<
autoIncParameter.GetAllowedValues()
                          << std::endl;
            }
        }
    }

    {
        // 5.  Modify the auto increment parameters in the group.
        Aws::RDS::Model::ModifyDBParameterGroupRequest request;
        request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBParameterGroupOutcome outcome =
                client.ModifyDBParameterGroup(request);

        if (outcome.IsSuccess()) {
```

```
            std::cout << "The DB parameter group was successfully modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::ModifyDBParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    std::cout
            << "You can get a list of parameters you've set by specifying a
source of 'user'."
            << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> userParameters;
    // 6.  Display the modified parameters in the group.
    if (!getDBParameters(PARAMETER_GROUP_NAME, NO_NAME_PREFIX, "user",
userParameters,
                         client)) {
        cleanUpResources(PARAMETER_GROUP_NAME, "", client);
        return false;
    }

    for (const auto &userParameter: userParameters) {
        std::cout << "  " << userParameter.GetParameterName() << ", " <<
                  userParameter.GetDescription() << ", parameter value - "
                  << userParameter.GetParameterValue() << std::endl;
    }

    printAsterisksLine();
    std::cout << "Checking for an existing DB instance." << std::endl;

    Aws::RDS::Model::DBInstance dbInstance;
    // 7.  Check if the DB instance already exists.
    if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
        cleanUpResources(PARAMETER_GROUP_NAME, "", client);
        return false;
    }

    if (dbInstance.DbInstancePortHasBeenSet()) {
        std::cout << "The DB instance already exists." << std::endl;
    }
    else {
```

```
        std::cout << "Let's create a DB instance." << std::endl;
        const Aws::String administratorName = askQuestion(
                "Enter an administrator username for the database: ");
        const Aws::String administratorPassword = askQuestion(
                "Enter a password for the administrator (at least 8 characters):
");
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 8.  Get a list of available engine versions.
        if (!getDBEngineVersions(DB_ENGINE, dbParameterGroupFamily,
engineVersions,
                                 client)) {
            cleanUpResources(PARAMETER_GROUP_NAME, "", client);
            return false;
        }

        std::cout << "The available engines for your parameter group are:" <<
std::endl;

        int index = 1;
        for (const Aws::RDS::Model::DBEngineVersion &engineVersion:
engineVersions) {
            std::cout << "  " << index << ": " <<
engineVersion.GetEngineVersion()
                      << std::endl;
            ++index;
        }
        int choice = askQuestionForIntRange("Which engine do you want to use? ",
1,

static_cast<int>(engineVersions.size()));
        const Aws::RDS::Model::DBEngineVersion engineVersion =
engineVersions[choice -
                                                                            1];

        Aws::String dbInstanceClass;
        // 9.  Get a list of micro instance classes.
        if (!chooseMicroDBInstanceClass(engineVersion.GetEngine(),
                                        engineVersion.GetEngineVersion(),
                                        dbInstanceClass,
                                        client)) {
            cleanUpResources(PARAMETER_GROUP_NAME, "", client);
            return false;
        }
```

```
       std::cout << "Creating a DB instance named '" << DB_INSTANCE_IDENTIFIER
                 << "' and database '" << DB_NAME << "'.\n"
                 << "The DB instance is configured to use your custom parameter
group '"
                 << PARAMETER_GROUP_NAME << "',\n"
                 << "selected engine version " <<
engineVersion.GetEngineVersion()
                 << ",\n"
                 << "selected DB instance class '" << dbInstanceClass << "',"
                 << " and " << DB_ALLOCATED_STORAGE << " GiB of " <<
DB_STORAGE_TYPE
                 << " storage.\nThis typically takes several minutes." <<
std::endl;

       Aws::RDS::Model::CreateDBInstanceRequest request;
       request.SetDBName(DB_NAME);
       request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
       request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);
       request.SetEngine(engineVersion.GetEngine());
       request.SetEngineVersion(engineVersion.GetEngineVersion());
       request.SetDBInstanceClass(dbInstanceClass);
       request.SetStorageType(DB_STORAGE_TYPE);
       request.SetAllocatedStorage(DB_ALLOCATED_STORAGE);
       request.SetMasterUsername(administratorName);
       request.SetMasterUserPassword(administratorPassword);

       Aws::RDS::Model::CreateDBInstanceOutcome outcome =
               client.CreateDBInstance(request);

       if (outcome.IsSuccess()) {
           std::cout << "The DB instance creation has started."
                     << std::endl;
       }
       else {
           std::cerr << "Error with RDS::CreateDBInstance. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
           cleanUpResources(PARAMETER_GROUP_NAME, "", client);
           return false;
       }
   }

   std::cout << "Waiting for the DB instance to become available." << std::endl;
```

```
   int counter = 0;
   // 11. Wait for the DB instance to become available.
   do {
       std::this_thread::sleep_for(std::chrono::seconds(1));
       ++counter;
       if (counter > 900) {
           std::cerr << "Wait for instance to become available timed out ofter "
                     << counter
                     << " seconds." << std::endl;
           cleanUpResources(PARAMETER_GROUP_NAME, DB_INSTANCE_IDENTIFIER,
client);
           return false;
       }

       dbInstance = Aws::RDS::Model::DBInstance();
       if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
           cleanUpResources(PARAMETER_GROUP_NAME, DB_INSTANCE_IDENTIFIER,
client);
           return false;
       }

       if ((counter % 20) == 0) {
           std::cout << "Current DB instance status is '"
                     << dbInstance.GetDBInstanceStatus()
                     << "' after " << counter << " seconds." << std::endl;
       }
   } while (dbInstance.GetDBInstanceStatus() != "available");

   if (dbInstance.GetDBInstanceStatus() == "available") {
       std::cout << "The DB instance has been created." << std::endl;
   }

   printAsterisksLine();

   // 12. Display the connection string that can be used to connect a 'mysql'
shell to the database.
   displayConnection(dbInstance);

   printAsterisksLine();

   if (askYesNoQuestion(
           "Do you want to create a snapshot of your DB instance (y/n)? ")) {
       Aws::String snapshotID(DB_INSTANCE_IDENTIFIER + "-" +
```

```
                                Aws::String(Aws::Utils::UUID::RandomUUID()));
        {
            std::cout << "Creating a snapshot named " << snapshotID << "." <<
std::endl;
            std::cout << "This typically takes a few minutes." << std::endl;

            // 13. Create a snapshot of the DB instance.
            Aws::RDS::Model::CreateDBSnapshotRequest request;
            request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
            request.SetDBSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBSnapshotOutcome outcome =
                    client.CreateDBSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with RDS::CreateDBSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(PARAMETER_GROUP_NAME, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
        }

        std::cout << "Waiting for snapshot to become available." << std::endl;

        Aws::RDS::Model::DBSnapshot snapshot;
        counter = 0;
        do {
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++counter;
            if (counter > 600) {
                std::cerr << "Wait for snapshot to be available timed out ofter "
                          << counter
                          << " seconds." << std::endl;
                cleanUpResources(PARAMETER_GROUP_NAME, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

```
            // 14. Wait for the snapshot to become available.
            Aws::RDS::Model::DescribeDBSnapshotsRequest request;
            request.SetDBSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBSnapshotsOutcome outcome =
                    client.DescribeDBSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBSnapshots()[0];
            }
            else {
                std::cerr << "Error with RDS::DescribeDBSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(PARAMETER_GROUP_NAME, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }

            if ((counter % 20) == 0) {
                std::cout << "Current snapshot status is '"
                          << snapshot.GetStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (snapshot.GetStatus() != "available");

        if (snapshot.GetStatus() != "available") {
            std::cout << "A snapshot has been created." << std::endl;
        }
    }

    printAsterisksLine();

    bool result = true;
    if (askYesNoQuestion(
            "Do you want to delete the DB instance and parameter group (y/n)? "))
 {
        result = cleanUpResources(PARAMETER_GROUP_NAME, DB_INSTANCE_IDENTIFIER,
 client);
    }

    return result;
}
```

```
//! Routine which gets DB parameters using the 'DescribeDBParameters' api.
/*!
 \sa getDBParameters()
 \param parameterGroupName: The name of the parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::getDBParameters(const Aws::String &parameterGroupName,
                                  const Aws::String &namePrefix,
                                  const Aws::String &source,
                                  Aws::Vector<Aws::RDS::Model::Parameter>
 &parametersResult,
                                  const Aws::RDS::RDSClient &client) {
    Aws::String marker;
    do {
        Aws::RDS::Model::DescribeDBParametersRequest request;
        request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBParametersOutcome outcome =
                client.DescribeDBParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }
```

```
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with RDS::DescribeDBParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::getDBEngineVersions(const Aws::String &engineName,
                                      const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                      const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // Used for pagination.

    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
```

```
        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            auto &engineVersions = outcome.GetResult().GetDBEngineVersions();
            engineVersionsResult.insert(engineVersionsResult.end(),
 engineVersions.begin(),
                                        engineVersions.end());
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with RDS::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }

    } while (!marker.empty());


    return true;
}


//! Routine which gets a DB instance description.
/*!
 \sa describeDBInstance()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                     Aws::RDS::Model::DBInstance &instanceResult,
                                     const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
```

```
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with RDS::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}


//! Routine which gets available 'micro' DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseMicroDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::chooseMicroDBInstanceClass(const Aws::String &engine,
                                             const Aws::String &engineVersion,
                                             Aws::String &dbInstanceClass,
                                             const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker;
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
```

```
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (instanceClass.find("micro") != std::string::npos) {
                    if (std::find(instanceClasses.begin(), instanceClasses.end(),
                                  instanceClass) ==
                        instanceClasses.end()) {
                        instanceClasses.push_back(instanceClass);
                    }
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with RDS::DescribeOrderableDBInstanceOptions. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available micro DB instance classes for your database
 engine are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which micro DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}

//! Routine which deletes resources created by the scenario.
/*!
```

```
\sa cleanUpResources()
\param parameterGroupName: A parameter group name, this may be empty.
\param dbInstanceIdentifier: A DB instance identifier, this may be empty.
\param client: 'RDSClient' instance.
\return bool: Successful completion.
*/
bool AwsDoc::RDS::cleanUpResources(const Aws::String &parameterGroupName,
                                   const Aws::String &dbInstanceIdentifier,
                                   const Aws::RDS::RDSClient &client) {
    bool result = true;
    if (!dbInstanceIdentifier.empty()) {
        {
            // 15. Delete the DB instance.
            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with RDS::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }

        std::cout
                << "Waiting for DB instance to delete before deleting the
 parameter group."
                << std::endl;
        std::cout << "This may take a while." << std::endl;

        int counter = 0;
        Aws::RDS::Model::DBInstance dbInstance;
        do {
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++counter;
```

```
            if (counter > 800) {
                std::cerr << "Wait for instance to delete timed out ofter " <<
 counter
                          << " seconds." << std::endl;
                return false;
            }

            dbInstance = Aws::RDS::Model::DBInstance();
            // 16. Wait for the DB instance to be deleted.
            if (!describeDBInstance(dbInstanceIdentifier, dbInstance, client)) {
                return false;
            }

            if (dbInstance.DBInstanceIdentifierHasBeenSet() && (counter % 20) ==
 0) {
                std::cout << "Current DB instance status is '"
                          << dbInstance.GetDBInstanceStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (dbInstance.DBInstanceIdentifierHasBeenSet());
    }

    if (!parameterGroupName.empty()) {
        // 17. Delete the parameter group.
        Aws::RDS::Model::DeleteDBParameterGroupRequest request;
        request.SetDBParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBParameterGroupOutcome outcome =
                client.DeleteDBParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::DeleteDBParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
  - CreateDBInstance
  - CreateDBParameterGroup
  - CreateDBSnapshot
  - DeleteDBInstance
  - DeleteDBParameterGroup
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeDBParameterGroups
  - DescribeDBParameters
  - DescribeDBSnapshots
  - DescribeOrderableDBInstanceOptions
  - ModifyDBParameterGroup

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
import (
 "context"
 "fmt"
 "log"
 "sort"
 "strconv"
 "strings"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/rds/actions"
 "github.com/google/uuid"
)

// GetStartedInstances is an interactive example that shows you how to use the
 AWS SDK for Go
// with Amazon Relation Database Service (Amazon RDS) to do the following:
//
//  1. Create a custom DB parameter group and set parameter values.
//  2. Create a DB instance that is configured to use the parameter group. The DB
 instance
//     also contains a database.
//  3. Take a snapshot of the DB instance.
//  4. Delete the DB instance and parameter group.
type GetStartedInstances struct {
 sdkConfig  aws.Config
 instances  actions.DbInstances
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedInstances constructs a GetStartedInstances instance from a
 configuration.
// It uses the specified config to get an Amazon RDS
// client and create wrappers for the actions used in the scenario.
func NewGetStartedInstances(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedInstances {
 rdsClient := rds.NewFromConfig(sdkConfig)
 return GetStartedInstances{
  sdkConfig:  sdkConfig,
  instances:  actions.DbInstances{RdsClient: rdsClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
```

```
func (scenario GetStartedInstances) Run(ctx context.Context, dbEngine string,
 parameterGroupName string,
 instanceName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Relational Database Service (Amazon RDS) DB
 Instance demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(ctx, dbEngine,
 parameterGroupName)
 scenario.SetUserParameters(ctx, parameterGroupName)
 instance := scenario.CreateInstance(ctx, instanceName, dbEngine, dbName,
 parameterGroup)
 scenario.DisplayConnection(instance)
 scenario.CreateSnapshot(ctx, instance)
 scenario.Cleanup(ctx, instance, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a
 specified
// database engine and create a DB parameter group that is compatible with a
// selected engine family.
func (scenario GetStartedInstances) CreateParameterGroup(ctx context.Context,
 dbEngine string,
 parameterGroupName string) *types.DBParameterGroup {

 log.Printf("Checking for an existing DB parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.instances.GetParameterGroup(ctx,
 parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
```

```
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.instances.GetEngineVersions(ctx, dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?
\n", families)
  log.Println("Creating a DB parameter group.")
  _, err = scenario.instances.CreateParameterGroup(
   ctx, parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.instances.GetParameterGroup(ctx,
 parameterGroupName)
  if err != nil {
   panic(err)
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup
}

// SetUserParameters shows how to get the parameters contained in a custom
 parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedInstances) SetUserParameters(ctx context.Context,
 parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
```

```
 dbParameters, err := scenario.instances.GetParameters(ctx, parameterGroupName,
 "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   *dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.instances.UpdateParameters(ctx, parameterGroupName, updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("To get a list of parameters that you set previously, specify a
 source of 'user'.")
 userParameters, err := scenario.instances.GetParameters(ctx, parameterGroupName,
 "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateInstance shows how to create a DB instance that contains a database of a
// specified type. The database is also configured to use a custom DB parameter
 group.
func (scenario GetStartedInstances) CreateInstance(ctx context.Context,
 instanceName string, dbEngine string,
```

```
dbName string, parameterGroup *types.DBParameterGroup) *types.DBInstance {

log.Println("Checking for an existing DB instance.")
instance, err := scenario.instances.GetInstance(ctx, instanceName)
if err != nil {
 panic(err)
}
if instance == nil {
 adminUsername := scenario.questioner.Ask(
  "Enter an administrator username for the database: ", demotools.NotEmpty{})
 adminPassword := scenario.questioner.AskPassword(
  "Enter a password for the administrator (at least 8 characters): ", 7)
 engineVersions, err := scenario.instances.GetEngineVersions(ctx, dbEngine,
  *parameterGroup.DBParameterGroupFamily)
 if err != nil {
  panic(err)
 }
 var engineChoices []string
 for _, engine := range engineVersions {
  engineChoices = append(engineChoices, *engine.EngineVersion)
 }
 engineIndex := scenario.questioner.AskChoice(
  "The available engines for your parameter group are:\n", engineChoices)
 engineSelection := engineVersions[engineIndex]
 instOpts, err := scenario.instances.GetOrderableInstances(ctx,
*engineSelection.Engine,
  *engineSelection.EngineVersion)
 if err != nil {
  panic(err)
 }
 optSet := map[string]struct{}{}
 for _, opt := range instOpts {
  if strings.Contains(*opt.DBInstanceClass, "micro") {
   optSet[*opt.DBInstanceClass] = struct{}{}
  }
 }
 var optChoices []string
 for opt := range optSet {
  optChoices = append(optChoices, opt)
 }
 sort.Strings(optChoices)
 optIndex := scenario.questioner.AskChoice(
  "The available micro DB instance classes for your database engine are:\n",
optChoices)
```

```
  storageType := "standard"
  allocatedStorage := int32(5)
  log.Printf("Creating a DB instance named %v and database %v.\n"+
   "The DB instance is configured to use your custom parameter group %v,\n"+
   "selected engine %v,\n"+
   "selected DB instance class %v,"+
   "and %v GiB of %v storage.\n"+
   "This typically takes several minutes.",
   instanceName, dbName, *parameterGroup.DBParameterGroupName,
 *engineSelection.EngineVersion,
   optChoices[optIndex], allocatedStorage, storageType)
  instance, err = scenario.instances.CreateInstance(
   ctx, instanceName, dbName, *engineSelection.Engine,
 *engineSelection.EngineVersion,
   *parameterGroup.DBParameterGroupName, optChoices[optIndex], storageType,
   allocatedStorage, adminUsername, adminPassword)
  if err != nil {
   panic(err)
  }
  for *instance.DBInstanceStatus != "available" {
   scenario.helper.Pause(30)
   instance, err = scenario.instances.GetInstance(ctx, instanceName)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Instance created and available.")
 }
 log.Println("Instance data:")
 log.Printf("\tDBInstanceIdentifier: %v\n", *instance.DBInstanceIdentifier)
 log.Printf("\tARN: %v\n", *instance.DBInstanceArn)
 log.Printf("\tStatus: %v\n", *instance.DBInstanceStatus)
 log.Printf("\tEngine: %v\n", *instance.Engine)
 log.Printf("\tEngine version: %v\n", *instance.EngineVersion)
 log.Println(strings.Repeat("-", 88))
 return instance
}

// DisplayConnection displays connection information about a DB instance and tips
// on how to connect to it.
func (scenario GetStartedInstances) DisplayConnection(instance *types.DBInstance)
 {
 log.Println(
  "You can now connect to your database by using your favorite MySQL client.\n" +
```

```
    "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n"
 +
    "that is running in the same VPC as your DB instance. Pass the endpoint,\n" +
    "port, and administrator username to 'mysql'. Then, enter your password\n" +
    "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *instance.Endpoint.Address, instance.Endpoint.Port, *instance.MasterUsername)
 log.Println("For more information, see the User Guide for RDS:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/
CHAP_GettingStarted.CreatingConnecting.MySQL.html#CHAP_GettingStarted.Connecting.MySQL")
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB instance snapshot and wait until it's
 available.
func (scenario GetStartedInstances) CreateSnapshot(ctx context.Context, instance
 *types.DBInstance) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB instance (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", *instance.DBInstanceIdentifier,
 scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.
\n", snapshotId)
  snapshot, err := scenario.instances.CreateSnapshot(ctx,
 *instance.DBInstanceIdentifier, snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.instances.GetSnapshot(ctx, snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBSnapshotIdentifier: %v\n", *snapshot.DBSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBInstanceIdentifier: %v\n", *snapshot.DBInstanceIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
```

```
 }
}

// Cleanup shows how to clean up a DB instance and DB parameter group.
// Before the DB parameter group can be deleted, all associated DB instances must
 first be deleted.
func (scenario GetStartedInstances) Cleanup(
 ctx context.Context, instance *types.DBInstance, parameterGroup
 *types.DBParameterGroup) {

 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance and parameter group (y/n)? ",
 "y") {
  log.Printf("Deleting database instance %v.\n", *instance.DBInstanceIdentifier)
  err := scenario.instances.DeleteInstance(ctx, *instance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance to delete. This typically takes several
 minutes.")
  for instance != nil {
   scenario.helper.Pause(30)
   instance, err = scenario.instances.GetInstance(ctx,
 *instance.DBInstanceIdentifier)
   if err != nil {
    panic(err)
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBParameterGroupName)
  err = scenario.instances.DeleteParameterGroup(ctx,
 *parameterGroup.DBParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}

// IScenarioHelper abstracts the function from a scenario so that it
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 UniqueId() string
```

```
}
type ScenarioHelper struct{}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// UniqueId returns a new UUID.
func (helper ScenarioHelper) UniqueId() string {
 return uuid.New().String()
}
```

Define functions that are called by the scenario to manage Amazon RDS actions.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}


// GetParameterGroup gets a DB parameter group by name.
func (instances *DbInstances) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBParameterGroup, error) {
 output, err := instances.RdsClient.DescribeDBParameterGroups(
  ctx, &rds.DescribeDBParameterGroupsInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
```

```
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBParameterGroups[0], err
 }
}



// CreateParameterGroup creates a DB parameter group that is based on the
 specified
// parameter group family.
func (instances *DbInstances) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBParameterGroup, error) {

 output, err := instances.RdsClient.CreateDBParameterGroup(ctx,
  &rds.CreateDBParameterGroupInput{
   DBParameterGroupName:   aws.String(parameterGroupName),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
   Description:            aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB parameter group.
func (instances *DbInstances) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := instances.RdsClient.DeleteDBParameterGroup(ctx,
  &rds.DeleteDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
```

```
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// GetParameters gets the parameters that are contained in a DB parameter group.
func (instances *DbInstances) GetParameters(ctx context.Context,
 parameterGroupName string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator := rds.NewDescribeDBParametersPaginator(instances.RdsClient,
  &rds.DescribeDBParametersInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Source:               aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get parameters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB parameter group.
func (instances *DbInstances) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := instances.RdsClient.ModifyDBParameterGroup(ctx,
  &rds.ModifyDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
```

```
   Parameters:           params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// CreateSnapshot creates a snapshot of a DB instance.
func (instances *DbInstances) CreateSnapshot(ctx context.Context, instanceName
 string, snapshotName string) (
 *types.DBSnapshot, error) {
 output, err := instances.RdsClient.CreateDBSnapshot(ctx,
 &rds.CreateDBSnapshotInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBSnapshot, nil
 }
}



// GetSnapshot gets a DB instance snapshot.
func (instances *DbInstances) GetSnapshot(ctx context.Context, snapshotName
 string) (*types.DBSnapshot, error) {
 output, err := instances.RdsClient.DescribeDBSnapshots(ctx,
  &rds.DescribeDBSnapshotsInput{
   DBSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBSnapshots[0], nil
 }
```

```
}



// CreateInstance creates a DB instance.
func (instances *DbInstances) CreateInstance(ctx context.Context, instanceName
 string, dbName string,
 dbEngine string, dbEngineVersion string, parameterGroupName string,
 dbInstanceClass string,
 storageType string, allocatedStorage int32, adminName string, adminPassword
 string) (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBName:               aws.String(dbName),
  DBParameterGroupName: aws.String(parameterGroupName),
  Engine:               aws.String(dbEngine),
  EngineVersion:        aws.String(dbEngineVersion),
  DBInstanceClass:      aws.String(dbInstanceClass),
  StorageType:          aws.String(storageType),
  AllocatedStorage:     aws.Int32(allocatedStorage),
  MasterUsername:       aws.String(adminName),
  MasterUserPassword:   aws.String(adminPassword),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}



// GetInstance gets data about a DB instance.
func (instances *DbInstances) GetInstance(ctx context.Context, instanceName
 string) (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
```

```
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (instances *DbInstances) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := instances.RdsClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (instances *DbInstances) GetEngineVersions(ctx context.Context, engine
 string, parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := instances.RdsClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
```

```
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (instances *DbInstances) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instanceOptions []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(instances.RdsClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instance options: %v\n", err)
   break
  } else {
   instanceOptions = append(instanceOptions,
 output.OrderableDBInstanceOptions...)
  }
 }
 return instanceOptions, err
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CreateDBInstance

- CreateDBParameterGroup
- CreateDBSnapshot
- DeleteDBInstance
- DeleteDBParameterGroup
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeDBParameterGroups
- DescribeDBParameters
- DescribeDBSnapshots
- DescribeOrderableDBInstanceOptions
- ModifyDBParameterGroup

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run multiple operations.

```
import com.google.gson.Gson;
import
 software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.model.CreateDbInstanceRequest;
import software.amazon.awssdk.services.rds.model.CreateDbInstanceResponse;
import software.amazon.awssdk.services.rds.model.CreateDbParameterGroupResponse;
import software.amazon.awssdk.services.rds.model.CreateDbSnapshotRequest;
import software.amazon.awssdk.services.rds.model.CreateDbSnapshotResponse;
import software.amazon.awssdk.services.rds.model.DBEngineVersion;
import software.amazon.awssdk.services.rds.model.DBInstance;
import software.amazon.awssdk.services.rds.model.DBParameterGroup;
```

```
import software.amazon.awssdk.services.rds.model.DBSnapshot;
import software.amazon.awssdk.services.rds.model.DeleteDbInstanceRequest;
import software.amazon.awssdk.services.rds.model.DeleteDbInstanceResponse;
import software.amazon.awssdk.services.rds.model.DescribeDbEngineVersionsRequest;
import
 software.amazon.awssdk.services.rds.model.DescribeDbEngineVersionsResponse;
import software.amazon.awssdk.services.rds.model.DescribeDbInstancesRequest;
import software.amazon.awssdk.services.rds.model.DescribeDbInstancesResponse;
import
 software.amazon.awssdk.services.rds.model.DescribeDbParameterGroupsResponse;
import software.amazon.awssdk.services.rds.model.DescribeDbParametersResponse;
import software.amazon.awssdk.services.rds.model.DescribeDbSnapshotsRequest;
import software.amazon.awssdk.services.rds.model.DescribeDbSnapshotsResponse;
import
 software.amazon.awssdk.services.rds.model.DescribeOrderableDbInstanceOptionsResponse;
import software.amazon.awssdk.services.rds.model.ModifyDbParameterGroupResponse;
import software.amazon.awssdk.services.rds.model.OrderableDBInstanceOption;
import software.amazon.awssdk.services.rds.model.Parameter;
import software.amazon.awssdk.services.rds.model.RdsException;
import software.amazon.awssdk.services.rds.model.CreateDbParameterGroupRequest;
import
 software.amazon.awssdk.services.rds.model.DescribeDbParameterGroupsRequest;
import software.amazon.awssdk.services.rds.model.DescribeDbParametersRequest;
import software.amazon.awssdk.services.rds.model.ModifyDbParameterGroupRequest;
import
 software.amazon.awssdk.services.rds.model.DescribeOrderableDbInstanceOptionsRequest;
import software.amazon.awssdk.services.rds.model.DeleteDbParameterGroupRequest;
import software.amazon.awssdk.services.secretsmanager.SecretsManagerClient;
import
 software.amazon.awssdk.services.secretsmanager.model.GetSecretValueRequest;
import
 software.amazon.awssdk.services.secretsmanager.model.GetSecretValueResponse;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
```

```
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
 * This Java example performs these tasks:
 *
 * 1. Returns a list of the available DB engines.
 * 2. Selects an engine family and create a custom DB parameter group.
 * 3. Gets the parameter groups.
 * 4. Gets parameters in the group.
 * 5. Modifies the auto_increment_offset parameter.
 * 6. Gets and displays the updated parameters.
 * 7. Gets a list of allowed engine versions.
 * 8. Gets a list of micro instance classes available for the selected engine.
 * 9. Creates an RDS database instance that contains a MySql database and uses
 * the parameter group.
 * 10. Waits for the DB instance to be ready and prints out the connection
 * endpoint value.
 * 11. Creates a snapshot of the DB instance.
 * 12. Waits for an RDS DB snapshot to be ready.
 * 13. Deletes the RDS DB instance.
 * 14. Deletes the parameter group.
 */
public class RDSScenario {
    public static long sleepTime = 20;
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = """

                Usage:
                    <dbGroupName> <dbParameterGroupFamily> <dbInstanceIdentifier>
 <dbName> <dbSnapshotIdentifier> <secretName>

                Where:
                    dbGroupName - The database group name.\s
                    dbParameterGroupFamily - The database parameter group name
 (for example, mysql8.0).
                    dbInstanceIdentifier - The database instance identifier\s
                    dbName - The database name.\s
```

```
                    dbSnapshotIdentifier - The snapshot identifier.\s
                    secretName - The name of the AWS Secrets Manager secret that
contains the database credentials"
                """;

        if (args.length != 6) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbGroupName = args[0];
        String dbParameterGroupFamily = args[1];
        String dbInstanceIdentifier = args[2];
        String dbName = args[3];
        String dbSnapshotIdentifier = args[4];
        String secretName = args[5];

        Gson gson = new Gson();
        User user = gson.fromJson(String.valueOf(getSecretValues(secretName)),
User.class);
        String masterUsername = user.getUsername();
        String masterUserPassword = user.getPassword();

        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();
        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon RDS example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Return a list of the available DB engines");
        describeDBEngines(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a custom parameter group");
        createDBParameterGroup(rdsClient, dbGroupName, dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get the parameter group");
        describeDbParameterGroups(rdsClient, dbGroupName);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the parameters in the group");
        describeDbParameters(rdsClient, dbGroupName, 0);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Modify the auto_increment_offset parameter");
        modifyDBParas(rdsClient, dbGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Display the updated value");
        describeDbParameters(rdsClient, dbGroupName, -1);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Get a list of allowed engine versions");
        getAllowedEngines(rdsClient, dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Get a list of micro instance classes available for
the selected engine");
        getMicroInstances(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
                "9. Create an RDS database instance that contains a MySql
database and uses the parameter group");
        String dbARN = createDatabaseInstance(rdsClient, dbGroupName,
dbInstanceIdentifier, dbName, masterUsername,
                masterUserPassword);
        System.out.println("The ARN of the new database is " + dbARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Wait for DB instance to be ready");
        waitForInstanceReady(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("11. Create a snapshot of the DB instance");
        createSnapshot(rdsClient, dbInstanceIdentifier, dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Wait for DB snapshot to be ready");
        waitForSnapshotReady(rdsClient, dbInstanceIdentifier,
dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Delete the DB instance");
        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Delete the parameter group");
        deleteParaGroup(rdsClient, dbGroupName, dbARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);

        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
            .region(region)
            .build();
    }

    public static String getSecretValues(String secretName) {
        SecretsManagerClient secretClient = getSecretClient();
        GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
                .secretId(secretName)
                .build();

        GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
        return valueResponse.secretString();
    }
```

```
   // Delete the parameter group after database has been deleted.
   // An exception is thrown if you attempt to delete the para group while
database
   // exists.
   public static void deleteParaGroup(RdsClient rdsClient, String dbGroupName,
String dbARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(dbARN) == 0) {
                       System.out.println(dbARN + " still exists");
                       didFind = true;
                   }
                   if ((index == listSize) && (!didFind)) {
                       // Went through the entire list and did not find the
database ARN.
                       isDataDel = true;
                   }
                   Thread.sleep(sleepTime * 1000);
                   index++;
               }
           }

           // Delete the para group.
           DeleteDbParameterGroupRequest parameterGroupRequest =
DeleteDbParameterGroupRequest.builder()
                   .dbParameterGroupName(dbGroupName)
                   .build();

           rdsClient.deleteDBParameterGroup(parameterGroupRequest);
```

```
            System.out.println(dbGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Delete the DB instance.
    public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
        try {
            DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.print("The status of the database is " +
response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Waits until the snapshot instance is available.
    public static void waitForSnapshotReady(RdsClient rdsClient, String
dbInstanceIdentifier,
            String dbSnapshotIdentifier) {
        try {
            boolean snapshotReady = false;
            String snapshotReadyStr;
            System.out.println("Waiting for the snapshot to become available.");

            DescribeDbSnapshotsRequest snapshotsRequest =
DescribeDbSnapshotsRequest.builder()
                    .dbSnapshotIdentifier(dbSnapshotIdentifier)
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();
```

```
            while (!snapshotReady) {
                DescribeDbSnapshotsResponse response =
rdsClient.describeDBSnapshots(snapshotsRequest);
                List<DBSnapshot> snapshotList = response.dbSnapshots();
                for (DBSnapshot snapshot : snapshotList) {
                    snapshotReadyStr = snapshot.status();
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }

            System.out.println("The Snapshot is available!");
        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Create an Amazon RDS snapshot.
    public static void createSnapshot(RdsClient rdsClient, String
dbInstanceIdentifier, String dbSnapshotIdentifier) {
        try {
            CreateDbSnapshotRequest snapshotRequest =
CreateDbSnapshotRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbSnapshotResponse response =
rdsClient.createDBSnapshot(snapshotRequest);
            System.out.println("The Snapshot id is " +
response.dbSnapshot().dbiResourceId());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

```
   // Waits until the database instance is available.
   public static void waitForInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbInstancesRequest instanceRequest =
DescribeDbInstancesRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            String endpoint = "";
            while (!instanceReady) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances(instanceRequest);
                List<DBInstance> instanceList = response.dbInstances();
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint().address();
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available! The connection
endpoint is " + endpoint);

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
   }

   // Create a database instance and return the ARN of the database.
   public static String createDatabaseInstance(RdsClient rdsClient,
            String dbGroupName,
            String dbInstanceIdentifier,
            String dbName,
            String userName,
            String userPassword) {
```

```
       try {
           CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .allocatedStorage(100)
                   .dbName(dbName)
                   .engine("mysql")
                   .dbInstanceClass("db.t3.medium") // Updated to a supported class
                   .engineVersion("8.0.32")         // Updated to a supported
version
                   .storageType("gp2")             // Changed to General Purpose SSD
(gp2)
                   .masterUsername(userName)
                   .masterUserPassword(userPassword)
                   .build();

           CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
           System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
           return response.dbInstance().dbInstanceArn();

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }

       return "";
   }

   // Get a list of micro instances.
   public static void getMicroInstances(RdsClient rdsClient) {
       try {
           DescribeOrderableDbInstanceOptionsRequest dbInstanceOptionsRequest =
DescribeOrderableDbInstanceOptionsRequest
                   .builder()
                   .engine("mysql")
                   .build();

           DescribeOrderableDbInstanceOptionsResponse response = rdsClient

.describeOrderableDBInstanceOptions(dbInstanceOptionsRequest);
```

```
            List<OrderableDBInstanceOption> orderableDBInstances =
response.orderableDBInstanceOptions();
            for (OrderableDBInstanceOption dbInstanceOption :
orderableDBInstances) {
                System.out.println("The engine version is " +
dbInstanceOption.engineVersion());
                System.out.println("The engine description is " +
dbInstanceOption.engine());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Get a list of allowed engine versions.
    public static void getAllowedEngines(RdsClient rdsClient, String
dbParameterGroupFamily) {
        try {
            DescribeDbEngineVersionsRequest versionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .engine("mysql")
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(versionsRequest);
            List<DBEngineVersion> dbEngines = response.dbEngineVersions();
            for (DBEngineVersion dbEngine : dbEngines) {
                System.out.println("The engine version is " +
dbEngine.engineVersion());
                System.out.println("The engine description is " +
dbEngine.dbEngineDescription());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Modify auto_increment_offset and auto_increment_increment parameters.
    public static void modifyDBParas(RdsClient rdsClient, String dbGroupName) {
```

```
        try {
            Parameter parameter1 = Parameter.builder()
                    .parameterName("auto_increment_offset")
                    .applyMethod("immediate")
                    .parameterValue("5")
                    .build();

            List<Parameter> paraList = new ArrayList<>();
            paraList.add(parameter1);
            ModifyDbParameterGroupRequest groupRequest =
ModifyDbParameterGroupRequest.builder()
                    .dbParameterGroupName(dbGroupName)
                    .parameters(paraList)
                    .build();

            ModifyDbParameterGroupResponse response =
rdsClient.modifyDBParameterGroup(groupRequest);
            System.out.println("The parameter group " +
response.dbParameterGroupName() + " was successfully modified");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Retrieve parameters in the group.
    public static void describeDbParameters(RdsClient rdsClient, String
dbGroupName, int flag) {
        try {
            DescribeDbParametersRequest dbParameterGroupsRequest;
            if (flag == 0) {
                dbParameterGroupsRequest = DescribeDbParametersRequest.builder()
                        .dbParameterGroupName(dbGroupName)
                        .build();
            } else {
                dbParameterGroupsRequest = DescribeDbParametersRequest.builder()
                        .dbParameterGroupName(dbGroupName)
                        .source("user")
                        .build();
            }

            DescribeDbParametersResponse response =
rdsClient.describeDBParameters(dbParameterGroupsRequest);
```

```
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbParameterGroups(RdsClient rdsClient, String
dbGroupName) {
        try {
            DescribeDbParameterGroupsRequest groupsRequest =
DescribeDbParameterGroupsRequest.builder()
                    .dbParameterGroupName(dbGroupName)
                    .maxRecords(20)
                    .build();

            DescribeDbParameterGroupsResponse response =
rdsClient.describeDBParameterGroups(groupsRequest);
            List<DBParameterGroup> groups = response.dbParameterGroups();
            for (DBParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbParameterGroupName());
```

```
                System.out.println("The group description is " +
group.description());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBParameterGroup(RdsClient rdsClient, String
dbGroupName, String dbParameterGroupFamily) {
        try {
            CreateDbParameterGroupRequest groupRequest =
CreateDbParameterGroupRequest.builder()
                    .dbParameterGroupName(dbGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbParameterGroupResponse response =
rdsClient.createDBParameterGroup(groupRequest);
            System.out.println("The group name is " +
response.dbParameterGroup().dbParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .defaultOnly(true)
                    .engine("mysql")
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();
```

```
            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateDBInstance
  - CreateDBParameterGroup
  - CreateDBSnapshot
  - DeleteDBInstance
  - DeleteDBParameterGroup
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeDBParameterGroups
  - DescribeDBParameters
  - DescribeDBSnapshots
  - DescribeOrderableDBInstanceOptions
  - ModifyDBParameterGroup

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/**
Before running this code example, set up your development environment, including
 your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This example requires an AWS Secrets Manager secret that contains the database
 credentials. If you do not create a
secret, this example will not work. For more details, see:

https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html

This example performs the following tasks:

1. Returns a list of the available DB engines by invoking the
 DescribeDbEngineVersions method.
2. Selects an engine family and create a custom DB parameter group by invoking
 the createDBParameterGroup method.
3. Gets the parameter groups by invoking the DescribeDbParameterGroups method.
4. Gets parameters in the group by invoking the DescribeDbParameters method.
5. Modifies both the auto_increment_offset and auto_increment_increment
 parameters by invoking the modifyDbParameterGroup method.
6. Gets and displays the updated parameters.
7. Gets a list of allowed engine versions by invoking the
 describeDbEngineVersions method.
8. Gets a list of micro instance classes available for the selected engine.
9. Creates an Amazon Relational Database Service (Amazon RDS) database instance
 that contains a MySQL database and uses the parameter group.
```

```
10. Waits for DB instance to be ready and prints out the connection endpoint
 value.
11. Creates a snapshot of the DB instance.
12. Waits for the DB snapshot to be ready.
13. Deletes the DB instance.
14. Deletes the parameter group.
 */

var sleepTime: Long = 20

suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <dbGroupName> <dbParameterGroupFamily> <dbInstanceIdentifier>
 <dbName> <dbSnapshotIdentifier><secretName>

        Where:
            dbGroupName - The database group name.
            dbParameterGroupFamily - The database parameter group name.
            dbInstanceIdentifier - The database instance identifier.
            dbName -  The database name.
            dbSnapshotIdentifier - The snapshot identifier.
            secretName - The name of the AWS Secrets Manager secret that contains
 the database credentials.
    """

    if (args.size != 6) {
        println(usage)
        exitProcess(1)
    }

    val dbGroupName = args[0]
    val dbParameterGroupFamily = args[1]
    val dbInstanceIdentifier = args[2]
    val dbName = args[3]
    val dbSnapshotIdentifier = args[4]
    val secretName = args[5]

    val gson = Gson()
    val user = gson.fromJson(getSecretValues(secretName).toString(),
 User::class.java)
    val username = user.username
    val userPassword = user.password
```

```
    println("1. Return a list of the available DB engines")
    describeDBEngines()

    println("2. Create a custom parameter group")
    createDBParameterGroup(dbGroupName, dbParameterGroupFamily)

    println("3. Get the parameter groups")
    describeDbParameterGroups(dbGroupName)

    println("4. Get the parameters in the group")
    describeDbParameters(dbGroupName, 0)

    println("5. Modify the auto_increment_offset parameter")
    modifyDBParas(dbGroupName)

    println("6. Display the updated value")
    describeDbParameters(dbGroupName, -1)

    println("7. Get a list of allowed engine versions")
    getAllowedEngines(dbParameterGroupFamily)

    println("8. Get a list of micro instance classes available for the selected
engine")
    getMicroInstances()

    println("9. Create an RDS database instance that contains a MySql database
and uses the parameter group")
    val dbARN = createDatabaseInstance(dbGroupName, dbInstanceIdentifier, dbName,
username, userPassword)
    println("The ARN of the new database is $dbARN")

    println("10. Wait for DB instance to be ready")
    waitForDbInstanceReady(dbInstanceIdentifier)

    println("11. Create a snapshot of the DB instance")
    createDbSnapshot(dbInstanceIdentifier, dbSnapshotIdentifier)

    println("12. Wait for DB snapshot to be ready")
    waitForSnapshotReady(dbInstanceIdentifier, dbSnapshotIdentifier)

    println("13. Delete the DB instance")
    deleteDbInstance(dbInstanceIdentifier)

    println("14. Delete the parameter group")
```

```
    if (dbARN != null) {
        deleteParaGroup(dbGroupName, dbARN)
    }

    println("The Scenario has successfully completed.")
}

suspend fun deleteParaGroup(
    dbGroupName: String,
    dbARN: String,
) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false // Reset this value.
            didFind = false // Reset this value.
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(dbARN) == 0) {
                        println("$dbARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database name.
                        isDataDel = true
                    }
                    index++
                }
            }
        }

        // Delete the para group.
        val parameterGroupRequest =
            DeleteDbParameterGroupRequest {
```

```
                dbParameterGroupName = dbGroupName
            }
        rdsClient.deleteDbParameterGroup(parameterGroupRequest)
        println("$dbGroupName was deleted.")
    }
}

suspend fun deleteDbInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}

// Waits until the snapshot instance is available.
suspend fun waitForSnapshotReady(
    dbInstanceIdentifierVal: String?,
    dbSnapshotIdentifierVal: String?,
) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest =
        DescribeDbSnapshotsRequest {
            dbSnapshotIdentifier = dbSnapshotIdentifierVal
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    while (!snapshotReady) {
        RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
            val response = rdsClient.describeDbSnapshots(snapshotsRequest)
            val snapshotList: List<DbSnapshot>? = response.dbSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
```

```
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        print(".")
                        delay(sleepTime * 1000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}

// Create an Amazon RDS snapshot.
suspend fun createDbSnapshot(
    dbInstanceIdentifierVal: String?,
    dbSnapshotIdentifierVal: String?,
) {
    val snapshotRequest =
        CreateDbSnapshotRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            dbSnapshotIdentifier = dbSnapshotIdentifierVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbSnapshot(snapshotRequest)
        print("The Snapshot id is ${response.dbSnapshot?.dbiResourceId}")
    }
}

// Waits until the database instance is available.
suspend fun waitForDbInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }
    var endpoint = ""
    while (!instanceReady) {
        RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
            val response = rdsClient.describeDbInstances(instanceRequest)
```

```
            val instanceList = response.dbInstances
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus.toString()
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint?.address.toString()
                        instanceReady = true
                    } else {
                        print(".")
                        delay(sleepTime * 1000)
                    }
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}

// Create a database instance and return the ARN of the database.
suspend fun createDatabaseInstance(
    dbGroupNameVal: String?,
    dbInstanceIdentifierVal: String?,
    dbNameVal: String?,
    masterUsernameVal: String?,
    masterUserPasswordVal: String?,
): String? {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            allocatedStorage = 100
            dbName = dbNameVal
            dbParameterGroupName = dbGroupNameVal
            engine = "mysql"
            dbInstanceClass = "db.t3.micro"
            engineVersion = "8.0.35"
            storageType = "gp2"
            masterUsername = masterUsernameVal
            masterUserPassword = masterUserPasswordVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
```

```
        return response.dbInstance?.dbInstanceArn
    }
}

// Get a list of micro instances.
suspend fun getMicroInstances() {
    val dbInstanceOptionsRequest =
        DescribeOrderableDbInstanceOptionsRequest {
            engine = "mysql"
        }
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeOrderableDbInstanceOptions(dbInstanceOptionsRequest)
        val orderableDBInstances = response.orderableDbInstanceOptions
        if (orderableDBInstances != null) {
            for (dbInstanceOption in orderableDBInstances) {
                println("The engine version is
 ${dbInstanceOption.engineVersion}")
                println("The engine description is ${dbInstanceOption.engine}")
            }
        }
    }
}

// Get a list of allowed engine versions.
suspend fun getAllowedEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest =
        DescribeDbEngineVersionsRequest {
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            engine = "mysql"
        }
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        val dbEngines: List<DbEngineVersion>? = response.dbEngineVersions
        if (dbEngines != null) {
            for (dbEngine in dbEngines) {
                println("The engine version is ${dbEngine.engineVersion}")
                println("The engine description is
 ${dbEngine.dbEngineDescription}")
            }
        }
    }
}
```

```
// Modify the auto_increment_offset parameter.
suspend fun modifyDBParas(dbGroupName: String) {
    val parameter1 =
        Parameter {
            parameterName = "auto_increment_offset"
            applyMethod = ApplyMethod.Immediate
            parameterValue = "5"
        }

    val paraList: ArrayList<Parameter> = ArrayList()
    paraList.add(parameter1)
    val groupRequest =
        ModifyDbParameterGroupRequest {
            dbParameterGroupName = dbGroupName
            parameters = paraList
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbParameterGroup(groupRequest)
        println("The parameter group ${response.dbParameterGroupName} was
 successfully modified")
    }
}

// Retrieve parameters in the group.
suspend fun describeDbParameters(
    dbGroupName: String?,
    flag: Int,
) {
    val dbParameterGroupsRequest: DescribeDbParametersRequest
    dbParameterGroupsRequest =
        if (flag == 0) {
            DescribeDbParametersRequest {
                dbParameterGroupName = dbGroupName
            }
        } else {
            DescribeDbParametersRequest {
                dbParameterGroupName = dbGroupName
                source = "user"
            }
        }
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbParameters(dbParameterGroupsRequest)
        val dbParameters: List<Parameter>? = response.parameters
```

```
        var paraName: String
        if (dbParameters != null) {
            for (para in dbParameters) {
                // Only print out information about either auto_increment_offset
 or auto_increment_increment.
                paraName = para.parameterName.toString()
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    System.out.println("*** The parameter value is
 ${para.parameterValue}")
                    System.out.println("*** The parameter data type is
 ${para.dataType}")
                    System.out.println("*** The parameter description is
 ${para.description}")
                    System.out.println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}

suspend fun describeDbParameterGroups(dbGroupName: String?) {
    val groupsRequest =
        DescribeDbParameterGroupsRequest {
            dbParameterGroupName = dbGroupName
            maxRecords = 20
        }
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbParameterGroups(groupsRequest)
        val groups = response.dbParameterGroups
        if (groups != null) {
            for (group in groups) {
                println("The group name is ${group.dbParameterGroupName}")
                println("The group description is ${group.description}")
            }
        }
    }
}

// Create a parameter group.
suspend fun createDBParameterGroup(
    dbGroupName: String?,
```

```
    dbParameterGroupFamilyVal: String?,
) {
    val groupRequest =
        CreateDbParameterGroupRequest {
            dbParameterGroupName = dbGroupName
            dbParameterGroupFamily = dbParameterGroupFamilyVal
            description = "Created by using the AWS SDK for Kotlin"
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbParameterGroup(groupRequest)
        println("The group name is
 ${response.dbParameterGroup?.dbParameterGroupName}")
    }
}

// Returns a list of the available DB engines.
suspend fun describeDBEngines() {
    val engineVersionsRequest =
        DescribeDbEngineVersionsRequest {
            defaultOnly = true
            engine = "mysql"
            maxRecords = 20
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(engineVersionsRequest)
        val engines: List<DbEngineVersion>? = response.dbEngineVersions

        // Get all DbEngineVersion objects.
        if (engines != null) {
            for (engineOb in engines) {
                println("The name of the DB parameter group family for the
 database engine is ${engineOb.dbParameterGroupFamily}.")
                println("The name of the database engine ${engineOb.engine}.")
                println("The version number of the database engine
 ${engineOb.engineVersion}")
            }
        }
    }
}

suspend fun getSecretValues(secretName: String?): String? {
    val valueRequest =
```

```
        GetSecretValueRequest {
            secretId = secretName
        }

    SecretsManagerClient.fromEnvironment { region = "us-west-2" }.use
 { secretsClient ->
        val valueResponse = secretsClient.getSecretValue(valueRequest)
        return valueResponse.secretString
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
  - CreateDBInstance
  - CreateDBParameterGroup
  - CreateDBSnapshot
  - DeleteDBInstance
  - DeleteDBParameterGroup
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeDBParameterGroups
  - DescribeDBParameters
  - DescribeDBSnapshots
  - DescribeOrderableDBInstanceOptions
  - ModifyDBParameterGroup

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
class RdsInstanceScenario:
    """Runs a scenario that shows how to get started using Amazon RDS DB
 instances."""

    def __init__(self, instance_wrapper):
        """
        :param instance_wrapper: An object that wraps Amazon RDS DB instance
 actions.
        """
        self.instance_wrapper = instance_wrapper

    def create_parameter_group(self, parameter_group_name, db_engine):
        """
        Shows how to get available engine versions for a specified database
 engine and
        create a DB parameter group that is compatible with a selected engine
 family.

        :param parameter_group_name: The name given to the newly created
 parameter group.
        :param db_engine: The database engine to use as a basis.
        :return: The newly created parameter group.
        """
        print(
            f"Checking for an existing DB instance parameter group named
 {parameter_group_name}."
        )
        parameter_group = self.instance_wrapper.get_parameter_group(
            parameter_group_name
        )
        if parameter_group is None:
            print(f"Getting available database engine versions for {db_engine}.")
            engine_versions =
 self.instance_wrapper.get_engine_versions(db_engine)
            families = list({ver["DBParameterGroupFamily"] for ver in
 engine_versions})
            family_index = q.choose("Which family do you want to use? ",
 families)
            print(f"Creating a parameter group.")
            self.instance_wrapper.create_parameter_group(
                parameter_group_name, families[family_index], "Example parameter
 group."
            )
```

```
            parameter_group = self.instance_wrapper.get_parameter_group(
                parameter_group_name
            )
        print(f"Parameter group {parameter_group['DBParameterGroupName']}:")
        pp(parameter_group)
        print("-" * 88)
        return parameter_group

    def update_parameters(self, parameter_group_name):
        """
        Shows how to get the parameters contained in a custom parameter group and
        update some of the parameter values in the group.

        :param parameter_group_name: The name of the parameter group to query and
modify.
        """
        print("Let's set some parameter values in your parameter group.")
        auto_inc_parameters = self.instance_wrapper.get_parameters(
            parameter_group_name, name_prefix="auto_increment"
        )
        update_params = []
        for auto_inc in auto_inc_parameters:
            if auto_inc["IsModifiable"] and auto_inc["DataType"] == "integer":
                print(f"The {auto_inc['ParameterName']} parameter is described
as:")
                print(f"\t{auto_inc['Description']}")
                param_range = auto_inc["AllowedValues"].split("-")
                auto_inc["ParameterValue"] = str(
                    q.ask(
                        f"Enter a value between {param_range[0]} and
{param_range[1]}: ",
                        q.is_int,
                        q.in_range(int(param_range[0]), int(param_range[1])),
                    )
                )
                update_params.append(auto_inc)
        self.instance_wrapper.update_parameters(parameter_group_name,
update_params)
        print(
            "You can get a list of parameters you've set by specifying a source
of 'user'."
        )
        user_parameters = self.instance_wrapper.get_parameters(
            parameter_group_name, source="user"
```

```
        )
        pp(user_parameters)
        print("-" * 88)

    def create_instance(self, instance_name, db_name, db_engine,
parameter_group):
        """
        Shows how to create a DB instance that contains a database of a specified
        type and is configured to use a custom DB parameter group.

        :param instance_name: The name given to the newly created DB instance.
        :param db_name: The name given to the created database.
        :param db_engine: The engine of the created database.
        :param parameter_group: The parameter group that is associated with the
DB instance.
        :return: The newly created DB instance.
        """
        print("Checking for an existing DB instance.")
        db_inst = self.instance_wrapper.get_db_instance(instance_name)
        if db_inst is None:
            print("Let's create a DB instance.")
            admin_username = q.ask(
                "Enter an administrator user name for the database: ",
q.non_empty
            )
            admin_password = q.ask(
                "Enter a password for the administrator (at least 8 characters):
",
                q.non_empty,
            )
            engine_versions = self.instance_wrapper.get_engine_versions(
                db_engine, parameter_group["DBParameterGroupFamily"]
            )
            engine_choices = [ver["EngineVersion"] for ver in engine_versions]
            print("The available engines for your parameter group are:")
            engine_index = q.choose("Which engine do you want to use? ",
engine_choices)
            engine_selection = engine_versions[engine_index]
            print(
                "The available micro DB instance classes for your database engine
are:"
            )
            inst_opts = self.instance_wrapper.get_orderable_instances(
                engine_selection["Engine"], engine_selection["EngineVersion"]
```

```
            )
            inst_choices = list(
                {
                    opt["DBInstanceClass"]
                    for opt in inst_opts
                    if "micro" in opt["DBInstanceClass"]
                }
            )
            inst_index = q.choose(
                "Which micro DB instance class do you want to use? ",
inst_choices
            )
            group_name = parameter_group["DBParameterGroupName"]
            storage_type = "standard"
            allocated_storage = 5
            print(
                f"Creating a DB instance named {instance_name} and database
{db_name}.\n"
                f"The DB instance is configured to use your custom parameter
group {group_name},\n"
                f"selected engine {engine_selection['EngineVersion']},\n"
                f"selected DB instance class {inst_choices[inst_index]},"
                f"and {allocated_storage} GiB of {storage_type} storage.\n"
                f"This typically takes several minutes."
            )
            db_inst = self.instance_wrapper.create_db_instance(
                db_name,
                instance_name,
                group_name,
                engine_selection["Engine"],
                engine_selection["EngineVersion"],
                inst_choices[inst_index],
                storage_type,
                allocated_storage,
                admin_username,
                admin_password,
            )
            while db_inst.get("DBInstanceStatus") != "available":
                wait(10)
                db_inst = self.instance_wrapper.get_db_instance(instance_name)
        print("Instance data:")
        pp(db_inst)
        print("-" * 88)
        return db_inst
```

```
    @staticmethod
    def display_connection(db_inst):
        """
        Displays connection information about a DB instance and tips on how to
        connect to it.

        :param db_inst: The DB instance to display.
        """
        print(
            "You can now connect to your database using your favorite MySql
 client.\n"
            "One way to connect is by using the 'mysql' shell on an Amazon EC2
 instance\n"
            "that is running in the same VPC as your DB instance. Pass the
 endpoint,\n"
            "port, and administrator user name to 'mysql' and enter your password
\n"
            "when prompted:\n"
        )
        print(
            f"\n\tmysql -h {db_inst['Endpoint']['Address']} -P
 {db_inst['Endpoint']['Port']} "
            f"-u {db_inst['MasterUsername']} -p\n"
        )
        print(
            "For more information, see the User Guide for Amazon RDS:\n"
            "\thttps://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/
CHAP_GettingStarted.CreatingConnecting.MySQL.html#CHAP_GettingStarted.Connecting.MySQL"
        )
        print("-" * 88)

    def create_snapshot(self, instance_name):
        """
        Shows how to create a DB instance snapshot and wait until it's available.

        :param instance_name: The name of a DB instance to snapshot.
        """
        if q.ask(
            "Do you want to create a snapshot of your DB instance (y/n)? ",
 q.is_yesno
        ):
            snapshot_id = f"{instance_name}-{uuid.uuid4()}"
            print(
```

```
                f"Creating a snapshot named {snapshot_id}. This typically takes a
few minutes."
            )
            snapshot = self.instance_wrapper.create_snapshot(snapshot_id,
instance_name)
            while snapshot.get("Status") != "available":
                wait(10)
                snapshot = self.instance_wrapper.get_snapshot(snapshot_id)
            pp(snapshot)
            print("-" * 88)

    def cleanup(self, db_inst, parameter_group_name):
        """
        Shows how to clean up a DB instance and parameter group.
        Before the parameter group can be deleted, all associated DB instances
must first
        be deleted.

        :param db_inst: The DB instance to delete.
        :param parameter_group_name: The DB parameter group to delete.
        """
        if q.ask(
            "\nDo you want to delete the DB instance and parameter group (y/n)?
",
            q.is_yesno,
        ):
            print(f"Deleting DB instance {db_inst['DBInstanceIdentifier']}.")

self.instance_wrapper.delete_db_instance(db_inst["DBInstanceIdentifier"])
            print(
                "Waiting for the DB instance to delete. This typically takes
several minutes."
            )
            while db_inst is not None:
                wait(10)
                db_inst = self.instance_wrapper.get_db_instance(
                    db_inst["DBInstanceIdentifier"]
                )
            print(f"Deleting parameter group {parameter_group_name}.")
            self.instance_wrapper.delete_parameter_group(parameter_group_name)

    def run_scenario(self, db_engine, parameter_group_name, instance_name,
db_name):
```

```
        logging.basicConfig(level=logging.INFO, format="%(levelname)s:
 %(message)s")

        print("-" * 88)
        print(
            "Welcome to the Amazon Relational Database Service (Amazon RDS)\n"
            "get started with DB instances demo."
        )
        print("-" * 88)

        parameter_group = self.create_parameter_group(parameter_group_name,
 db_engine)
        self.update_parameters(parameter_group_name)
        db_inst = self.create_instance(
            instance_name, db_name, db_engine, parameter_group
        )
        self.display_connection(db_inst)
        self.create_snapshot(instance_name)
        self.cleanup(db_inst, parameter_group_name)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    try:
        scenario = RdsInstanceScenario(InstanceWrapper.from_client())
        scenario.run_scenario(
            "mysql",
            "doc-example-parameter-group",
            "doc-example-instance",
            "docexampledb",
        )
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

Define functions that are called by the scenario to manage Amazon RDS actions.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
```

```
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The parameter group.
        """
        try:
            response = self.rds_client.describe_db_parameter_groups(
                DBParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
```

```
    ):
        """
        Creates a DB parameter group that is based on the specified parameter
group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_parameter_group(
                DBParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            self.rds_client.delete_db_parameter_group(
                DBParameterGroupName=parameter_group_name
            )
        except ClientError as err:
```

```
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator = self.rds_client.get_paginator("describe_db_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
```

```
            raise
        else:
            return parameters


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_parameter_group(
                DBParameterGroupName=parameter_group_name,
Parameters=update_parameters
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def create_snapshot(self, snapshot_id, instance_id):
        """
        Creates a snapshot of a DB instance.

        :param snapshot_id: The ID to give the created snapshot.
        :param instance_id: The ID of the DB instance to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_snapshot(
                DBSnapshotIdentifier=snapshot_id,
DBInstanceIdentifier=instance_id
            )
            snapshot = response["DBSnapshot"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def get_snapshot(self, snapshot_id):
        """
        Gets a DB instance snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_snapshots(
                DBSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
```

```
                                        engine versions to those that are
compatible with
                                        this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst


    def create_db_instance(
        self,
        db_name,
        instance_id,
        parameter_group_name,
        db_engine,
```

```
        db_engine_version,
        instance_class,
        storage_type,
        allocated_storage,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB instance.

        :param db_name: The name of the database that is created in the DB
instance.
        :param instance_id: The ID to give the newly created DB instance.
        :param parameter_group_name: A parameter group to associate with the DB
instance.
        :param db_engine: The database engine of a database to create in the DB
instance.
        :param db_engine_version: The engine version for the created database.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :param storage_type: The storage type of the DB instance.
        :param allocated_storage: The amount of storage allocated on the DB
instance, in GiBs.
        :param admin_name: The name of the admin user for the created database.
        :param admin_password: The admin password for the created database.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBName=db_name,
                DBInstanceIdentifier=instance_id,
                DBParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                DBInstanceClass=instance_class,
                StorageType=storage_type,
                AllocatedStorage=allocated_storage,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
```

```
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateDBInstance
  - CreateDBParameterGroup
  - CreateDBSnapshot

- DeleteDBInstance
- DeleteDBParameterGroup
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeDBParameterGroups
- DescribeDBParameters
- DescribeDBSnapshots
- DescribeOrderableDBInstanceOptions
- ModifyDBParameterGroup

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The `Package.swift` file.

```
// swift-tools-version: 5.9
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "rds-scenario",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
```

```
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.4.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "rds-scenario",
            dependencies: [
                .product(name: "AWSRDS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

The Swift code file, `entry.swift`.

```
// An example that shows how to use the AWS SDK for Swift to perform a variety
// of operations using Amazon Relational Database Service (RDS).
//

import ArgumentParser
import Foundation
import AWSRDS

struct ExampleCommand: ParsableCommand {
    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion = "us-east-1"
    @Option(help: "The username to use for the database administrator.")
    var dbUsername = "admin"
    @Option(help: "The password to use for the database administrator.")
    var dbPassword: String
```

```
    static var configuration = CommandConfiguration(
        commandName: "rds-scenario",
        abstract: """
        Performs various operations to demonstrate the use of Amazon RDS
 Instances
        using the AWS SDK for Swift.
        """,
        discussion: """
        """
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let example = try await Example(region: awsRegion, username: dbUsername,
 password: dbPassword)

        await example.run()
    }
}

class Example {
    let rdsClient: RDSClient

    // Storage for AWS RDS properties

    let dbUsername: String
    let dbPassword: String
    var dbInstanceIdentifier: String
    var dbSnapshotIdentifier: String
    var dbParameterGroupName: String
    var dbParameterGroup: RDSClientTypes.DBParameterGroup?
    var selectedEngineVersion: String?

    init(region: String, username: String, password: String) async throws{
        let rdsConfig = try await RDSClient.RDSClientConfiguration(region:
 region)
        rdsClient = RDSClient(config: rdsConfig)

        dbUsername = username
        dbPassword = password
        dbParameterGroupName = ""
        dbInstanceIdentifier = ""
        dbSnapshotIdentifier = ""
```

```
    }

    /// The example's main body.
    func run() async {
        var parameterGroupFamilies: Set<String> = []

        //=========================================================================
        // 1. Get available database engine families for MySQL.
        //=========================================================================

        let engineVersions = await getDBEngineVersions(engineName: "mysql")

        for version in engineVersions {
            if version.dbParameterGroupFamily != nil {
                parameterGroupFamilies.insert(version.dbParameterGroupFamily!)
            }
        }

        if engineVersions.count > 0 {
            selectedEngineVersion = engineVersions.last!.engineVersion
        } else {
            print("*** Unable to find a valid database engine version. Canceling
operations.")
            await cleanUp()
            return
        }

        print("Found \(parameterGroupFamilies.count) parameter group families:")
        for family in parameterGroupFamilies {
            print("    \(family)")
        }

        //=========================================================================
        // 2. Select an engine family and create a custom DB parameter group.
        //    We select a family by sorting the set of family names, then
        //    choosing the last one.
        //=========================================================================

        let sortedFamilies = parameterGroupFamilies.sorted()

        guard let selectedFamily = sortedFamilies.last else {
            print("*** Unable to find a database engine family. Canceling
operations.")
            await cleanUp()
```

```
            return
        }

        print("Selected database engine family \(selectedFamily)")

        dbParameterGroupName = tempName(prefix: "rds-example")
        print("Creating a database parameter group named \(dbParameterGroupName)
using \(selectedFamily)")
        dbParameterGroup = await createDBParameterGroup(groupName:
dbParameterGroupName,
                                                        familyName:
selectedFamily)

        //===========================================================================
        // 3. Get the parameter group's details.
        //===========================================================================

        print("Getting the database parameter group list...")
        let dbParameterGroupList = await describeDBParameterGroups(groupName:
dbParameterGroupName)
        guard let dbParameterGroupList else {
            await cleanUp()
            return
        }

        print("Found \(dbParameterGroupList.count) parameter groups...")
        for group in dbParameterGroupList {
            print("    \(group.dbParameterGroupName ?? "<unknown>")")
        }
        print()

        //===========================================================================
        // 4. Get a list of the parameter group's parameters. This list is
        //    likely to be long, so use pagination. Find the
        //    auto_increment_offset and auto_increment_increment parameters.
        //===========================================================================

        let parameters = await describeDBParameters(groupName:
dbParameterGroupName)

        //===========================================================================
        // 5. Parse and display each parameter's name, description, and
        //    allowed values.
        //===========================================================================
```

```
        for parameter in parameters {
            let name = parameter.parameterName
            guard let name else {
                print("*** Unable to get parameter name!")
                continue
            }

            if name == "auto_increment_offset" || name ==
"auto_increment_increment" {
                print("Parameter \(name):")
                print("          Value: \(parameter.parameterValue ??
"<undefined>")")
                print("      Data type: \(parameter.dataType ?? "<unknown>")")
                print("    Description: \(parameter.description ?? "")")
                print(" Allowed values: \(parameter.allowedValues ??
"<unspecified")")
                print(String(repeating: "=", count: 78))
            }
        }

        //============================================================
        // 6. Modify both the auto_increment_offset and
        //    auto_increment_increment parameters in one call in the custom
        //    parameter group. Set their parameterValue fields to a new
        //    permitted value.
        //============================================================

        print("Setting auto_increment_offset and auto_increment_increment both to
5...")
        await modifyDBParameters(groupName: dbParameterGroupName)

        //============================================================
        // 7. Get and display the updated parameters, specifying a source of
        //    "user" to get only the modified parameters.
        //============================================================

        let updatedParameters = await describeDBParameters(groupName:
dbParameterGroupName, source: "user")

        for parameter in updatedParameters {
            let name = parameter.parameterName
            guard let name else {
                print("*** Unable to get parameter name!")
```

```
                continue
            }

            print("Parameter \(name):")
            print("          Value: \(parameter.parameterValue ??
"<undefined>")")
            print("      Data type: \(parameter.dataType ?? "<unknown>")")
            print("    Description: \(parameter.description ?? "")")
            print(" Allowed values: \(parameter.allowedValues ??
"<unspecified")")
            print(String(repeating: "=", count: 78))
        }

        //========================================================================
        // 8. Get a list of allowed engine versions using
        //    DescribeRDSEngineVersions.
        //========================================================================

        await listAllowedEngines(family: selectedFamily)

        //========================================================================
        // 9. Get a list of micro instance classes available for the selected
        //    engine and engine version.
        //========================================================================

        let dbInstanceClass = await chooseMicroInstance(engine: "mysql",
engineVersion: selectedEngineVersion)
        guard let dbInstanceClass else {
            print("Did not get a valid instance class. Canceling operations.")
            await cleanUp()
            return
        }

        //========================================================================
        // 10. Create an RDS database that contains a MySQL database and uses
        //     the parameter group we created.
        //========================================================================

        print("Creating the database instance...")

        guard let instanceClass = dbInstanceClass.dbInstanceClass else {
            print("Instance class name is unknown. Canceling operations.")
            await cleanUp()
            return
```

```
        }

        dbInstanceIdentifier = tempName(prefix: "sample-identifier")
        let dbInstanceArn = await createDBInstance(
            name: "SampleDatabase\(Int.random(in: 1000000..<1000000000))",
            instanceIdentifier: dbInstanceIdentifier,
            parameterGroupName: dbParameterGroupName,
            engine: "mysql",
            engineVersion: selectedEngineVersion!,
            instanceClass: instanceClass,
            username: dbUsername,
            password: dbPassword
        )

        if dbInstanceArn == nil {
            await cleanUp()
            return
        }

        //========================================================================
        // 11. Wait for the database instance to be ready by calling
        //     DescribeDBInstances repeatedly until it reports
        //     dbInstanceStatus as "available". This can take upwards of 10
        //     minutes, let the user know that.
        //========================================================================

        guard let endpoint = await waitUntilDBInstanceReady(instanceIdentifier:
dbInstanceIdentifier) else {
            print("\nDid not get a valid endpoint from AWS RDS.")
            await cleanUp()
            return
        }

        guard let endpointAddress = endpoint.address else {
            print("\nNo endpoint address returned.")
            await cleanUp()
            return
        }
        guard let endpointPort = endpoint.port else {
            print("\nNo endpoint port returned.")
            await cleanUp()
            return
        }
```

```
        //=========================================================================
        // 12. Display connection information for the database instance.
        //=========================================================================

        print("\nTo connect to the new database instance using 'mysql' from the
shell:")
        print("    mysql -h \(endpointAddress) -P \(endpointPort) -u
\(self.dbUsername)")

        //=========================================================================
        // 13. Create a snapshot of the database instance.
        //=========================================================================

        dbSnapshotIdentifier = tempName(prefix: "sample-snapshot")
        await createDBSnapshot(instanceIdentifier: dbInstanceIdentifier,
snapshotIdentifier: dbSnapshotIdentifier)

        //=========================================================================
        // 14. Wait for the snapshot to be ready.
        //=========================================================================

        await waitUntilDBSnapshotReady(instanceIdentifier: dbInstanceIdentifier,
snapshotIdentifier: dbSnapshotIdentifier)

        // That's it! Clean up and exit!

        print("Example complete! Cleaning up...")
        await cleanUp()
    }

    /// Clean up by discarding and closing down all allocated EC2 items.
    func cleanUp() async {
        print("Deleting the database instance \(dbInstanceIdentifier)...")
        await deleteDBInstance(instanceIdentifier: dbInstanceIdentifier)
        await waitUntilDBInstanceDeleted(instanceIdentifier:
dbInstanceIdentifier)

        print("Deleting the database parameter group \(dbParameterGroupName)...")
        await deleteDBParameterGroup(groupName: dbParameterGroupName)
    }

    /// Get all the database engine versions available for the specified
    /// database engine.
    ///
```

```
    /// - Parameter engineName: The name of the database engine to query.
    ///
    /// - Returns: An array of `RDSClientTypes.DBEngineVersion` structures,
    ///   each describing one supported version of the specified database.
    func getDBEngineVersions(engineName: String) async ->
[RDSClientTypes.DBEngineVersion] {
        do {
            let output = try await rdsClient.describeDBEngineVersions(
                input: DescribeDBEngineVersionsInput(
                    engine: engineName
                )
            )

            return output.dbEngineVersions ?? []
        } catch {
            return []
        }
    }

    /// Create a new database parameter group with the specified name.
    ///
    /// - Parameters:
    ///   - groupName: The name of the new parameter group.
    ///   - familyName: The name of the parameter group family.
    /// - Returns:
    func createDBParameterGroup(groupName: String, familyName: String) async ->
RDSClientTypes.DBParameterGroup? {
        do {
            let output = try await rdsClient.createDBParameterGroup(
                input: CreateDBParameterGroupInput(
                    dbParameterGroupFamily: familyName,
                    dbParameterGroupName: groupName,
                    description: "Created using the AWS SDK for Swift"
                )
            )
            return output.dbParameterGroup
        } catch {
            print("*** Error creating the parameter group:
\(error.localizedDescription)")
            return nil
        }
    }

    /// Get descriptions of the database parameter groups matching the given
```

```
    /// name.
    ///
    /// - Parameter groupName: The name of the parameter group to describe.
    ///
    /// - Returns: An array of [RDSClientTypes.DBParameterGroup] objects
    ///   describing the parameter group.
    func describeDBParameterGroups(groupName: String) async ->
[RDSClientTypes.DBParameterGroup]? {
        do {
            let output = try await rdsClient.describeDBParameterGroups(
                input: DescribeDBParameterGroupsInput(
                    dbParameterGroupName: groupName
                )
            )
            return output.dbParameterGroups
        } catch {
            print("*** Error getting the database parameter group's details:
\(error.localizedDescription)")
            return nil
        }
    }

    /// Returns the detailed parameter list for the specified database
    /// parameter group.
    ///
    /// - Parameters:
    ///   - groupName: The name of the parameter group to return parameters for.
    ///   - source: The types of parameters to return (`user`, `system`, or
    ///     `engine-default`).
    ///
    /// - Returns: An array of `RdSClientTypes.Parameter` objects, each
    ///   describing one of the group's parameters.
    func describeDBParameters(groupName: String, source: String? = nil) async ->
[RDSClientTypes.Parameter] {
        var parameterList: [RDSClientTypes.Parameter] = []

        do {
            let pages = rdsClient.describeDBParametersPaginated(
                input: DescribeDBParametersInput(
                    dbParameterGroupName: groupName,
                    source: source
                )
            )
```

```
            for try await page in pages {
                guard let parameters = page.parameters else {
                    return []
                }

                parameterList += parameters
            }
        } catch {
            print("*** Error getting database parameters:
\(error.localizedDescription)")
            return []
        }

        return parameterList
    }

    /// Demonstrates modifying two of the specified database parameter group's
    /// parameters.
    ///
    /// - Parameter groupName: The name of the parameter group to change
    ///   parameters for.
    func modifyDBParameters(groupName: String) async {
        let parameter1 = RDSClientTypes.Parameter(
            applyMethod: RDSClientTypes.ApplyMethod.immediate,
            parameterName: "auto_increment_offset",
            parameterValue: "5"
        )
        let parameter2 = RDSClientTypes.Parameter(
            applyMethod: RDSClientTypes.ApplyMethod.immediate,
            parameterName: "auto_increment_increment",
            parameterValue: "5"
        )

        let parameterList = [parameter1, parameter2]

        do {
            _ = try await rdsClient.modifyDBParameterGroup(
                input: ModifyDBParameterGroupInput(
                    dbParameterGroupName: groupName,
                    parameters: parameterList
                )
            )

            print("Successfully modified the parameter group \(groupName).")
```

```
        } catch {
            print("*** Error modifying the parameter group \(groupName):
\(error.localizedDescription)")
        }
    }

    /// Output a list of the database engine versions supported by the
    /// specified family.
    ///
    /// - Parameter family: The family for which to list allowed database
    ///   engines.
    func listAllowedEngines(family: String?) async {
        do {
            let output = try await rdsClient.describeDBEngineVersions(
                input: DescribeDBEngineVersionsInput(
                    dbParameterGroupFamily: family,
                    engine: "mysql"
                )
            )

            guard let engineVersions = output.dbEngineVersions else {
                print("No engine versions returned.")
                return
            }

            print("Found \(engineVersions.count) database engine versions:")
            for version in engineVersions {
                print("    \(version.engineVersion ?? "<unknown>"):
\(version.dbEngineDescription ?? "")")
            }
        } catch {
            print("*** Error getting database engine version list:
\(error.localizedDescription)")
            return
        }
    }

    /// Print a list of available database instances with "micro" in the class
    /// name, then return one of them to be used by other code.
    ///
    /// - Parameters:
    ///   - engine: The database engine for which to list database instance
    ///     classes.
    ///   - engineVersion: The database version for which to list instances.
```

```
    ///
    /// - Returns: An `RDSClientTypes.OrderableDBInstanceOption` describing
    ///   the selected instance type.
    func chooseMicroInstance(engine: String = "mysql", engineVersion: String? =
nil) async -> RDSClientTypes.OrderableDBInstanceOption? {
        do {
            let pages = rdsClient.describeOrderableDBInstanceOptionsPaginated(
                input: DescribeOrderableDBInstanceOptionsInput(
                    engine: engine,
                    engineVersion: engineVersion
                )
            )

            var optionsList: [RDSClientTypes.OrderableDBInstanceOption] = []

            for try await page in pages {
                guard let orderableDBInstanceOptions =
page.orderableDBInstanceOptions else {
                    continue
                }

                for dbInstanceOption in orderableDBInstanceOptions {
                    guard let className = dbInstanceOption.dbInstanceClass else {
                        continue
                    }
                    if className.contains("micro") {
                        optionsList.append(dbInstanceOption)
                    }
                }
            }

            print("Found \(optionsList.count) database instances of 'micro' class
types:")
            for dbInstanceOption in optionsList {
                print("    \(dbInstanceOption.engine ??
"<unknown>") \(dbInstanceOption.engineVersion ?? "<unknown>")
(\(dbInstanceOption.dbInstanceClass ?? "<unknown class>"))")
            }

            return optionsList[0]
        } catch {
            print("*** Error getting a list of orderable instance options:
\(error.localizedDescription)")
            return nil
```

```
        }
    }

    /// Create a new database instance.
    ///
    /// - Parameters:
    ///   - name: The name of the database to create.
    ///   - instanceIdentifier: The identifier to give the new database
    ///     instance.
    ///   - parameterGroupName: The name of the parameter group to associate
    ///     with the new database instance.
    ///   - engine: The database engine to use.
    ///   - engineVersion: The version of the database given by `engine` to
    ///     use.
    ///   - instanceClass: The memory and compute capacity of the database
    ///     instance, such as `db.m5.large``.
    ///   - username: The admin user's username to establish for the new
    ///     instance.
    ///   - password: The password to use for the specified user's access.
    ///
    /// - Returns: A string indicating the ARN of the newly created database
    ///   instance, or nil if the instance couldn't be created.
    func createDBInstance(name: String, instanceIdentifier: String,
parameterGroupName: String,
                          engine: String, engineVersion: String, instanceClass:
String,
                          username: String, password: String) async -> String? {
        do {
            let output = try await rdsClient.createDBInstance(
                input: CreateDBInstanceInput(
                    allocatedStorage: 100,
                    dbInstanceClass: instanceClass,
                    dbInstanceIdentifier: instanceIdentifier,
                    dbName: name,
                    dbParameterGroupName: parameterGroupName,
                    engine: engine,
                    engineVersion: engineVersion,
                    masterUserPassword: password,
                    masterUsername: username,
                    storageType: "gp2"
                )
            )

            guard let dbInstance = output.dbInstance else {
```

```
                print("*** Unable to get the database instance.")
                return nil
            }

            return dbInstance.dbInstanceArn
        } catch {
            print("*** An error occurred while creating the database instance:
\(error.localizedDescription)")
            return nil
        }
    }

    /// Wait until the specified database is available to use.
    ///
    /// - Parameter instanceIdentifier: The database instance identifier of the
    ///   database to wait for.
    func waitUntilDBInstanceReady(instanceIdentifier: String) async ->
RDSClientTypes.Endpoint? {
        do {
            putString("Waiting for the database instance to be ready to use. This
may take 10 minutes or more...")
            while true {
                let output = try await rdsClient.describeDBInstances(
                    input: DescribeDBInstancesInput(
                        dbInstanceIdentifier: instanceIdentifier
                    )
                )

                guard let instanceList = output.dbInstances else {
                    continue
                }

                for instance in instanceList {
                    let status = instance.dbInstanceStatus

                    guard let status else {
                        print("\nUnable to determine the status.")
                        continue
                    }

                    if status.contains("available") {
                        return instance.endpoint
                    } else {
                        putString(".")
```

```
                        do {
                            try await Task.sleep(for: .seconds(15))
                        } catch {
                            print("*** Error pausing the task!")
                        }
                    }
                }
            }
        } catch {
            print("*** Unable to wait until the database is ready:
\(error.localizedDescription)")
            return nil
        }
    }

    /// Create a snapshot of the specified name.
    ///
    /// - Parameters:
    ///   - instanceIdentifier: The identifier of the database instance to
    ///     snapshot.
    ///   - snapshotIdentifier: A unique identifier to give the newly-created
    ///     snapshot.
    func createDBSnapshot(instanceIdentifier: String, snapshotIdentifier: String)
async {
        do {
            let output = try await rdsClient.createDBSnapshot(
                input: CreateDBSnapshotInput(
                    dbInstanceIdentifier: instanceIdentifier,
                    dbSnapshotIdentifier: snapshotIdentifier
                )
            )

            guard let snapshot = output.dbSnapshot else {
                print("No snapshot returned.")
                return
            }

            print("The snapshot has been created with ID
\(snapshot.dbiResourceId ?? "<unknown>")")
        } catch {
            print("*** Unable to create the database snapshot named
\(snapshotIdentifier): \(error.localizedDescription)")
        }
    }
```

```
    /// Wait until the specified database snapshot is available to use.
    ///
    /// - Parameters:
    ///   - instanceIdentifier: The identifier of the database for which the
    ///     snapshot was taken.
    ///   - snapshotIdentifier: The identifier of the snapshot to wait for.
    func waitUntilDBSnapshotReady(instanceIdentifier: String, snapshotIdentifier:
String) async {
        var snapshotReady = false

        putString("Waiting for the snapshot to be ready...")

        do {
            while !snapshotReady {
                let output = try await rdsClient.describeDBSnapshots(
                    input: DescribeDBSnapshotsInput(
                        dbInstanceIdentifier: instanceIdentifier,
                        dbSnapshotIdentifier: snapshotIdentifier
                    )
                )

                guard let snapshotList = output.dbSnapshots else {
                    return
                }

                for snapshot in snapshotList {
                    guard let snapshotReadyStr = snapshot.status else {
                        return
                    }

                    if snapshotReadyStr.contains("available") {
                        snapshotReady = true
                        print()
                    } else {
                        putString(".")
                        do {
                            try await Task.sleep(for: .seconds(15))
                        } catch {
                            print("\n*** Error pausing the task!")
                        }
                    }
                }
            }
```

```
        } catch {
            print("\n*** Unable to wait for the database snapshot to be ready:
\(error.localizedDescription)")
        }
    }

    /// Delete the specified database instance.
    ///
    /// - Parameter instanceIdentifier: The identifier of the database
    ///   instance to delete.
    func deleteDBInstance(instanceIdentifier: String) async {
        do {
            _ = try await rdsClient.deleteDBInstance(
                input: DeleteDBInstanceInput(
                    dbInstanceIdentifier: instanceIdentifier,
                    deleteAutomatedBackups: true,
                    skipFinalSnapshot: true
                )
            )
        } catch {
            print("*** Error deleting the database instance
\(instanceIdentifier): \(error.localizedDescription)")
        }
    }

    /// Wait until the specified database instance has been deleted.
    ///
    /// - Parameter instanceIdentifier: The identifier of the database
    ///   instance to wait for.
    func waitUntilDBInstanceDeleted(instanceIdentifier: String) async {
        putString("Waiting for the database instance to be deleted. This may take
a few minutes...")
        do {
            var isDatabaseDeleted = false
            var foundInstance = false

            while !isDatabaseDeleted {
                let output = try await rdsClient.describeDBInstances(input:
DescribeDBInstancesInput())
                guard let instanceList = output.dbInstances else {
                    return
                }

                foundInstance = false
```

```
                for instance in instanceList {
                    guard let foundInstanceIdentifier =
instance.dbInstanceIdentifier else {
                        continue
                    }

                    if instanceIdentifier == foundInstanceIdentifier {
                        foundInstance = true
                        break
                    } else {
                        putString(".")
                        do {
                            try await Task.sleep(for: .seconds(15))
                        } catch {
                            print("\n*** Error pausing the task!")
                        }
                    }
                }
                if !foundInstance {
                    isDatabaseDeleted = true
                    print()
                }
            }
        } catch {
            print("\n*** Error waiting for the database instance to be deleted:
\(error.localizedDescription)")
        }
    }

    /// Delete the specified database parameter group.
    ///
    /// - Parameter groupName: The name of the parameter group to delete.
    func deleteDBParameterGroup(groupName: String) async {
        do {
            _ = try await rdsClient.deleteDBParameterGroup(
                input: DeleteDBParameterGroupInput(
                    dbParameterGroupName: groupName
                )
            )
        } catch {
            print("*** Error deleting the database parameter group \(groupName):
\(error.localizedDescription)")
        }
```

```
    }

    /// Generate and return a unique file name that begins with the specified
    /// string.
    ///
    /// - Parameters:
    ///   - prefix: Text to use at the beginning of the returned name.
    ///
    /// - Returns: A string containing a unique filename that begins with the
    ///   specified `prefix`.
    ///
    /// The returned name uses a random number between 1 million and 1 billion to
    /// provide reasonable certainty of uniqueness for the purposes of this
    /// example.
    func tempName(prefix: String) -> String {
        return "\(prefix)-\(Int.random(in: 1000000..<1000000000))"
    }

    /// Print a string to standard output without a trailing newline, and
    /// without buffering.
    ///
    /// - Parameter str: The string to output.
    func putString(_ str: String = "") {
        if str.length >= 1 {
            let data = str.data(using: .utf8)
            guard let data else {
                return
            }
            FileHandle.standardOutput.write(data)
        }
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
```

```
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Swift API reference*.
  - CreateDBInstance
  - CreateDBParameterGroup
  - CreateDBSnapshot
  - DeleteDBInstance
  - DeleteDBParameterGroup
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeDBParameterGroups
  - DescribeDBParameters
  - DescribeDBSnapshots
  - DescribeOrderableDBInstanceOptions
  - ModifyDBParameterGroup

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Actions for Amazon RDS using AWS SDKs

The following code examples demonstrate how to perform individual Amazon RDS actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon RDS API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Amazon RDS using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Relational Database Service API Reference.

Examples

- Use CreateDBInstance with an AWS SDK or CLI
- Use CreateDBParameterGroup with an AWS SDK or CLI
- Use CreateDBSnapshot with an AWS SDK or CLI
- Use DeleteDBInstance with an AWS SDK or CLI
- Use DeleteDBParameterGroup with an AWS SDK or CLI
- Use DescribeAccountAttributes with an AWS SDK or CLI
- Use DescribeDBEngineVersions with an AWS SDK or CLI
- Use DescribeDBInstances with an AWS SDK or CLI
- Use DescribeDBParameterGroups with an AWS SDK or CLI
- Use DescribeDBParameters with an AWS SDK or CLI
- Use DescribeDBSnapshots with an AWS SDK or CLI
- Use DescribeOrderableDBInstanceOptions with an AWS SDK or CLI
- Use GenerateRDSAuthToken with an AWS SDK
- Use ModifyDBInstance with an AWS SDK or CLI
- Use ModifyDBParameterGroup with an AWS SDK or CLI
- Use RebootDBInstance with an AWS SDK or CLI

Use `CreateDBInstance` with an AWS SDK or CLI

The following code examples show how to use `CreateDBInstance`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Create an RDS DB instance with a particular set of properties. Use the
action DescribeDBInstancesAsync
   /// to determine when the DB instance is ready to use.
   /// </summary>
   /// <param name="dbName">Name for the DB instance.</param>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <param name="parameterGroupName">DB parameter group to associate with the
instance.</param>
   /// <param name="dbEngine">The engine for the DB instance.</param>
   /// <param name="dbEngineVersion">Version for the DB instance.</param>
   /// <param name="instanceClass">Class for the DB instance.</param>
   /// <param name="allocatedStorage">The amount of storage in gibibytes (GiB)
to allocate to the DB instance.</param>
   /// <param name="adminName">Admin user name.</param>
   /// <param name="adminPassword">Admin user password.</param>
   /// <returns>DB instance object.</returns>
   public async Task<DBInstance> CreateDBInstance(string dbName, string
dbInstanceIdentifier,
       string parameterGroupName, string dbEngine, string dbEngineVersion,
       string instanceClass, int allocatedStorage, string adminName, string
adminPassword)
   {
       var response = await _amazonRDS.CreateDBInstanceAsync(
           new CreateDBInstanceRequest()
           {
               DBName = dbName,
               DBInstanceIdentifier = dbInstanceIdentifier,
               DBParameterGroupName = parameterGroupName,
               Engine = dbEngine,
               EngineVersion = dbEngineVersion,
               DBInstanceClass = instanceClass,
               AllocatedStorage = allocatedStorage,
               MasterUsername = adminName,
               MasterUserPassword = adminPassword
           });

       return response.DBInstance;
   }
```

- For API details, see CreateDBInstance in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::CreateDBInstanceRequest request;
    request.SetDBName(DB_NAME);
    request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
    request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);
    request.SetEngine(engineVersion.GetEngine());
    request.SetEngineVersion(engineVersion.GetEngineVersion());
    request.SetDBInstanceClass(dbInstanceClass);
    request.SetStorageType(DB_STORAGE_TYPE);
    request.SetAllocatedStorage(DB_ALLOCATED_STORAGE);
    request.SetMasterUsername(administratorName);
    request.SetMasterUserPassword(administratorPassword);

    Aws::RDS::Model::CreateDBInstanceOutcome outcome =
            client.CreateDBInstance(request);

    if (outcome.IsSuccess()) {
        std::cout << "The DB instance creation has started."
                  << std::endl;
    }
    else {
        std::cerr << "Error with RDS::CreateDBInstance. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        cleanUpResources(PARAMETER_GROUP_NAME, "", client);
```

```
            return false;
        }
```

- For API details, see CreateDBInstance in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To create a DB instance

The following `create-db-instance` example uses the required options to launch a new DB instance.

```
aws rds create-db-instance \
    --db-instance-identifier test-mysql-instance \
    --db-instance-class db.t3.micro \
    --engine mysql \
    --master-username admin \
    --master-user-password secret99 \
    --allocated-storage 20
```

Output:

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "test-mysql-instance",
        "DBInstanceClass": "db.t3.micro",
        "Engine": "mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "admin",
        "AllocatedStorage": 20,
        "PreferredBackupWindow": "12:55-13:25",
        "BackupRetentionPeriod": 1,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-12345abc",
                "Status": "active"
            }
        ],
```

```
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.mysql5.7",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-2ff2ff2f",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2c"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2d"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2a"
                    },
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "us-west-2b"
                    },
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sun:08:07-sun:08:37",
        "PendingModifiedValues": {
```

```
            "MasterUserPassword": "****"
        },
        "MultiAZ": false,
        "EngineVersion": "5.7.22",
        "AutoMinorVersionUpgrade": true,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "general-public-license",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "default:mysql-5-7",
                "Status": "in-sync"
            }
        ],
        "PubliclyAccessible": true,
        "StorageType": "gp2",
        "DbInstancePort": 0,
        "StorageEncrypted": false,
        "DbiResourceId": "db-5555EXAMPLE44444444EXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "DBInstanceArn": "arn:aws:rds:us-west-2:123456789012:db:test-mysql-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": []
    }
}
```

For more information, see Creating an Amazon RDS DB Instance in the *Amazon RDS User Guide*.

- For API details, see CreateDBInstance in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// CreateInstance creates a DB instance.
func (instances *DbInstances) CreateInstance(ctx context.Context, instanceName
 string, dbName string,
 dbEngine string, dbEngineVersion string, parameterGroupName string,
 dbInstanceClass string,
 storageType string, allocatedStorage int32, adminName string, adminPassword
 string) (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBName:               aws.String(dbName),
  DBParameterGroupName: aws.String(parameterGroupName),
  Engine:               aws.String(dbEngine),
  EngineVersion:        aws.String(dbEngineVersion),
```

```
  DBInstanceClass:      aws.String(dbInstanceClass),
  StorageType:          aws.String(storageType),
  AllocatedStorage:     aws.Int32(allocatedStorage),
  MasterUsername:       aws.String(adminName),
  MasterUserPassword:   aws.String(adminPassword),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- For API details, see CreateDBInstance in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import com.google.gson.Gson;
import
 software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.model.DescribeDbInstancesRequest;
import software.amazon.awssdk.services.rds.model.CreateDbInstanceRequest;
import software.amazon.awssdk.services.rds.model.CreateDbInstanceResponse;
import software.amazon.awssdk.services.rds.model.RdsException;
import software.amazon.awssdk.services.rds.model.DescribeDbInstancesResponse;
import software.amazon.awssdk.services.rds.model.DBInstance;
import software.amazon.awssdk.services.secretsmanager.SecretsManagerClient;
import
 software.amazon.awssdk.services.secretsmanager.model.GetSecretValueRequest;
```

```
import
 software.amazon.awssdk.services.secretsmanager.model.GetSecretValueResponse;

import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For more details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
 *
 */

public class CreateDBInstance {
    public static long sleepTime = 20;

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <dbInstanceIdentifier> <dbName> <secretName>

                Where:
                    dbInstanceIdentifier - The database instance identifier.\s
                    dbName - The database name.\s
                    secretName - The name of the AWS Secrets Manager secret that
 contains the database credentials."
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String dbInstanceIdentifier = args[0];
        String dbName = args[1];
        String secretName = args[2];
        Gson gson = new Gson();
        User user = gson.fromJson(String.valueOf(getSecretValues(secretName)),
User.class);
        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        createDatabaseInstance(rdsClient, dbInstanceIdentifier, dbName,
user.getUsername(), user.getPassword());
        waitForInstanceReady(rdsClient, dbInstanceIdentifier);
        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
                .region(region)

.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
                .build();
    }

    private static String getSecretValues(String secretName) {
        SecretsManagerClient secretClient = getSecretClient();
        GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
                .secretId(secretName)
                .build();

        GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
        return valueResponse.secretString();
    }

    public static void createDatabaseInstance(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbName,
            String userName,
            String userPassword) {
```

```
       try {
           CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .allocatedStorage(100)
                   .dbName(dbName)
                   .engine("mysql")
                   .dbInstanceClass("db.t3.medium") // Updated to a supported class
                   .engineVersion("8.0.32")         // Updated to a supported
version
                   .storageType("gp2")             // Changed to General Purpose SSD
(gp2)
                   .masterUsername(userName)
                   .masterUserPassword(userPassword)
                   .build();

           CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
           System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }

   // Waits until the database instance is available.
   public static void waitForInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
       boolean instanceReady = false;
       String instanceReadyStr;
       System.out.println("Waiting for instance to become available.");
       try {
           DescribeDbInstancesRequest instanceRequest =
DescribeDbInstancesRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .build();

           // Loop until the cluster is ready.
           while (!instanceReady) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances(instanceRequest);
               List<DBInstance> instanceList = response.dbInstances();
```

```
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available"))
                        instanceReady = true;
                    else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see CreateDBInstance in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createDatabaseInstance(
    dbInstanceIdentifierVal: String?,
    dbNamedbVal: String?,
    masterUsernameVal: String?,
    masterUserPasswordVal: String?,
) {
    val instanceRequest =
        CreateDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            allocatedStorage = 100
            dbName = dbNamedbVal
```

```
            engine = "mysql"
            dbInstanceClass = "db.t3.micro" // Use a supported instance class
            engineVersion = "8.0.39" // Use a supported engine version
            storageType = "gp2"
            masterUsername = masterUsernameVal
            masterUserPassword = masterUserPasswordVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
    }
}

// Waits until the database instance is available.
suspend fun waitForInstanceReady(dbInstanceIdentifierVal: String?) {
    val sleepTime: Long = 20
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest =
        DescribeDbInstancesRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            val instanceList = response.dbInstances
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus.toString()
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true
                    } else {
                        println("...$instanceReadyStr")
                        delay(sleepTime * 1000)
                    }
                }
            }
        }
        println("Database instance is available!")
    }
```

```
}
```

- For API details, see CreateDBInstance in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require __DIR__ . '/vendor/autoload.php';

use Aws\Exception\AwsException;



$rdsClient = new Aws\Rds\RdsClient([
    'region' => 'us-east-2'
]);

$dbIdentifier = '<<{{db-identifier}}>>';
$dbClass = 'db.t2.micro';
$storage = 5;
$engine = 'MySQL';
$username = 'MyUser';
$password = 'MyPassword';

try {
    $result = $rdsClient->createDBInstance([
        'DBInstanceIdentifier' => $dbIdentifier,
        'DBInstanceClass' => $dbClass,
        'AllocatedStorage' => $storage,
        'Engine' => $engine,
        'MasterUsername' => $username,
        'MasterUserPassword' => $password,
```

```
    ]);
    var_dump($result);
} catch (AwsException $e) {
    echo $e->getMessage();
    echo "\n";
}
```

- For API details, see CreateDBInstance in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_db_instance(
        self,
        db_name,
```

```
        instance_id,
        parameter_group_name,
        db_engine,
        db_engine_version,
        instance_class,
        storage_type,
        allocated_storage,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB instance.

        :param db_name: The name of the database that is created in the DB
instance.
        :param instance_id: The ID to give the newly created DB instance.
        :param parameter_group_name: A parameter group to associate with the DB
instance.
        :param db_engine: The database engine of a database to create in the DB
instance.
        :param db_engine_version: The engine version for the created database.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :param storage_type: The storage type of the DB instance.
        :param allocated_storage: The amount of storage allocated on the DB
instance, in GiBs.
        :param admin_name: The name of the admin user for the created database.
        :param admin_password: The admin password for the created database.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBName=db_name,
                DBInstanceIdentifier=instance_id,
                DBParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                DBInstanceClass=instance_class,
                StorageType=storage_type,
                AllocatedStorage=allocated_storage,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            db_inst = response["DBInstance"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- For API details, see CreateDBInstance in *AWS SDK for Python (Boto3) API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Create a new database instance.
    ///
    /// - Parameters:
    ///   - name: The name of the database to create.
    ///   - instanceIdentifier: The identifier to give the new database
    ///     instance.
    ///   - parameterGroupName: The name of the parameter group to associate
    ///     with the new database instance.
    ///   - engine: The database engine to use.
    ///   - engineVersion: The version of the database given by `engine` to
    ///     use.
    ///   - instanceClass: The memory and compute capacity of the database
    ///     instance, such as `db.m5.large``.
    ///   - username: The admin user's username to establish for the new
    ///     instance.
```

```
    ///   - password: The password to use for the specified user's access.
    ///
    /// - Returns: A string indicating the ARN of the newly created database
    ///   instance, or nil if the instance couldn't be created.
    func createDBInstance(name: String, instanceIdentifier: String,
parameterGroupName: String,
                          engine: String, engineVersion: String, instanceClass:
String,
                          username: String, password: String) async -> String? {
        do {
            let output = try await rdsClient.createDBInstance(
                input: CreateDBInstanceInput(
                    allocatedStorage: 100,
                    dbInstanceClass: instanceClass,
                    dbInstanceIdentifier: instanceIdentifier,
                    dbName: name,
                    dbParameterGroupName: parameterGroupName,
                    engine: engine,
                    engineVersion: engineVersion,
                    masterUserPassword: password,
                    masterUsername: username,
                    storageType: "gp2"
                )
            )

            guard let dbInstance = output.dbInstance else {
                print("*** Unable to get the database instance.")
                return nil
            }

            return dbInstance.dbInstanceArn
        } catch {
            print("*** An error occurred while creating the database instance:
\(error.localizedDescription)")
            return nil
        }
    }
```

- For API details, see CreateDBInstance in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateDBParameterGroup` with an AWS SDK or CLI

The following code examples show how to use `CreateDBParameterGroup`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Create a new DB parameter group. Use the action
DescribeDBParameterGroupsAsync
   /// to determine when the DB parameter group is ready to use.
   /// </summary>
   /// <param name="name">Name of the DB parameter group.</param>
   /// <param name="family">Family of the DB parameter group.</param>
   /// <param name="description">Description of the DB parameter group.</param>
   /// <returns>The new DB parameter group.</returns>
   public async Task<DBParameterGroup> CreateDBParameterGroup(
       string name, string family, string description)
   {
       var response = await _amazonRDS.CreateDBParameterGroupAsync(
           new CreateDBParameterGroupRequest()
           {
               DBParameterGroupName = name,
               DBParameterGroupFamily = family,
```

```
                Description = description
            });
        return response.DBParameterGroup;
    }
```

- For API details, see CreateDBParameterGroup in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::CreateDBParameterGroupRequest request;
    request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);
    request.SetDBParameterGroupFamily(dbParameterGroupFamily);
    request.SetDescription("Example parameter group.");

    Aws::RDS::Model::CreateDBParameterGroupOutcome outcome =
            client.CreateDBParameterGroup(request);

    if (outcome.IsSuccess()) {
        std::cout << "The DB parameter group was successfully created."
                  << std::endl;
    }
    else {
        std::cerr << "Error with RDS::CreateDBParameterGroup. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
```

```
        }
```

- For API details, see CreateDBParameterGroup in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To create a DB parameter group

The following `create-db-parameter-group` example creates a DB parameter group.

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family MySQL5.6 \
    --description "My new parameter group"
```

Output:

```
{
    "DBParameterGroup": {
        "DBParameterGroupName": "mydbparametergroup",
        "DBParameterGroupFamily": "mysql5.6",
        "Description": "My new parameter group",
        "DBParameterGroupArn": "arn:aws:rds:us-
east-1:123456789012:pg:mydbparametergroup"
    }
}
```

For more information, see Creating a DB Parameter Group in the *Amazon RDS User Guide*.

- For API details, see CreateDBParameterGroup in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// CreateParameterGroup creates a DB parameter group that is based on the
 specified
// parameter group family.
func (instances *DbInstances) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBParameterGroup, error) {

 output, err := instances.RdsClient.CreateDBParameterGroup(ctx,
  &rds.CreateDBParameterGroupInput{
   DBParameterGroupName:   aws.String(parameterGroupName),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
   Description:            aws.String(description),
  })
 if err != nil {
```

```
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBParameterGroup, err
 }
}
```

- For API details, see CreateDBParameterGroup in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void createDBParameterGroup(RdsClient rdsClient, String
dbGroupName, String dbParameterGroupFamily) {
       try {
           CreateDbParameterGroupRequest groupRequest =
CreateDbParameterGroupRequest.builder()
                   .dbParameterGroupName(dbGroupName)
                   .dbParameterGroupFamily(dbParameterGroupFamily)
                   .description("Created by using the AWS SDK for Java")
                   .build();

           CreateDbParameterGroupResponse response =
rdsClient.createDBParameterGroup(groupRequest);
           System.out.println("The group name is " +
response.dbParameterGroup().dbParameterGroupName());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- For API details, see CreateDBParameterGroup in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB parameter group that is based on the specified parameter
 group
        family.

        :param parameter_group_name: The name of the newly created parameter
 group.
```

```
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_parameter_group(
                DBParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- For API details, see CreateDBParameterGroup in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    " iv_dbparametergroupname   = 'mydbparametergroup'
    " iv_dbparametergroupfamily = 'mysql8.0'
    " iv_description            = 'My custom DB parameter group for MySQL 8.0'
```

```
    TRY.
        oo_result = lo_rds->createdbparametergroup(
          iv_dbparametergroupname   = iv_dbparametergroupname
          iv_dbparametergroupfamily = iv_dbparametergroupfamily
          iv_description            = iv_description ).
        MESSAGE 'DB parameter group created.' TYPE 'I'.
      CATCH /aws1/cx_rdsdbparmgralrexfault.
        MESSAGE 'DB parameter group already exists.' TYPE 'I'.
      CATCH /aws1/cx_rdsdbprmgrquotaexcd00.
        MESSAGE 'DB parameter group quota exceeded.' TYPE 'I'.
    ENDTRY.
```

- For API details, see CreateDBParameterGroup in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Create a new database parameter group with the specified name.
    ///
    /// - Parameters:
    ///   - groupName: The name of the new parameter group.
    ///   - familyName: The name of the parameter group family.
    /// - Returns:
    func createDBParameterGroup(groupName: String, familyName: String) async ->
 RDSClientTypes.DBParameterGroup? {
        do {
            let output = try await rdsClient.createDBParameterGroup(
                input: CreateDBParameterGroupInput(
                    dbParameterGroupFamily: familyName,
                    dbParameterGroupName: groupName,
                    description: "Created using the AWS SDK for Swift"
                )
```

```
            )
            return output.dbParameterGroup
        } catch {
            print("*** Error creating the parameter group:
\(error.localizedDescription)")
            return nil
        }
    }
```

- For API details, see CreateDBParameterGroup in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateDBSnapshot` with an AWS SDK or CLI

The following code examples show how to use `CreateDBSnapshot`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Create a snapshot of a DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
```

```
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBSnapshot> CreateDBSnapshot(string dbInstanceIdentifier,
string snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBSnapshotAsync(
            new CreateDBSnapshotRequest()
            {
                DBSnapshotIdentifier = snapshotIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier
            });

        return response.DBSnapshot;
    }
```

- For API details, see CreateDBSnapshot in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::CreateDBSnapshotRequest request;
            request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
            request.SetDBSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBSnapshotOutcome outcome =
                    client.CreateDBSnapshot(request);
```

```
            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with RDS::CreateDBSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(PARAMETER_GROUP_NAME, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }
```

- For API details, see CreateDBSnapshot in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To create a DB snapshot

The following `create-db-snapshot` example creates a DB snapshot.

```
aws rds create-db-snapshot \
    --db-instance-identifier database-mysql \
    --db-snapshot-identifier mydbsnapshot
```

Output:

```
{
    "DBSnapshot": {
        "DBSnapshotIdentifier": "mydbsnapshot",
        "DBInstanceIdentifier": "database-mysql",
        "Engine": "mysql",
        "AllocatedStorage": 100,
        "Status": "creating",
        "Port": 3306,
        "AvailabilityZone": "us-east-1b",
        "VpcId": "vpc-6594f31c",
        "InstanceCreateTime": "2019-04-30T15:45:53.663Z",
        "MasterUsername": "admin",
```

```
        "EngineVersion": "5.6.40",
        "LicenseModel": "general-public-license",
        "SnapshotType": "manual",
        "Iops": 1000,
        "OptionGroupName": "default:mysql-5-6",
        "PercentProgress": 0,
        "StorageType": "io1",
        "Encrypted": true,
        "KmsKeyId": "arn:aws:kms:us-east-1:123456789012:key/
AKIAIOSFODNN7EXAMPLE",
        "DBSnapshotArn": "arn:aws:rds:us-
east-1:123456789012:snapshot:mydbsnapshot",
        "IAMDatabaseAuthenticationEnabled": false,
        "ProcessorFeatures": [],
        "DbiResourceId": "db-AKIAIOSFODNN7EXAMPLE"
    }
}
```

For more information, see Creating a DB Snapshot in the *Amazon RDS User Guide*.

- For API details, see CreateDBSnapshot in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbInstances struct {
 RdsClient *rds.Client
}



// CreateSnapshot creates a snapshot of a DB instance.
func (instances *DbInstances) CreateSnapshot(ctx context.Context, instanceName
 string, snapshotName string) (
 *types.DBSnapshot, error) {
 output, err := instances.RdsClient.CreateDBSnapshot(ctx,
 &rds.CreateDBSnapshotInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBSnapshot, nil
 }
}
```

- For API details, see CreateDBSnapshot in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    // Create an Amazon RDS snapshot.
    public static void createSnapshot(RdsClient rdsClient, String
dbInstanceIdentifier, String dbSnapshotIdentifier) {
        try {
```

```
            CreateDbSnapshotRequest snapshotRequest =
 CreateDbSnapshotRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbSnapshotResponse response =
 rdsClient.createDBSnapshot(snapshotRequest);
            System.out.println("The Snapshot id is " +
 response.dbSnapshot().dbiResourceId());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- For API details, see CreateDBSnapshot in *AWS SDK for Java 2.x API Reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require __DIR__ . '/vendor/autoload.php';

use Aws\Exception\AwsException;



$rdsClient = new Aws\Rds\RdsClient([
    'region' => 'us-east-2'
]);

$dbIdentifier = '<<{{db-identifier}}>>';
```

```
$snapshotName = '<<{{backup_2018_12_25}}>>';

try {
    $result = $rdsClient->createDBSnapshot([
        'DBInstanceIdentifier' => $dbIdentifier,
        'DBSnapshotIdentifier' => $snapshotName,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    echo $e->getMessage();
    echo "\n";
}
```

- For API details, see CreateDBSnapshot in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
```

```
        return cls(rds_client)


    def create_snapshot(self, snapshot_id, instance_id):
        """
        Creates a snapshot of a DB instance.

        :param snapshot_id: The ID to give the created snapshot.
        :param instance_id: The ID of the DB instance to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_snapshot(
                DBSnapshotIdentifier=snapshot_id,
DBInstanceIdentifier=instance_id
            )
            snapshot = response["DBSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- For API details, see CreateDBSnapshot in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-rds' # v2: require 'aws-sdk'

# Create a snapshot for an Amazon Relational Database Service (Amazon RDS)
# DB instance.
#
# @param rds_resource [Aws::RDS::Resource] The resource containing SDK logic.
# @param db_instance_name [String] The name of the Amazon RDS DB instance.
# @return [Aws::RDS::DBSnapshot, nil] The snapshot created, or nil if error.
def create_snapshot(rds_resource, db_instance_name)
  id = "snapshot-#{rand(10**6)}"
  db_instance = rds_resource.db_instance(db_instance_name)
  db_instance.create_snapshot({
                                db_snapshot_identifier: id
                              })
rescue Aws::Errors::ServiceError => e
  puts "Couldn't create DB instance snapshot #{id}:\n #{e.message}"
end
```

- For API details, see CreateDBSnapshot in *AWS SDK for Ruby API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Create a snapshot of the specified name.
    ///
    /// - Parameters:
    ///   - instanceIdentifier: The identifier of the database instance to
    ///     snapshot.
    ///   - snapshotIdentifier: A unique identifier to give the newly-created
    ///     snapshot.
```

```
   func createDBSnapshot(instanceIdentifier: String, snapshotIdentifier: String)
async {
       do {
           let output = try await rdsClient.createDBSnapshot(
               input: CreateDBSnapshotInput(
                   dbInstanceIdentifier: instanceIdentifier,
                   dbSnapshotIdentifier: snapshotIdentifier
               )
           )

           guard let snapshot = output.dbSnapshot else {
               print("No snapshot returned.")
               return
           }

           print("The snapshot has been created with ID
\(snapshot.dbiResourceId ?? "<unknown>")")
       } catch {
           print("*** Unable to create the database snapshot named
\(snapshotIdentifier): \(error.localizedDescription)")
       }
   }
```

- For API details, see CreateDBSnapshot in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteDBInstance` with an AWS SDK or CLI

The following code examples show how to use `DeleteDBInstance`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Delete a particular DB instance.
/// </summary>
/// <param name="dbInstanceIdentifier">DB instance identifier.</param>
/// <returns>DB instance object.</returns>
public async Task<DBInstance> DeleteDBInstance(string dbInstanceIdentifier)
{
    var response = await _amazonRDS.DeleteDBInstanceAsync(
        new DeleteDBInstanceRequest()
        {
            DBInstanceIdentifier = dbInstanceIdentifier,
            SkipFinalSnapshot = true,
            DeleteAutomatedBackups = true
        });

    return response.DBInstance;
}
```

- For API details, see DeleteDBInstance in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DeleteDBInstanceRequest request;
        request.SetDBInstanceIdentifier(dbInstanceIdentifier);
        request.SetSkipFinalSnapshot(true);
        request.SetDeleteAutomatedBackups(true);

        Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                client.DeleteDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB instance deletion has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::DeleteDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- For API details, see DeleteDBInstance in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete a DB instance

The following `delete-db-instance` example deletes the specified DB instance after creating a final DB snapshot named `test-instance-final-snap`.

```
aws rds delete-db-instance \
    --db-instance-identifier test-instance \
    --final-db-snapshot-identifier test-instance-final-snap
```

Output:

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "test-instance",
        "DBInstanceStatus": "deleting",
        ...some output truncated...
    }
}
```

- For API details, see DeleteDBInstance in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (instances *DbInstances) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := instances.RdsClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see DeleteDBInstance in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.model.DeleteDbInstanceRequest;
import software.amazon.awssdk.services.rds.model.DeleteDbInstanceResponse;
import software.amazon.awssdk.services.rds.model.RdsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteDBInstance {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <dbInstanceIdentifier>\s

                Where:
                    dbInstanceIdentifier - The database instance identifier\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbInstanceIdentifier = args[0];
        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        rdsClient.close();
    }

    public static void deleteDatabaseInstance(RdsClient rdsClient, String
 dbInstanceIdentifier) {
        try {
```

```
            DeleteDbInstanceRequest deleteDbInstanceRequest =
 DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
 rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.print("The status of the database is " +
 response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DeleteDBInstance in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteDatabaseInstance(dbInstanceIdentifierVal: String?) {
    val deleteDbInstanceRequest =
        DeleteDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            deleteAutomatedBackups = true
            skipFinalSnapshot = true
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
```

```
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}
```

- For API details, see DeleteDBInstance in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require __DIR__ . '/vendor/autoload.php';

use Aws\Exception\AwsException;


//Create an RDSClient
$rdsClient = new Aws\Rds\RdsClient([
    'region' => 'us-east-1'
]);

$dbIdentifier = '<<{{db-identifier}}>>';

try {
    $result = $rdsClient->deleteDBInstance([
        'DBInstanceIdentifier' => $dbIdentifier,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    echo $e->getMessage();
    echo "\n";
}
```

- For API details, see DeleteDBInstance in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
```

```
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- For API details, see DeleteDBInstance in *AWS SDK for Python (Boto3) API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Delete the specified database instance.
    ///
    /// - Parameter instanceIdentifier: The identifier of the database
    ///   instance to delete.
    func deleteDBInstance(instanceIdentifier: String) async {
        do {
            _ = try await rdsClient.deleteDBInstance(
                input: DeleteDBInstanceInput(
                    dbInstanceIdentifier: instanceIdentifier,
                    deleteAutomatedBackups: true,
                    skipFinalSnapshot: true
```

```
                )
            )
        } catch {
            print("*** Error deleting the database instance
\(instanceIdentifier): \(error.localizedDescription)")
        }
    }
```

- For API details, see DeleteDBInstance in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteDBParameterGroup` with an AWS SDK or CLI

The following code examples show how to use `DeleteDBParameterGroup`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a DB parameter group. The group cannot be a default DB parameter
group
    /// or be associated with any DB instances.
    /// </summary>
```

```
    /// <param name="name">Name of the DB parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDBParameterGroup(string name)
    {
        var response = await _amazonRDS.DeleteDBParameterGroupAsync(
            new DeleteDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
            });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteDBParameterGroup in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::DeleteDBParameterGroupRequest request;
    request.SetDBParameterGroupName(parameterGroupName);

    Aws::RDS::Model::DeleteDBParameterGroupOutcome outcome =
            client.DeleteDBParameterGroup(request);

    if (outcome.IsSuccess()) {
        std::cout << "The DB parameter group was successfully deleted."
                  << std::endl;
    }
```

```
        else {
            std::cerr << "Error with RDS::DeleteDBParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- For API details, see DeleteDBParameterGroup in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete a DB parameter group

The following command example deletes a DB parameter group.

```
aws rds delete-db-parameter-group \
    --db-parameter-group-name mydbparametergroup
```

This command produces no output.

For more information, see Working with DB Parameter Groups in the *Amazon RDS User Guide*.

- For API details, see DeleteDBParameterGroup in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
```

```
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// DeleteParameterGroup deletes the named DB parameter group.
func (instances *DbInstances) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := instances.RdsClient.DeleteDBParameterGroup(ctx,
  &rds.DeleteDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see DeleteDBParameterGroup in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   // Delete the parameter group after database has been deleted.
   // An exception is thrown if you attempt to delete the para group while
database
   // exists.
   public static void deleteParaGroup(RdsClient rdsClient, String dbGroupName,
String dbARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(dbARN) == 0) {
                       System.out.println(dbARN + " still exists");
                       didFind = true;
                   }
                   if ((index == listSize) && (!didFind)) {
                       // Went through the entire list and did not find the
database ARN.
                       isDataDel = true;
                   }
                   Thread.sleep(sleepTime * 1000);
                   index++;
               }
           }

           // Delete the para group.
           DeleteDbParameterGroupRequest parameterGroupRequest =
DeleteDbParameterGroupRequest.builder()
                   .dbParameterGroupName(dbGroupName)
                   .build();

           rdsClient.deleteDBParameterGroup(parameterGroupRequest);
```

```
            System.out.println(dbGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- For API details, see DeleteDBParameterGroup in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB parameter group.
```

```
        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            self.rds_client.delete_db_parameter_group(
                DBParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- For API details, see DeleteDBParameterGroup in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
" iv_dbparametergroupname = 'mydbparametergroup'
TRY.
    lo_rds->deletedbparametergroup(
      iv_dbparametergroupname = iv_dbparametergroupname ).
    MESSAGE 'DB parameter group deleted.' TYPE 'I'.
  CATCH /aws1/cx_rdsdbprmgrnotfndfault.
    MESSAGE 'DB parameter group not found.' TYPE 'I'.
  CATCH /aws1/cx_rdsinvdbprmgrstatef00.
    MESSAGE 'DB parameter group is in an invalid state.' TYPE 'I'.
```

```
    ENDTRY.
```

- For API details, see DeleteDBParameterGroup in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Delete the specified database parameter group.
    ///
    /// - Parameter groupName: The name of the parameter group to delete.
    func deleteDBParameterGroup(groupName: String) async {
        do {
            _ = try await rdsClient.deleteDBParameterGroup(
                input: DeleteDBParameterGroupInput(
                    dbParameterGroupName: groupName
                )
            )
        } catch {
            print("*** Error deleting the database parameter group \(groupName):
 \(error.localizedDescription)")
        }
    }
```

- For API details, see DeleteDBParameterGroup in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeAccountAttributes` with an AWS SDK or CLI

The following code examples show how to use `DescribeAccountAttributes`.

CLI

AWS CLI

To describe account attributes

The following `describe-account-attributes` example retrieves the attributes for the current AWS account.

```
aws rds describe-account-attributes
```

Output:

```
{
    "AccountQuotas": [
        {
            "Max": 40,
            "Used": 4,
            "AccountQuotaName": "DBInstances"
        },
        {
            "Max": 40,
            "Used": 0,
            "AccountQuotaName": "ReservedDBInstances"
        },
        {
            "Max": 100000,
            "Used": 40,
            "AccountQuotaName": "AllocatedStorage"
        },
        {
            "Max": 25,
            "Used": 0,
            "AccountQuotaName": "DBSecurityGroups"
        },
        {
            "Max": 20,
            "Used": 0,
            "AccountQuotaName": "AuthorizationsPerDBSecurityGroup"
```

```
        },
        {
            "Max": 50,
            "Used": 1,
            "AccountQuotaName": "DBParameterGroups"
        },
        {
            "Max": 100,
            "Used": 3,
            "AccountQuotaName": "ManualSnapshots"
        },
        {
            "Max": 20,
            "Used": 0,
            "AccountQuotaName": "EventSubscriptions"
        },
        {
            "Max": 50,
            "Used": 1,
            "AccountQuotaName": "DBSubnetGroups"
        },
        {
            "Max": 20,
            "Used": 1,
            "AccountQuotaName": "OptionGroups"
        },
        {
            "Max": 20,
            "Used": 6,
            "AccountQuotaName": "SubnetsPerDBSubnetGroup"
        },
        {
            "Max": 5,
            "Used": 0,
            "AccountQuotaName": "ReadReplicasPerMaster"
        },
        {
            "Max": 40,
            "Used": 1,
            "AccountQuotaName": "DBClusters"
        },
        {
            "Max": 50,
            "Used": 0,
```

```
            "AccountQuotaName": "DBClusterParameterGroups"
        },
        {
            "Max": 5,
            "Used": 0,
            "AccountQuotaName": "DBClusterRoles"
        }
    ]
}
```

- For API details, see DescribeAccountAttributes in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.model.AccountQuota;
import software.amazon.awssdk.services.rds.model.RdsException;
import
 software.amazon.awssdk.services.rds.model.DescribeAccountAttributesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeAccountAttributes {
    public static void main(String[] args) {
        Region region = Region.US_WEST_2;
```

```
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        getAccountAttributes(rdsClient);
        rdsClient.close();
    }

    public static void getAccountAttributes(RdsClient rdsClient) {
        try {
            DescribeAccountAttributesResponse response =
 rdsClient.describeAccountAttributes();
            List<AccountQuota> quotasList = response.accountQuotas();
            for (AccountQuota quotas : quotasList) {
                System.out.println("Name is: " + quotas.accountQuotaName());
                System.out.println("Max value is " + quotas.max());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DescribeAccountAttributes in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getAccountAttributes() {
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeAccountAttributes(DescribeAccountAttributesRequest {})
```

```
        response.accountQuotas?.forEach { quotas ->
            val response = response.accountQuotas
            println("Name is: ${quotas.accountQuotaName}")
            println("Max value is ${quotas.max}")
        }
    }
}
```

- For API details, see DescribeAccountAttributes in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeDBEngineVersions` with an AWS SDK or CLI

The following code examples show how to use `DescribeDBEngineVersions`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
```

```
    /// <param name="dbParameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>List of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>> DescribeDBEngineVersions(string
 engine,
        string dbParameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = dbParameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
```

```
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::getDBEngineVersions(const Aws::String &engineName,
                                      const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                      const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // Used for pagination.

    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }


        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            auto &engineVersions = outcome.GetResult().GetDBEngineVersions();
            engineVersionsResult.insert(engineVersionsResult.end(),
 engineVersions.begin(),
                                        engineVersions.end());
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with RDS::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }

    } while (!marker.empty());
```

```
    return true;
}
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To describe the DB engine versions for the MySQL DB engine

The following `describe-db-engine-versions` example displays details about each of the DB engine versions for the specified DB engine.

```
aws rds describe-db-engine-versions \
    --engine mysql
```

Output:

```
{
    "DBEngineVersions": [
        {
            "Engine": "mysql",
            "EngineVersion": "5.5.46",
            "DBParameterGroupFamily": "mysql5.5",
            "DBEngineDescription": "MySQL Community Edition",
            "DBEngineVersionDescription": "MySQL 5.5.46",
            "ValidUpgradeTarget": [
                {
                    "Engine": "mysql",
                    "EngineVersion": "5.5.53",
                    "Description": "MySQL 5.5.53",
                    "AutoUpgrade": false,
                    "IsMajorVersionUpgrade": false
                },
                {
                    "Engine": "mysql",
                    "EngineVersion": "5.5.54",
                    "Description": "MySQL 5.5.54",
```

```
                "AutoUpgrade": false,
                "IsMajorVersionUpgrade": false
            },
            {
                "Engine": "mysql",
                "EngineVersion": "5.5.57",
                "Description": "MySQL 5.5.57",
                "AutoUpgrade": false,
                "IsMajorVersionUpgrade": false
            },
            ...some output truncated...
        ]
    }
```

For more information, see What Is Amazon Relational Database Service (Amazon RDS)? in the *Amazon RDS User Guide*.

- For API details, see DescribeDBEngineVersions in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
```

```
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (instances *DbInstances) GetEngineVersions(ctx context.Context, engine
 string, parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := instances.RdsClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .defaultOnly(true)
```

```
                    .engine("mysql")
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
engineOb.engine());
                System.out.println("The version number of the database engine " +
engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
```

```
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    " iv_engine                = 'mysql'
    " iv_dbparametergroupfamily = 'mysql8.0' (optional - filters by parameter
group family)
    TRY.
        oo_result = lo_rds->describedbengineversions(
          iv_engine                = iv_engine
          iv_dbparametergroupfamily = iv_dbparametergroupfamily ).
        DATA(lv_version_count) = lines( oo_result->get_dbengineversions( ) ).
        MESSAGE |Retrieved { lv_version_count } engine versions.| TYPE 'I'.
    ENDTRY.
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Get all the database engine versions available for the specified
    /// database engine.
    ///
    /// - Parameter engineName: The name of the database engine to query.
    ///
    /// - Returns: An array of `RDSClientTypes.DBEngineVersion` structures,
    ///   each describing one supported version of the specified database.
    func getDBEngineVersions(engineName: String) async ->
 [RDSClientTypes.DBEngineVersion] {
        do {
            let output = try await rdsClient.describeDBEngineVersions(
                input: DescribeDBEngineVersionsInput(
                    engine: engineName
                )
            )

            return output.dbEngineVersions ?? []
        } catch {
            return []
        }
    }
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeDBInstances` with an AWS SDK or CLI

The following code examples show how to use `DescribeDBInstances`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Returns a list of DB instances.
   /// </summary>
   /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
   /// <returns>List of DB instances.</returns>
   public async Task<List<DBInstance>> DescribeDBInstances(string
dbInstanceIdentifier = null)
   {
       var results = new List<DBInstance>();
       var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
           new DescribeDBInstancesRequest
           {
               DBInstanceIdentifier = dbInstanceIdentifier
           });
       // Get the entire list using the paginator.
       await foreach (var instances in instancesPaginator.DBInstances)
       {
           results.Add(instances);
       }
       return results;
   }
```

- For API details, see DescribeDBInstances in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets a DB instance description.
/*!
 \sa describeDBInstance()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                     Aws::RDS::Model::DBInstance &instanceResult,
                                     const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with RDS::DescribeDBInstances. "
```

```
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

- For API details, see DescribeDBInstances in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To describe a DB instance

The following `describe-db-instances` example retrieves details about the specified DB instance.

```
aws rds describe-db-instances \
    --db-instance-identifier mydbinstancecf
```

Output:

```
{
    "DBInstances": [
        {
            "DBInstanceIdentifier": "mydbinstancecf",
            "DBInstanceClass": "db.t3.small",
            "Engine": "mysql",
            "DBInstanceStatus": "available",
            "MasterUsername": "masterawsuser",
            "Endpoint": {
                "Address": "mydbinstancecf.abcexample.us-
east-1.rds.amazonaws.com",
                "Port": 3306,
                "HostedZoneId": "Z2R2ITUGPM61AM"
```

```
            },
            ...some output truncated...
        }
    ]
}
```

- For API details, see DescribeDBInstances in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (instances *DbInstances) GetInstance(ctx context.Context, instanceName
 string) (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
```

```
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.model.DescribeDbInstancesResponse;
import software.amazon.awssdk.services.rds.model.DBInstance;
import software.amazon.awssdk.services.rds.model.RdsException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeDBInstances {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeInstances(rdsClient);
        rdsClient.close();
    }

    public static void describeInstances(RdsClient rdsClient) {
        try {
            DescribeDbInstancesResponse response =
 rdsClient.describeDBInstances();
            List<DBInstance> instanceList = response.dbInstances();
            for (DBInstance instance : instanceList) {
                System.out.println("Instance ARN is: " +
 instance.dbInstanceArn());
                System.out.println("The Engine is " + instance.engine());
                System.out.println("Connection endpoint is" +
 instance.endpoint().address());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun describeInstances() {
    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbInstances(DescribeDbInstancesRequest
 {})
        response.dbInstances?.forEach { instance ->
            println("Instance Identifier is ${instance.dbInstanceIdentifier}")
            println("The Engine is ${instance.engine}")
            println("Connection endpoint is ${instance.endpoint?.address}")
        }
    }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for Kotlin API reference*.

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require __DIR__ . '/vendor/autoload.php';

use Aws\Exception\AwsException;
```

```
//Create an RDSClient
$rdsClient = new Aws\Rds\RdsClient([
    'region' => 'us-east-2'
]);

try {
    $result = $rdsClient->describeDBInstances();
    foreach ($result['DBInstances'] as $instance) {
        print('<p>DB Identifier: ' . $instance['DBInstanceIdentifier']);
        print('<br />Endpoint: ' . $instance['Endpoint']["Address"]
            . ':' . $instance['Endpoint']["Port"]);
        print('<br />Current Status: ' . $instance["DBInstanceStatus"]);
        print('</p>');
    }
    print(" Raw Result ");
    var_dump($result);
} catch (AwsException $e) {
    echo $e->getMessage();
    echo "\n";
}
```

- For API details, see DescribeDBInstances in *AWS SDK for PHP API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
```

```
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst
```

- For API details, see DescribeDBInstances in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-rds' # v2: require 'aws-sdk'

# List all Amazon Relational Database Service (Amazon RDS) DB instances.
#
# @param rds_resource [Aws::RDS::Resource] An SDK for Ruby Amazon RDS resource.
# @return [Array, nil] List of all DB instances, or nil if error.
def list_instances(rds_resource)
  db_instances = []
  rds_resource.db_instances.each do |i|
    db_instances.append({
                          "name": i.id,
                          "status": i.db_instance_status
                        })
  end
  db_instances
rescue Aws::Errors::ServiceError => e
  puts "Couldn't list instances:\n#{e.message}"
end
```

- For API details, see DescribeDBInstances in *AWS SDK for Ruby API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Wait until the specified database is available to use.
    ///
    /// - Parameter instanceIdentifier: The database instance identifier of the
    ///   database to wait for.
    func waitUntilDBInstanceReady(instanceIdentifier: String) async ->
 RDSClientTypes.Endpoint? {
        do {
            putString("Waiting for the database instance to be ready to use. This
 may take 10 minutes or more...")
            while true {
                let output = try await rdsClient.describeDBInstances(
                    input: DescribeDBInstancesInput(
                        dbInstanceIdentifier: instanceIdentifier
                    )
                )

                guard let instanceList = output.dbInstances else {
                    continue
                }

                for instance in instanceList {
                    let status = instance.dbInstanceStatus

                    guard let status else {
                        print("\nUnable to determine the status.")
                        continue
                    }

                    if status.contains("available") {
                        return instance.endpoint
                    } else {
                        putString(".")
                        do {
                            try await Task.sleep(for: .seconds(15))
                        } catch {
                            print("*** Error pausing the task!")
                        }
                    }
                }
            }
        } catch {
```

```
            print("*** Unable to wait until the database is ready:
 \(error.localizedDescription)")
            return nil
        }
    }
```

- For API details, see DescribeDBInstances in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeDBParameterGroups` with an AWS SDK or CLI

The following code examples show how to use `DescribeDBParameterGroups`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get descriptions of DB parameter groups.
    /// </summary>
    /// <param name="name">Optional name of the DB parameter group to describe.</
param>
    /// <returns>The list of DB parameter group descriptions.</returns>
```

```
    public async Task<List<DBParameterGroup>> DescribeDBParameterGroups(string
 name = null)
    {
        var response = await _amazonRDS.DescribeDBParameterGroupsAsync(
            new DescribeDBParameterGroupsRequest()
            {
                DBParameterGroupName = name
            });
        return response.DBParameterGroups;
    }
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DescribeDBParameterGroupsRequest request;
        request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBParameterGroupsOutcome outcome =
                client.DescribeDBParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB parameter group named '" <<
                      PARAMETER_GROUP_NAME << "' already exists." << std::endl;
            dbParameterGroupFamily = outcome.GetResult().GetDBParameterGroups()
[0].GetDBParameterGroupFamily();
```

```
        }

        else {
            std::cerr << "Error with RDS::DescribeDBParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To describe your DB parameter group

The following `describe-db-parameter-groups` example retrieves details about your DB parameter groups.

```
aws rds describe-db-parameter-groups
```

Output:

```
{
    "DBParameterGroups": [
        {
            "DBParameterGroupName": "default.aurora-mysql5.7",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "Default parameter group for aurora-mysql5.7",
            "DBParameterGroupArn": "arn:aws:rds:us-
east-1:123456789012:pg:default.aurora-mysql5.7"
        },
        {
            "DBParameterGroupName": "default.aurora-postgresql9.6",
            "DBParameterGroupFamily": "aurora-postgresql9.6",
            "Description": "Default parameter group for aurora-postgresql9.6",
            "DBParameterGroupArn": "arn:aws:rds:us-
east-1:123456789012:pg:default.aurora-postgresql9.6"
        },
        {
```

```
            "DBParameterGroupName": "default.aurora5.6",
            "DBParameterGroupFamily": "aurora5.6",
            "Description": "Default parameter group for aurora5.6",
            "DBParameterGroupArn": "arn:aws:rds:us-
east-1:123456789012:pg:default.aurora5.6"
        },
        {
            "DBParameterGroupName": "default.mariadb10.1",
            "DBParameterGroupFamily": "mariadb10.1",
            "Description": "Default parameter group for mariadb10.1",
            "DBParameterGroupArn": "arn:aws:rds:us-
east-1:123456789012:pg:default.mariadb10.1"
        },
        ...some output truncated...
    ]
}
```

For more information, see Working with DB Parameter Groups in the *Amazon RDS User Guide*.

- For API details, see DescribeDBParameterGroups in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbInstances struct {
 RdsClient *rds.Client
}



// GetParameterGroup gets a DB parameter group by name.
func (instances *DbInstances) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBParameterGroup, error) {
 output, err := instances.RdsClient.DescribeDBParameterGroups(
  ctx, &rds.DescribeDBParameterGroupsInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBParameterGroups[0], err
 }
}
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void describeDbParameterGroups(RdsClient rdsClient, String
dbGroupName) {
       try {
           DescribeDbParameterGroupsRequest groupsRequest =
DescribeDbParameterGroupsRequest.builder()
                   .dbParameterGroupName(dbGroupName)
                   .maxRecords(20)
                   .build();

           DescribeDbParameterGroupsResponse response =
rdsClient.describeDBParameterGroups(groupsRequest);
           List<DBParameterGroup> groups = response.dbParameterGroups();
           for (DBParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbParameterGroupName());
               System.out.println("The group description is " +
group.description());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""
```

```
    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The parameter group.
        """
        try:
            response = self.rds_client.describe_db_parameter_groups(
                DBParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-rds' # v2: require 'aws-sdk'

# List all Amazon Relational Database Service (Amazon RDS) parameter groups.
#
# @param rds_resource [Aws::RDS::Resource] An SDK for Ruby Amazon RDS resource.
# @return [Array, nil] List of all parameter groups, or nil if error.
def list_parameter_groups(rds_resource)
  parameter_groups = []
  rds_resource.db_parameter_groups.each do |p|
    parameter_groups.append({
                              "name": p.db_parameter_group_name,
                              "description": p.description
                            })
  end
  parameter_groups
rescue Aws::Errors::ServiceError => e
  puts "Couldn't list parameter groups:\n #{e.message}"
end
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
" iv_dbparametergroupname = 'mydbparametergroup'
TRY.
    oo_result = lo_rds->describedbparametergroups(
      iv_dbparametergroupname = iv_dbparametergroupname ).
    MESSAGE 'DB parameter group retrieved.' TYPE 'I'.
  CATCH /aws1/cx_rdsdbprmgrnotfndfault.
    MESSAGE 'DB parameter group not found.' TYPE 'I'.
ENDTRY.
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Get descriptions of the database parameter groups matching the given
    /// name.
    ///
    /// - Parameter groupName: The name of the parameter group to describe.
    ///
    /// - Returns: An array of [RDSClientTypes.DBParameterGroup] objects
```

```
    ///   describing the parameter group.
    func describeDBParameterGroups(groupName: String) async ->
[RDSClientTypes.DBParameterGroup]? {
        do {
            let output = try await rdsClient.describeDBParameterGroups(
                input: DescribeDBParameterGroupsInput(
                    dbParameterGroupName: groupName
                )
            )
            return output.dbParameterGroups
        } catch {
            print("*** Error getting the database parameter group's details:
\(error.localizedDescription)")
            return nil
        }
    }
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeDBParameters` with an AWS SDK or CLI

The following code examples show how to use `DescribeDBParameters`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Get a list of DB parameters from a specific parameter group.
   /// </summary>
   /// <param name="dbParameterGroupName">Name of a specific DB parameter
group.</param>
   /// <param name="source">Optional source for selecting parameters.</param>
   /// <returns>List of parameter values.</returns>
   public async Task<List<Parameter>> DescribeDBParameters(string
dbParameterGroupName, string source = null)
   {
       var results = new List<Parameter>();
       var paginateParameters = _amazonRDS.Paginators.DescribeDBParameters(
           new DescribeDBParametersRequest()
           {
               DBParameterGroupName = dbParameterGroupName,
               Source = source
           });
       // Get the entire list using the paginator.
       await foreach (var parameters in paginateParameters.Parameters)
       {
           results.Add(parameters);
       }
       return results;
   }
```

- For API details, see DescribeDBParameters in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets DB parameters using the 'DescribeDBParameters' api.
/*!
 \sa getDBParameters()
 \param parameterGroupName: The name of the parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::getDBParameters(const Aws::String &parameterGroupName,
                                  const Aws::String &namePrefix,
                                  const Aws::String &source,
                                  Aws::Vector<Aws::RDS::Model::Parameter>
 &parametersResult,
                                  const Aws::RDS::RDSClient &client) {
    Aws::String marker;
    do {
        Aws::RDS::Model::DescribeDBParametersRequest request;
        request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
```

```
        }

        Aws::RDS::Model::DescribeDBParametersOutcome outcome =
                client.DescribeDBParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with RDS::DescribeDBParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

- For API details, see DescribeDBParameters in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To describe the parameters in a DB parameter group

The following `describe-db-parameters` example retrieves the details of the specified DB parameter group.

```
aws rds describe-db-parameters \
    --db-parameter-group-name mydbpg
```

Output:

```
{
    "Parameters": [
        {
            "ParameterName": "allow-suspicious-udfs",
            "Description": "Controls whether user-defined functions that have
 only an xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot"
        },
        {
            "ParameterName": "auto_generate_certs",
            "Description": "Controls whether the server autogenerates SSL key and
 certificate files in the data directory, if they do not already exist.",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot"
        },
        ...some output truncated...
    ]
}
```

For more information, see Working with DB Parameter Groups in the *Amazon RDS User Guide*.

- For API details, see DescribeDBParameters in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB parameter group.
func (instances *DbInstances) GetParameters(ctx context.Context,
 parameterGroupName string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator := rds.NewDescribeDBParametersPaginator(instances.RdsClient,
  &rds.DescribeDBParametersInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Source:               aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
```

```
  if err != nil {
   log.Printf("Couldn't get parameters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- For API details, see DescribeDBParameters in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   // Retrieve parameters in the group.
   public static void describeDbParameters(RdsClient rdsClient, String
dbGroupName, int flag) {
       try {
           DescribeDbParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest = DescribeDbParametersRequest.builder()
                       .dbParameterGroupName(dbGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest = DescribeDbParametersRequest.builder()
                       .dbParameterGroupName(dbGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbParametersResponse response =
rdsClient.describeDBParameters(dbParameterGroupsRequest);
```

```
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- For API details, see DescribeDBParameters in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
```

```
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator = self.rds_client.get_paginator("describe_db_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
```

```
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters
```

- For API details, see DescribeDBParameters in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-rds' # v2: require 'aws-sdk'

# List all Amazon Relational Database Service (Amazon RDS) parameter groups.
#
# @param rds_resource [Aws::RDS::Resource] An SDK for Ruby Amazon RDS resource.
# @return [Array, nil] List of all parameter groups, or nil if error.
def list_parameter_groups(rds_resource)
  parameter_groups = []
  rds_resource.db_parameter_groups.each do |p|
    parameter_groups.append({
                              "name": p.db_parameter_group_name,
                              "description": p.description
                            })
  end
  parameter_groups
rescue Aws::Errors::ServiceError => e
```

```
  puts "Couldn't list parameter groups:\n #{e.message}"
end
```

- For API details, see DescribeDBParameters in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
" iv_dbparametergroupname = 'mydbparametergroup'
" iv_source               = 'user' (optional - filters by parameter source)
TRY.
    oo_result = lo_rds->describedbparameters(
      iv_dbparametergroupname = iv_dbparametergroupname
      iv_source               = iv_source ).
    DATA(lv_param_count) = lines( oo_result->get_parameters( ) ).
    MESSAGE |Retrieved { lv_param_count } parameters.| TYPE 'I'.
  CATCH /aws1/cx_rdsdbprmgrnotfndfault.
    MESSAGE 'DB parameter group not found.' TYPE 'I'.
ENDTRY.
```

- For API details, see DescribeDBParameters in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Returns the detailed parameter list for the specified database
    /// parameter group.
    ///
    /// - Parameters:
    ///   - groupName: The name of the parameter group to return parameters for.
    ///   - source: The types of parameters to return (`user`, `system`, or
    ///     `engine-default`).
    ///
    /// - Returns: An array of `RdSClientTypes.Parameter` objects, each
    ///   describing one of the group's parameters.
    func describeDBParameters(groupName: String, source: String? = nil) async ->
 [RDSClientTypes.Parameter] {
        var parameterList: [RDSClientTypes.Parameter] = []

        do {
            let pages = rdsClient.describeDBParametersPaginated(
                input: DescribeDBParametersInput(
                    dbParameterGroupName: groupName,
                    source: source
                )
            )

            for try await page in pages {
                guard let parameters = page.parameters else {
                    return []
                }

                parameterList += parameters
            }
        } catch {
            print("*** Error getting database parameters:
 \(error.localizedDescription)")
            return []
        }

        return parameterList
    }
```

- For API details, see DescribeDBParameters in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeDBSnapshots` with an AWS SDK or CLI

The following code examples show how to use `DescribeDBSnapshots`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Return a list of DB snapshots for a particular DB instance.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <returns>List of DB snapshots.</returns>
   public async Task<List<DBSnapshot>> DescribeDBSnapshots(string
dbInstanceIdentifier)
   {
       var results = new List<DBSnapshot>();
       var snapshotsPaginator = _amazonRDS.Paginators.DescribeDBSnapshots(
           new DescribeDBSnapshotsRequest()
           {
               DBInstanceIdentifier = dbInstanceIdentifier
           });

       // Get the entire list using the paginator.
       await foreach (var snapshots in snapshotsPaginator.DBSnapshots)
```

```
        {
            results.Add(snapshots);
        }
        return results;
    }
```

- For API details, see DescribeDBSnapshots in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DescribeDBSnapshotsRequest request;
            request.SetDBSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBSnapshotsOutcome outcome =
                    client.DescribeDBSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBSnapshots()[0];
            }
            else {
                std::cerr << "Error with RDS::DescribeDBSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(PARAMETER_GROUP_NAME, DB_INSTANCE_IDENTIFIER,
client);
                return false;
```

```
            }
```

- For API details, see DescribeDBSnapshots in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

Example 1: To describe a DB snapshot for a DB instance

The following `describe-db-snapshots` example retrieves the details of a DB snapshot for a DB instance.

```
aws rds describe-db-snapshots \
    --db-snapshot-identifier mydbsnapshot
```

Output:

```
{
    "DBSnapshots": [
        {
            "DBSnapshotIdentifier": "mydbsnapshot",
            "DBInstanceIdentifier": "mysqldb",
            "SnapshotCreateTime": "2018-02-08T22:28:08.598Z",
            "Engine": "mysql",
            "AllocatedStorage": 20,
            "Status": "available",
            "Port": 3306,
            "AvailabilityZone": "us-east-1f",
            "VpcId": "vpc-6594f31c",
            "InstanceCreateTime": "2018-02-08T22:24:55.973Z",
            "MasterUsername": "mysqladmin",
            "EngineVersion": "5.6.37",
            "LicenseModel": "general-public-license",
            "SnapshotType": "manual",
            "OptionGroupName": "default:mysql-5-6",
            "PercentProgress": 100,
            "StorageType": "gp2",
            "Encrypted": false,
            "DBSnapshotArn": "arn:aws:rds:us-
east-1:123456789012:snapshot:mydbsnapshot",
```

```
            "IAMDatabaseAuthenticationEnabled": false,
            "ProcessorFeatures": [],
            "DbiResourceId": "db-AKIAIOSFODNN7EXAMPLE"
        }
    ]
}
```

For more information, see Creating a DB Snapshot in the *Amazon RDS User Guide*.

Example 2: To find the number of manual snapshots taken

The following `describe-db-snapshots` example uses the `length` operator in the `--query` option to return the number of manual snapshots that have been taken in a particular AWS Region.

```
aws rds describe-db-snapshots \
    --snapshot-type manual \
    --query "length(*[].{DBSnapshots:SnapshotType})" \
    --region eu-central-1
```

Output:

```
35
```

For more information, see Creating a DB Snapshot in the *Amazon RDS User Guide*.

- For API details, see DescribeDBSnapshots in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
```

```
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetSnapshot gets a DB instance snapshot.
func (instances *DbInstances) GetSnapshot(ctx context.Context, snapshotName
 string) (*types.DBSnapshot, error) {
 output, err := instances.RdsClient.DescribeDBSnapshots(ctx,
  &rds.DescribeDBSnapshotsInput{
   DBSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBSnapshots[0], nil
 }
}
```

- For API details, see DescribeDBSnapshots in *AWS SDK for Go API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_snapshot(self, snapshot_id):
        """
        Gets a DB instance snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_snapshots(
                DBSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- For API details, see DescribeDBSnapshots in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-rds' # v2: require 'aws-sdk'

# List all Amazon Relational Database Service (Amazon RDS) DB instance
# snapshots.
#
# @param rds_resource [Aws::RDS::Resource] An SDK for Ruby Amazon RDS resource.
# @return instance_snapshots [Array, nil] All instance snapshots, or nil if
 error.
def list_instance_snapshots(rds_resource)
  instance_snapshots = []
  rds_resource.db_snapshots.each do |s|
    instance_snapshots.append({
                                "id": s.snapshot_id,
                                "status": s.status
                              })
  end
  instance_snapshots
rescue Aws::Errors::ServiceError => e
  puts "Couldn't list instance snapshots:\n #{e.message}"
end
```

- For API details, see DescribeDBSnapshots in *AWS SDK for Ruby API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Wait until the specified database snapshot is available to use.
    ///
    /// - Parameters:
    ///   - instanceIdentifier: The identifier of the database for which the
    ///     snapshot was taken.
    ///   - snapshotIdentifier: The identifier of the snapshot to wait for.
    func waitUntilDBSnapshotReady(instanceIdentifier: String, snapshotIdentifier:
 String) async {
        var snapshotReady = false

        putString("Waiting for the snapshot to be ready...")

        do {
            while !snapshotReady {
                let output = try await rdsClient.describeDBSnapshots(
                    input: DescribeDBSnapshotsInput(
                        dbInstanceIdentifier: instanceIdentifier,
                        dbSnapshotIdentifier: snapshotIdentifier
                    )
                )

                guard let snapshotList = output.dbSnapshots else {
                    return
                }

                for snapshot in snapshotList {
                    guard let snapshotReadyStr = snapshot.status else {
                        return
                    }
```

```
                    if snapshotReadyStr.contains("available") {
                        snapshotReady = true
                        print()
                    } else {
                        putString(".")
                        do {
                            try await Task.sleep(for: .seconds(15))
                        } catch {
                            print("\n*** Error pausing the task!")
                        }
                    }
                }
            }
        } catch {
            print("\n*** Unable to wait for the database snapshot to be ready:
\(error.localizedDescription)")
        }
    }
```

- For API details, see DescribeDBSnapshots in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeOrderableDBInstanceOptions` with an AWS SDK or CLI

The following code examples show how to use `DescribeOrderableDBInstanceOptions`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Get a list of orderable DB instance options for a specific
   /// engine and engine version.
   /// </summary>
   /// <param name="engine">Name of the engine.</param>
   /// <param name="engineVersion">Version of the engine.</param>
   /// <returns>List of OrderableDBInstanceOptions.</returns>
   public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptions(string engine, string engineVersion)
   {
       // Use a paginator to get a list of DB instance options.
       var results = new List<OrderableDBInstanceOption>();
       var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
           new DescribeOrderableDBInstanceOptionsRequest()
           {
               Engine = engine,
               EngineVersion = engineVersion,
           });
       // Get the entire list using the paginator.
       await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
       {
           results.Add(instanceOptions);
       }
       return results;
   }
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available 'micro' DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseMicroDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::RDS::chooseMicroDBInstanceClass(const Aws::String &engine,
                                             const Aws::String &engineVersion,
                                             Aws::String &dbInstanceClass,
                                             const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker;
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
```

```
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (instanceClass.find("micro") != std::string::npos) {
                    if (std::find(instanceClasses.begin(), instanceClasses.end(),
                                  instanceClass) ==
                        instanceClasses.end()) {
                        instanceClasses.push_back(instanceClass);
                    }
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with RDS::DescribeOrderableDBInstanceOptions. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available micro DB instance classes for your database
 engine are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which micro DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To describe orderable DB instance options

The following `describe-orderable-db-instance-options` example retrieves details about the orderable options for a DB instances running the MySQL DB engine.

```
aws rds describe-orderable-db-instance-options \
    --engine mysql
```

Output:

```
{
    "OrderableDBInstanceOptions": [
        {
            "MinStorageSize": 5,
            "ReadReplicaCapable": true,
            "MaxStorageSize": 6144,
            "AvailabilityZones": [
                {
                    "Name": "us-east-1a"
                },
                {
                    "Name": "us-east-1b"
                },
                {
                    "Name": "us-east-1c"
                },
                {
                    "Name": "us-east-1d"
                }
            ],
            "SupportsIops": false,
            "AvailableProcessorFeatures": [],
```

```
            "MultiAZCapable": true,
            "DBInstanceClass": "db.m1.large",
            "Vpc": true,
            "StorageType": "gp2",
            "LicenseModel": "general-public-license",
            "EngineVersion": "5.5.46",
            "SupportsStorageEncryption": false,
            "SupportsEnhancedMonitoring": true,
            "Engine": "mysql",
            "SupportsIAMDatabaseAuthentication": false,
            "SupportsPerformanceInsights": false
        }
    ]
    ...some output truncated...
}
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}
```

```
// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (instances *DbInstances) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instanceOptions []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(instances.RdsClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instance options: %v\n", err)
   break
  } else {
   instanceOptions = append(instanceOptions,
 output.OrderableDBInstanceOptions...)
  }
 }
 return instanceOptions, err
}
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for Go API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
 instance.
        :param db_engine_version: The engine version that must be supported by
 the DB instance.
        :return: The list of DB instance options that can be used to create a
 compatible DB instance.
        """
        try:
            inst_opts = []
```

```
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    " iv_engine        = 'mysql'
    " iv_engineversion = '8.0.35'
    TRY.
        oo_result = lo_rds->descrorderabledbinstoptions(
          iv_engine        = iv_engine
          iv_engineversion = iv_engineversion ).
        DATA(lv_option_count) = lines( oo_result-
>get_orderabledbinstoptions( ) ).
```

```
        MESSAGE |Retrieved { lv_option_count } orderable DB instance options.|
TYPE 'I'.
   ENDTRY.
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Print a list of available database instances with "micro" in the class
    /// name, then return one of them to be used by other code.
    ///
    /// - Parameters:
    ///   - engine: The database engine for which to list database instance
    ///     classes.
    ///   - engineVersion: The database version for which to list instances.
    ///
    /// - Returns: An `RDSClientTypes.OrderableDBInstanceOption` describing
    ///   the selected instance type.
    func chooseMicroInstance(engine: String = "mysql", engineVersion: String? =
nil) async -> RDSClientTypes.OrderableDBInstanceOption? {
        do {
            let pages = rdsClient.describeOrderableDBInstanceOptionsPaginated(
                input: DescribeOrderableDBInstanceOptionsInput(
                    engine: engine,
                    engineVersion: engineVersion
                )
            )

            var optionsList: [RDSClientTypes.OrderableDBInstanceOption] = []
```

```
            for try await page in pages {
                guard let orderableDBInstanceOptions =
page.orderableDBInstanceOptions else {
                    continue
                }

                for dbInstanceOption in orderableDBInstanceOptions {
                    guard let className = dbInstanceOption.dbInstanceClass else {
                        continue
                    }
                    if className.contains("micro") {
                        optionsList.append(dbInstanceOption)
                    }
                }
            }

            print("Found \(optionsList.count) database instances of 'micro' class
types:")
            for dbInstanceOption in optionsList {
                print("    \(dbInstanceOption.engine ??
"<unknown>") \(dbInstanceOption.engineVersion ?? "<unknown>")
(\(dbInstanceOption.dbInstanceClass ?? "<unknown class>"))")
            }

            return optionsList[0]
        } catch {
            print("*** Error getting a list of orderable instance options:
\(error.localizedDescription)")
            return nil
        }
    }
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GenerateRDSAuthToken` with an AWS SDK

The following code example shows how to use `GenerateRDSAuthToken`.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Use the RdsUtilities class to generate an authentication token.

```
public class GenerateRDSAuthToken {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <dbInstanceIdentifier> <masterUsername>

                Where:
                    dbInstanceIdentifier - The database instance identifier.\s
                    masterUsername - The master user name.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbInstanceIdentifier = args[0];
        String masterUsername = args[1];
        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        String token = getAuthToken(rdsClient, dbInstanceIdentifier,
 masterUsername);
```

```
        System.out.println("The token response is " + token);
    }

    public static String getAuthToken(RdsClient rdsClient, String
 dbInstanceIdentifier, String masterUsername) {

        RdsUtilities utilities = rdsClient.utilities();
        try {
            GenerateAuthenticationTokenRequest tokenRequest =
 GenerateAuthenticationTokenRequest.builder()
                    .credentialsProvider(ProfileCredentialsProvider.create())
                    .username(masterUsername)
                    .port(3306)
                    .hostname(dbInstanceIdentifier)
                    .build();

            return utilities.generateAuthenticationToken(tokenRequest);

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see GenerateRDSAuthToken in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ModifyDBInstance` with an AWS SDK or CLI

The following code examples show how to use `ModifyDBInstance`.

CLI

AWS CLI

Example 1: To modify a DB instance

The following `modify-db-instance` example associates an option group and a parameter group with a compatible Microsoft SQL Server DB instance. The `--apply-immediately` parameter causes the option and parameter groups to be associated immediately, instead of waiting until the next maintenance window.

```
aws rds modify-db-instance \
    --db-instance-identifier database-2 \
    --option-group-name test-se-2017 \
    --db-parameter-group-name test-sqlserver-se-2017 \
    --apply-immediately
```

Output:

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "database-2",
        "DBInstanceClass": "db.r4.large",
        "Engine": "sqlserver-se",
        "DBInstanceStatus": "available",

        ...output omitted...

        "DBParameterGroups": [
            {
                "DBParameterGroupName": "test-sqlserver-se-2017",
                "ParameterApplyStatus": "applying"
            }
        ],
        "AvailabilityZone": "us-west-2d",

        ...output omitted...

        "MultiAZ": true,
        "EngineVersion": "14.00.3281.6.v1",
        "AutoMinorVersionUpgrade": false,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "license-included",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "test-se-2017",
                "Status": "pending-apply"
            }
```

```
        ],
        "CharacterSetName": "SQL_Latin1_General_CP1_CI_AS",
        "SecondaryAvailabilityZone": "us-west-2c",
        "PubliclyAccessible": true,
        "StorageType": "gp2",

        ...output omitted...

        "DeletionProtection": false,
        "AssociatedRoles": [],
        "MaxAllocatedStorage": 1000
    }
}
```

For more information, see Modifying an Amazon RDS DB Instance in the *Amazon RDS User Guide*.

Example 2: To associate VPC security group with a DB instance

The following `modify-db-instance` example associates a specific VPC security group and removes DB security groups from a DB instance:

```
aws rds modify-db-instance \
    --db-instance-identifier dbName \
    --vpc-security-group-ids sg-ID
```

Output:

```
{
"DBInstance": {
    "DBInstanceIdentifier": "dbName",
    "DBInstanceClass": "db.t3.micro",
    "Engine": "mysql",
    "DBInstanceStatus": "available",
    "MasterUsername": "admin",
    "Endpoint": {
        "Address": "dbName.abcdefghijk.us-west-2.rds.amazonaws.com",
        "Port": 3306,
        "HostedZoneId": "ABCDEFGHIJK1234"
    },
    "AllocatedStorage": 20,
    "InstanceCreateTime": "2024-02-15T00:37:58.793000+00:00",
    "PreferredBackupWindow": "11:57-12:27",
```

```
    "BackupRetentionPeriod": 7,
    "DBSecurityGroups": [],
    "VpcSecurityGroups": [
        {
            "VpcSecurityGroupId": "sg-ID",
            "Status": "active"
        }
    ],
    ... output omitted ...
    "MultiAZ": false,
    "EngineVersion": "8.0.35",
    "AutoMinorVersionUpgrade": true,
    "ReadReplicaDBInstanceIdentifiers": [],
    "LicenseModel": "general-public-license",

    ... output omitted ...
    }
}
```

For more information, see Controlling access with security groups in the *Amazon RDS User Guide*.

- For API details, see ModifyDBInstance in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.model.ModifyDbInstanceRequest;
import software.amazon.awssdk.services.rds.model.ModifyDbInstanceResponse;
import software.amazon.awssdk.services.rds.model.RdsException;

/**
 * Before running this Java V2 code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ModifyDBInstance {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <dbInstanceIdentifier> <dbSnapshotIdentifier>\s
                Where:
                    dbInstanceIdentifier - The database instance identifier.\s
                    masterUserPassword - The updated password that corresponds to
 the master user name.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbInstanceIdentifier = args[0];
        String masterUserPassword = args[1];
        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        updateIntance(rdsClient, dbInstanceIdentifier, masterUserPassword);
        rdsClient.close();
    }

    public static void updateIntance(RdsClient rdsClient, String
 dbInstanceIdentifier, String masterUserPassword) {
        try {
            // For a demo - modify the DB instance by modifying the master
 password.
            ModifyDbInstanceRequest modifyDbInstanceRequest =
 ModifyDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .publiclyAccessible(true)
```

```
                    .masterUserPassword(masterUserPassword)
                    .build();

            ModifyDbInstanceResponse instanceResponse =
 rdsClient.modifyDBInstance(modifyDbInstanceRequest);
            System.out.print("The ARN of the modified database is: " +
 instanceResponse.dbInstance().dbInstanceArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ModifyDBInstance in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun updateIntance(
    dbInstanceIdentifierVal: String?,
    masterUserPasswordVal: String?,
) {
    val request =
        ModifyDbInstanceRequest {
            dbInstanceIdentifier = dbInstanceIdentifierVal
            publiclyAccessible = true
            masterUserPassword = masterUserPasswordVal
        }

    RdsClient.fromEnvironment { region = "us-west-2" }.use { rdsClient ->
        val instanceResponse = rdsClient.modifyDbInstance(request)
```

```
        println("The ARN of the modified database is
 ${instanceResponse.dbInstance?.dbInstanceArn}")
    }
}
```

- For API details, see ModifyDBInstance in *AWS SDK for Kotlin API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ModifyDBParameterGroup` with an AWS SDK or CLI

The following code examples show how to use `ModifyDBParameterGroup`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Learn the basics

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Update a DB parameter group. Use the action
 DescribeDBParameterGroupsAsync
    /// to determine when the DB parameter group is ready to use.
    /// </summary>
    /// <param name="name">Name of the DB parameter group.</param>
    /// <param name="parameters">List of parameters. Maximum of 20 per request.</
param>
```

```
    /// <returns>The updated DB parameter group name.</returns>
    public async Task<string> ModifyDBParameterGroup(
        string name, List<Parameter> parameters)
    {
        var response = await _amazonRDS.ModifyDBParameterGroupAsync(
            new ModifyDBParameterGroupRequest()
            {
                DBParameterGroupName = name,
                Parameters = parameters,
            });
        return response.DBParameterGroupName;
    }
```

- For API details, see ModifyDBParameterGroup in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::ModifyDBParameterGroupRequest request;
    request.SetDBParameterGroupName(PARAMETER_GROUP_NAME);
    request.SetParameters(updateParameters);

    Aws::RDS::Model::ModifyDBParameterGroupOutcome outcome =
            client.ModifyDBParameterGroup(request);

    if (outcome.IsSuccess()) {
        std::cout << "The DB parameter group was successfully modified."
```

```
                    << std::endl;
        }
        else {
            std::cerr << "Error with RDS::ModifyDBParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- For API details, see ModifyDBParameterGroup in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To modify a DB parameter group

The following `modify-db-parameter-group` example changes the value of the `clr enabled` parameter in a DB parameter group. The `--apply-immediately` parameter causes the DB parameter group to be modified immediately, instead of waiting until the next maintenance window.

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name test-sqlserver-se-2017 \
    --parameters "ParameterName='clr
 enabled',ParameterValue=1,ApplyMethod=immediate"
```

Output:

```
{
    "DBParameterGroupName": "test-sqlserver-se-2017"
}
```

For more information, see Modifying Parameters in a DB Parameter Group in the *Amazon RDS User Guide*.

- For API details, see ModifyDBParameterGroup in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// UpdateParameters updates parameters in a named DB parameter group.
func (instances *DbInstances) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := instances.RdsClient.ModifyDBParameterGroup(ctx,
  &rds.ModifyDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Parameters:           params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see ModifyDBParameterGroup in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   // Modify auto_increment_offset and auto_increment_increment parameters.
   public static void modifyDBParas(RdsClient rdsClient, String dbGroupName) {
       try {
           Parameter parameter1 = Parameter.builder()
                   .parameterName("auto_increment_offset")
                   .applyMethod("immediate")
                   .parameterValue("5")
                   .build();

           List<Parameter> paraList = new ArrayList<>();
           paraList.add(parameter1);
           ModifyDbParameterGroupRequest groupRequest =
ModifyDbParameterGroupRequest.builder()
                   .dbParameterGroupName(dbGroupName)
                   .parameters(paraList)
                   .build();

           ModifyDbParameterGroupResponse response =
rdsClient.modifyDBParameterGroup(groupRequest);
           System.out.println("The parameter group " +
response.dbParameterGroupName() + " was successfully modified");

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- For API details, see ModifyDBParameterGroup in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class InstanceWrapper:
    """Encapsulates Amazon RDS DB instance actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon RDS client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_parameter_group(
```

```
                DBParameterGroupName=parameter_group_name,
Parameters=update_parameters
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- For API details, see ModifyDBParameterGroup in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
" iv_dbparametergroupname = 'mydbparametergroup'
" it_parameters - table containing parameter objects with:
"   - parametername = 'max_connections'
"   - parametervalue = '100'
"   - applymethod = 'immediate' or 'pending-reboot'
TRY.
    oo_result = lo_rds->modifydbparametergroup(
      iv_dbparametergroupname = iv_dbparametergroupname
      it_parameters           = it_parameters ).
    MESSAGE 'DB parameter group modified.' TYPE 'I'.
  CATCH /aws1/cx_rdsdbprmgrnotfndfault.
    MESSAGE 'DB parameter group not found.' TYPE 'I'.
```

```
      CATCH /aws1/cx_rdsinvdbprmgrstatef00.
        MESSAGE 'DB parameter group is in an invalid state.' TYPE 'I'.
    ENDTRY.
```

- For API details, see ModifyDBParameterGroup in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSRDS

    /// Demonstrates modifying two of the specified database parameter group's
    /// parameters.
    ///
    /// - Parameter groupName: The name of the parameter group to change
    ///   parameters for.
    func modifyDBParameters(groupName: String) async {
        let parameter1 = RDSClientTypes.Parameter(
            applyMethod: RDSClientTypes.ApplyMethod.immediate,
            parameterName: "auto_increment_offset",
            parameterValue: "5"
        )
        let parameter2 = RDSClientTypes.Parameter(
            applyMethod: RDSClientTypes.ApplyMethod.immediate,
            parameterName: "auto_increment_increment",
            parameterValue: "5"
        )

        let parameterList = [parameter1, parameter2]

        do {
            _ = try await rdsClient.modifyDBParameterGroup(
                input: ModifyDBParameterGroupInput(
                    dbParameterGroupName: groupName,
```

```
                    parameters: parameterList
                )
            )

            print("Successfully modified the parameter group \(groupName).")
        } catch {
            print("*** Error modifying the parameter group \(groupName):
 \(error.localizedDescription)")
        }
    }
```

- For API details, see ModifyDBParameterGroup in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `RebootDBInstance` with an AWS SDK or CLI

The following code examples show how to use `RebootDBInstance`.

CLI

AWS CLI

To reboot a DB instance

The following `reboot-db-instance` example starts a reboot of the specified DB instance.

```
aws rds reboot-db-instance \
    --db-instance-identifier test-mysql-instance
```

Output:

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "test-mysql-instance",
        "DBInstanceClass": "db.t3.micro",
        "Engine": "mysql",
        "DBInstanceStatus": "rebooting",
        "MasterUsername": "admin",
```

```
        "Endpoint": {
            "Address": "test-mysql-instance.############.us-
west-2.rds.amazonaws.com",
            "Port": 3306,
            "HostedZoneId": "Z1PVIF0EXAMPLE"
        },

    ... output omitted...

    }
}
```

For more information, see Rebooting a DB Instance in the *Amazon RDS User Guide*.

- For API details, see RebootDBInstance in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.model.RebootDbInstanceRequest;
import software.amazon.awssdk.services.rds.model.RebootDbInstanceResponse;
import software.amazon.awssdk.services.rds.model.RdsException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class RebootDBInstance {
```

```
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <dbInstanceIdentifier>\s

                Where:
                    dbInstanceIdentifier - The database instance identifier\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbInstanceIdentifier = args[0];
        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        rebootInstance(rdsClient, dbInstanceIdentifier);
        rdsClient.close();
    }

    public static void rebootInstance(RdsClient rdsClient, String
 dbInstanceIdentifier) {
        try {
            RebootDbInstanceRequest rebootDbInstanceRequest =
 RebootDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            RebootDbInstanceResponse instanceResponse =
 rdsClient.rebootDBInstance(rebootDbInstanceRequest);
            System.out.print("The database " +
 instanceResponse.dbInstance().dbInstanceArn() + " was rebooted");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see RebootDBInstance in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon RDS using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon RDS with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon RDS or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Create an Aurora Serverless work item tracker

Create an Aurora Serverless work item tracker

The following code examples show how to create a web application that tracks work items in an Amazon Aurora Serverless database and uses Amazon Simple Email Service (Amazon SES) to send reports.

.NET

SDK for .NET

Shows how to use the AWS SDK for .NET to create a web application that tracks work items in an Amazon Aurora database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses a front end built with React.js to interact with a RESTful .NET backend.

- Integrate a React web application with AWS services.
- List, add, update, and delete items in an Aurora table.

- Send an email report of filtered work items using Amazon SES.
- Deploy and manage example resources with the included AWS CloudFormation script.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

C++

SDK for C++

Shows how to create a web application that tracks and reports on work items stored in an Amazon Aurora Serverless database.

For complete source code and instructions on how to set up a C++ REST API that queries Amazon Aurora Serverless data and for use by a React application, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Java

SDK for Java 2.x

Shows how to create a web application that tracks and reports on work items stored in an Amazon RDS database.

For complete source code and instructions on how to set up a Spring REST API that queries Amazon Aurora Serverless data and for use by a React application, see the full example on GitHub.

For complete source code and instructions on how to set up and run an example that uses the JDBC API, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

JavaScript

SDK for JavaScript (v3)

Shows how to use the AWS SDK for JavaScript (v3) to create a web application that tracks work items in an Amazon Aurora database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses a front end built with React.js to interact with an Express Node.js backend.

- Integrate a React.js web application with AWS services.
- List, add, and update items in an Aurora table.
- Send an email report of filtered work items by using Amazon SES.
- Deploy and manage example resources with the included AWS CloudFormation script.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Kotlin

SDK for Kotlin

Shows how to create a web application that tracks and reports on work items stored in an Amazon RDS database.

For complete source code and instructions on how to set up a Spring REST API that queries Amazon Aurora Serverless data and for use by a React application, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

PHP

SDK for PHP

Shows how to use the AWS SDK for PHP to create a web application that tracks work items in an Amazon RDS database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses a front end built with React.js to interact with a RESTful PHP backend.

- Integrate a React.js web application with AWS services.
- List, add, update, and delete items in an Amazon RDS table.
- Send an email report of filtered work items using Amazon SES.
- Deploy and manage example resources with the included AWS CloudFormation script.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service

- Amazon SES

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) to create a REST service that tracks work items in an Amazon Aurora Serverless database and emails reports by using Amazon Simple Email Service (Amazon SES). This example uses the Flask web framework to handle HTTP routing and integrates with a React webpage to present a fully functional web application.

- Build a Flask REST service that integrates with AWS services.
- Read, write, and update work items that are stored in an Aurora Serverless database.
- Create an AWS Secrets Manager secret that contains database credentials and use it to authenticate calls to the database.
- Use Amazon SES to send email reports of work items.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Serverless examples for Amazon RDS

The following code examples show how to use Amazon RDS with AWS SDKs.

Examples

- Connecting to an Amazon RDS database in a Lambda function

Connecting to an Amazon RDS database in a Lambda function

The following code examples show how to implement a Lambda function that connects to an RDS database. The function makes a simple database request and returns the result.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using .NET.

```
using System.Data;
using System.Text.Json;
using Amazon.Lambda.APIGatewayEvents;
using Amazon.Lambda.Core;
using MySql.Data.MySqlClient;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace aws_rds;

public class InputModel
{
    public string key1 { get; set; }
    public string key2 { get; set; }
}

public class Function
{
    /// <summary>
    // Handles the Lambda function execution for connecting to RDS using IAM
 authentication.
    /// </summary>
```

```
    /// <param name="input">The input event data passed to the Lambda function</
param>
    /// <param name="context">The Lambda execution context that provides runtime
 information</param>
    /// <returns>A response object containing the execution result</returns>

    public async Task<APIGatewayProxyResponse>
 FunctionHandler(APIGatewayProxyRequest request, ILambdaContext context)
    {
        // Sample Input: {"body": "{\"key1\":\"20\", \"key2\":\"25\"}"}
        var input = JsonSerializer.Deserialize<InputModel>(request.Body);

        /// Obtain authentication token
        var authToken = RDSAuthTokenGenerator.GenerateAuthToken(
            Environment.GetEnvironmentVariable("RDS_ENDPOINT"),
            Convert.ToInt32(Environment.GetEnvironmentVariable("RDS_PORT")),
            Environment.GetEnvironmentVariable("RDS_USERNAME")
        );

        /// Build the Connection String with the Token
        string connectionString =
 $"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT")};" +

 $"Port={Environment.GetEnvironmentVariable("RDS_PORT")};" +

 $"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME")};" +
                                  $"Pwd={authToken};";


        try
        {
            await using var connection = new MySqlConnection(connectionString);
            await connection.OpenAsync();

            const string sql = "SELECT @param1 + @param2 AS Sum";

            await using var command = new MySqlCommand(sql, connection);
            command.Parameters.AddWithValue("@param1", int.Parse(input.key1 ??
 "0"));
            command.Parameters.AddWithValue("@param2", int.Parse(input.key2 ??
 "0"));

            await using var reader = await command.ExecuteReaderAsync();
            if (await reader.ReadAsync())
```

```
            {
                int result = reader.GetInt32("Sum");

                //Sample Response: {"statusCode":200,"body":"{\"message\":\"The
 sum is: 45\"}","isBase64Encoded":false}
                return new APIGatewayProxyResponse
                {
                    StatusCode = 200,
                    Body = JsonSerializer.Serialize(new { message = $"The sum is:
 {result}" })
                };
            }

        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }

        return new APIGatewayProxyResponse
        {
            StatusCode = 500,
            Body = JsonSerializer.Serialize(new { error = "Internal server
 error" })
        };
    }
}
```

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using Go.

```
/*
```

```
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")

 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }

 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
```

```
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }

 defer db.Close()

 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
import software.amazon.awssdk.services.rdsdata.model.Field;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class RdsLambdaHandler implements
 RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
 event, Context context) {
        APIGatewayProxyResponseEvent response = new
 APIGatewayProxyResponseEvent();

        try {
            // Obtain auth token
            String token = createAuthToken();

            // Define connection configuration
```

```
            String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true",
                    System.getenv("ProxyHostName"),
                    System.getenv("Port"),
                    System.getenv("DBName"));

            // Establish a connection to the database
            try (Connection connection =
 DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
 token);
                 PreparedStatement statement =
 connection.prepareStatement("SELECT ? + ? AS sum")) {

                statement.setInt(1, 3);
                statement.setInt(2, 2);

                try (ResultSet resultSet = statement.executeQuery()) {
                    if (resultSet.next()) {
                        int sum = resultSet.getInt("sum");
                        response.setStatusCode(200);
                        response.setBody("The selected sum is: " + sum);
                    }
                }
            }

        } catch (Exception e) {
            response.setStatusCode(500);
            response.setBody("Error: " + e.getMessage());
        }

        return response;
    }

    private String createAuthToken() {
        // Create RDS Data Service client
        RdsDataClient rdsDataClient = RdsDataClient.builder()
                .region(Region.of(System.getenv("AWS_REGION")))
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();

        // Define authentication request
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
                .resourceArn(System.getenv("ProxyHostName"))
                .secretArn(System.getenv("DBUserName"))
```

```
                .database(System.getenv("DBName"))
                .sql("SELECT 'RDS IAM Authentication'")
                .build();

        // Execute request and obtain authentication token
        ExecuteStatementResponse response =
 rdsDataClient.executeStatement(request);
        Field tokenField = response.records().get(0).get(0);

        return tokenField.stringValue();
    }
}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
```

```
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

Connecting to an Amazon RDS database in a Lambda function using TypeScript.

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that
 the DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
```

```
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
 body: string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

PHP

SDK for PHP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
```

```
# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }


    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new
 AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }

    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();
```

```
        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];

        // Create the connection to the DB
        $conn = new PDO(

 "mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
            $connectionConfig['user'],
            $connectionConfig['password'],
            [
                PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
                PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
            ]
        );

        // Obtain the result of the query
        $stmt = $conn->prepare('SELECT ?+? AS sum');
        $stmt->execute([3, 2]);

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();

        return [
            'sum' => $result['sum']
        ];
    }
}
```

```
$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using Python.

```
import json
import os
import boto3
import pymysql

# RDS settings
proxy_host_name = os.environ['PROXY_HOST_NAME']
port = int(os.environ['PORT'])
db_name = os.environ['DB_NAME']
db_user_name = os.environ['DB_USER_NAME']
aws_region = os.environ['AWS_REGION']


# Fetch RDS Auth Token
def get_auth_token():
    client = boto3.client('rds')
    token = client.generate_db_auth_token(
        DBHostname=proxy_host_name,
        Port=port
        DBUsername=db_user_name
        Region=aws_region
    )
    return token

def lambda_handler(event, context):
    token = get_auth_token()
```

```
    try:
        connection = pymysql.connect(
            host=proxy_host_name,
            user=db_user_name,
            password=token,
            db=db_name,
            port=port,
            ssl={'ca': 'Amazon RDS'}  # Ensure you have the CA bundle for SSL
connection
        )

        with connection.cursor() as cursor:
            cursor.execute('SELECT %s + %s AS sum', (3, 2))
            result = cursor.fetchone()

        return result

    except Exception as e:
        return (f"Error: {str(e)}")  # Return an error message if an exception
occurs
```

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using Ruby.

```
# Ruby code here.

require 'aws-sdk-rds'
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
  endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
```

```
port = ENV['Port']           # 3306
user = ENV['DBUser']
region = ENV['DBRegion']     # 'us-east-1'
db_name = ENV['DBName']

credentials = Aws::Credentials.new(
  ENV['AWS_ACCESS_KEY_ID'],
  ENV['AWS_SECRET_ACCESS_KEY'],
  ENV['AWS_SESSION_TOKEN']
)
rds_client = Aws::RDS::AuthTokenGenerator.new(
  region: region,
  credentials: credentials
)

token = rds_client.auth_token(
  endpoint: endpoint+ ':' + port,
  user_name: user,
  region: region
)

begin
  conn = Mysql2::Client.new(
    host: endpoint,
    username: user,
    password: token,
    port: port,
    database: db_name,
    sslca: '/var/task/global-bundle.pem',
    sslverify: true,
    enable_cleartext_plugin: true
  )
  a = 3
  b = 2
  result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
  puts result
  conn.close
  {
    statusCode: 200,
    body: result.to_json
  }
rescue => e
  puts "Database connection failed due to #{e}"
end
```

```
end
```

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
```

```
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
 aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
 SignableBody::Bytes(&[]))
            .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}
```

```
async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

    let pool = sqlx::postgres::PgPoolOptions::new()
        .connect_with(opts)
        .await?;

    let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
        .bind(3)
        .bind(2)
        .fetch_one(&pool)
        .await?;

    println!("Result: {:?}", result);

    Ok(json!({
        "statusCode": 200,
        "content-type": "text/plain",
        "body": format!("The selected sum is: {result}")
    }))
}
```

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Security in Amazon RDS

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon RDS , see AWS services in scope by compliance program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon RDS . The following topics show you how to configure Amazon RDS to meet your security and compliance objectives. You also learn how to use other AWS services that help you monitor and secure your Amazon RDS resources.

You can manage access to your Amazon RDS resources and your databases on a DB instance . The method you use to manage access depends on what type of task the user needs to perform with Amazon RDS :

- Run your DB instance in a virtual private cloud (VPC) based on the Amazon VPC service for the greatest possible network access control. For more information about creating a DB instance in a VPC, see Amazon VPC and Amazon RDS .
- Use AWS Identity and Access Management (IAM) policies to assign permissions that determine who is allowed to manage Amazon RDS resources. For example, you can use IAM to determine who is allowed to create, describe, modify, and delete DB instances , tag resources, or modify security groups.

- Use security groups to control what IP addresses or Amazon EC2 instances can connect to your databases on a DB instance . When you first create a DB instance , its firewall prevents any database access except through rules specified by an associated security group.
- Use Secure Socket Layer (SSL) or Transport Layer Security (TLS) connections with DB instances running the Db2, MySQL, MariaDB, PostgreSQL, Oracle, or Microsoft SQL Server database engines. For more information on using SSL/TLS with a DB instance , see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
- Use Amazon RDS encryption to secure your DB instances and snapshots at rest. Amazon RDS encryption uses the industry standard AES-256 encryption algorithm to encrypt your data on the server that hosts your DB instance . For more information, see Encrypting Amazon RDS resources .
- Use network encryption and transparent data encryption with Oracle DB instances; for more information, see Oracle native network encryption and Oracle Transparent Data Encryption
- Use the security features of your DB engine to control who can log in to the databases on a DB instance . These features work just as if the database was on your local network.

> **Note**
>
> You have to configure security only for your use cases. You don't have to configure security access for processes that Amazon RDS manages. These include creating backups, replicating data between a primary DB instance and a read replica, and other processes.

For more information on managing access to Amazon RDS resources and your databases on a DB instance , see the following topics.

Topics

- Database authentication with Amazon RDS
- Password management with Amazon RDS and AWS Secrets Manager
- Data protection in Amazon RDS
- Identity and access management for Amazon RDS
- Logging and monitoring in Amazon RDS
- Compliance validation for Amazon RDS
- Resilience in Amazon RDS

- Infrastructure security in Amazon RDS
- Amazon RDS API and interface VPC endpoints (AWS PrivateLink)
- Security best practices for Amazon RDS
- Controlling access with security groups
- Master user account privileges
- Using service-linked roles for Amazon RDS
- Amazon VPC and Amazon RDS

Database authentication with Amazon RDS

Amazon RDS supports several ways to authenticate database users.

Password, Kerberos, and IAM database authentication use different methods of authenticating to the database. Therefore, a specific user can log in to a database using only one authentication method.

For PostgreSQL, use only one of the following role settings for a user of a specific database:

- To use IAM database authentication, assign the `rds_iam` role to the user.
- To use Kerberos authentication, assign the `rds_ad` role to the user.
- To use password authentication, don't assign either the `rds_iam` or `rds_ad` roles to the user.

Don't assign both the `rds_iam` and `rds_ad` roles to a user of a PostgreSQL database either directly or indirectly by nested grant access. If the `rds_iam` role is added to the master user, IAM authentication takes precedence over password authentication so the master user has to log in as an IAM user.

> **Important**
>
> We strongly recommend that you do not use the master user directly in your applications. Instead, adhere to the best practice of using a database user created with the minimal privileges required for your application.

Topics

- Password authentication

- IAM database authentication
- Kerberos authentication

Password authentication

With *password authentication,* your database performs all administration of user accounts. You create users with SQL statements such as `CREATE USER`, with the appropriate clause required by the DB engine for specifying passwords. For example, in MySQL the statement is `CREATE USER` *`name`* `IDENTIFIED BY` *`password`*, while in PostgreSQL, the statement is `CREATE USER` *`name`* `WITH PASSWORD` *`password`*.

With password authentication, your database controls and authenticates user accounts. If a DB engine has strong password management features, they can enhance security. Database authentication might be easier to administer using password authentication when you have small user communities. Because clear text passwords are generated in this case, integrating with AWS Secrets Manager can enhance security.

For information about using Secrets Manager with Amazon RDS , see Creating a basic secret and Rotating secrets for supported Amazon RDS databases in the *AWS Secrets Manager User Guide*. For information about programmatically retrieving your secrets in your custom applications, see Retrieving the secret value in the *AWS Secrets Manager User Guide*.

IAM database authentication

You can authenticate to your DB instance using AWS Identity and Access Management (IAM) database authentication. With this authentication method, you don't need to use a password when you connect to a DB instance . Instead, you use an authentication token.

For more information about IAM database authentication, including information about availability for specific DB engines, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL .

Kerberos authentication

Amazon RDS supports external authentication of database users using Kerberos and Microsoft Active Directory. Kerberos is a network authentication protocol that uses tickets and symmetric-key cryptography to eliminate the need to transmit passwords over the network. Kerberos has been built into Active Directory and is designed to authenticate users to network resources, such as databases.

Amazon RDS support for Kerberos and Active Directory provides the benefits of single sign-on and centralized authentication of database users. You can keep your user credentials in Active Directory. Active Directory provides a centralized place for storing and managing credentials for multiple DB instances .

To use credentials from your self-managed Active Directory, you need to setup a trust relationship to the Directory Service for Microsoft Active Directory that the DB instance is joined to.

RDS for PostgreSQL and RDS for MySQL support one-way and two-way forest trust relationships with forest-wide authentication or selective authentication.

In some scenarios, you can configure Kerberos authentication over an external trust relationship. This requires your self-managed Active Directory to have additional settings. This includes but is not limited to Kerberos Forest Search Order.

Microsoft SQL Server and PostgreSQL DB instances support one-way and two-way forest trust relationships. Oracle DB instances support one-way and two-way external and forest trust relationships. For more information, see When to create a trust relationship in the *Directory Service Administration Guide*.

For information about Kerberos authentication with a specific DB engine, see the following:

- Working with AWS Managed Active Directory with RDS for SQL Server
- Using Kerberos authentication for Amazon RDS for MySQL
- Configuring Kerberos authentication for Amazon RDS for Oracle
- Using Kerberos authentication with Amazon RDS for PostgreSQL
- Using Kerberos authentication for Amazon RDS for Db2 .

> **Note**
>
> Currently, Kerberos authentication isn't supported for MariaDB DB instances.

Password management with Amazon RDS and AWS Secrets Manager

Amazon RDS integrates with Secrets Manager to manage master user passwords for your DB instances and Multi-AZ DB clusters.

Topics

- Limitations for Secrets Manager integration with Amazon RDS
- Overview of managing master user passwords with AWS Secrets Manager
- Benefits of managing master user passwords with Secrets Manager
- Permissions required for Secrets Manager integration
- Enforcing RDS management of the master user password in AWS Secrets Manager
- Managing the master user password for a DB instance with Secrets Manager
- Managing the master user password for an RDS for Oracle tenant database with Secrets Manager
- Managing the master user password for a Multi-AZDB cluster with Secrets Manager
- Rotating the master user password secret for a DB instance
- Rotating the master user password secret for a Multi-AZDB cluster
- Viewing the details about a secret for a DB instance
- Viewing the details about a secret for a Multi-AZDB cluster
- Viewing the details about a secret for a tenant database
- Region and version availability

Limitations for Secrets Manager integration with Amazon RDS

Managing master user passwords with Secrets Manager isn't supported for the following features:

- Creating a read replica when the source DB or DB cluster manages credentials with Secrets Manager. This applies to all DB engines except RDS for SQL Server.
- Amazon RDS Blue/Green Deployments
- Amazon RDS Custom
- Oracle Data Guard switchover

Overview of managing master user passwords with AWS Secrets Manager

With AWS Secrets Manager, you can replace hard-coded credentials in your code, including database passwords, with an API call to Secrets Manager to retrieve the secret programmatically. For more information about Secrets Manager, see the AWS Secrets Manager User Guide.

When you store database secrets in Secrets Manager, your AWS account incurs charges. For information about pricing, see AWS Secrets Manager Pricing.

You can specify that RDS manages the master user password in Secrets Manager for an Amazon RDS DB instance or Multi-AZ DB cluster when you perform one of the following operations:

- Create a DB instance
- Create a Multi-AZ DB cluster
- Create a tenant database in an RDS for Oracle CDB
- Modify a DB instance
- Modify a Multi-AZ DB cluster
- Modify a tenant database (RDS for Oracle only)
- Restore a DB instance from Amazon S3
- Restore a DB instance from a snapshot or to a point in time (RDS for Oracle only)

When you specify that RDS manages the master user password in Secrets Manager, RDS generates the password and stores it in Secrets Manager. You can interact directly with the secret to retrieve the credentials for the master user. You can also specify a customer managed key to encrypt the secret, or use the KMS key that is provided by Secrets Manager.

RDS manages the settings for the secret and rotates the secret every seven days by default. You can modify some of the settings, such as the rotation schedule. If you delete a DB instance that manages a secret in Secrets Manager, the secret and its associated metadata are also deleted.

To connect to a DB instance or Multi-AZ DB cluster with the credentials in a secret, you can retrieve the secret from Secrets Manager. For more information, see Retrieve secrets from AWS Secrets Manager and Connect to a SQL database with credentials in an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

Benefits of managing master user passwords with Secrets Manager

Managing RDS master user passwords with Secrets Manager provides the following benefits:

- RDS automatically generates database credentials.
- RDS automatically stores and manages database credentials in AWS Secrets Manager.
- RDS rotates database credentials regularly, without requiring application changes.
- Secrets Manager secures database credentials from human access and plain text view.
- Secrets Manager allows retrieval of database credentials in secrets for database connections.
- Secrets Manager allows fine-grained control of access to database credentials in secrets using IAM.
- You can optionally separate database encryption from credentials encryption with different KMS keys.
- You can eliminate manual management and rotation of database credentials.
- You can monitor database credentials easily with AWS CloudTrail and Amazon CloudWatch.

For more information about the benefits of Secrets Manager, see the AWS Secrets Manager User Guide.

Permissions required for Secrets Manager integration

Users must have the required permissions to perform operations related to Secrets Manager integration. You can create IAM policies that grant permissions to perform specific API operations on the specified resources they need. You can then attach those policies to the IAM permission sets or roles that require those permissions. For more information, see Identity and access management for Amazon RDS.

For create, modify, or restore operations, the user who specifies that Amazon RDS manages the master user password in Secrets Manager must have permissions to perform the following operations:

- `kms:DescribeKey`
- `secretsmanager:CreateSecret`
- `secretsmanager:TagResource`

The `kms:DescribeKey` permission is required to access your customer-managed key for the `MasterUserSecretKmsKeyId` and to describe `aws/secretsmanager`.

For create, modify, or restore operations, the user who specifies the customer managed key to encrypt the secret in Secrets Manager must have permissions to perform the following operations:

- `kms:Decrypt`
- `kms:GenerateDataKey`
- `kms:CreateGrant`

For modify operations, the user who rotates the master user password in Secrets Manager must have permissions to perform the following operation:

- `secretsmanager:RotateSecret`

Enforcing RDS management of the master user password in AWS Secrets Manager

You can use IAM condition keys to enforce RDS management of the master user password in AWS Secrets Manager. The following policy doesn't allow users to create or restore DB instances or DB clusters or create or modify tenant databases unless the master user password is managed by RDS in Secrets Manager.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["rds:CreateDBInstance", "rds:CreateDBCluster",
 "rds:RestoreDBInstanceFromS3", "rds:RestoreDBClusterFromS3",
                       "rds:RestoreDBInstanceFromDBSnapshot",
 "rds:RestoreDBInstanceToPointInTime", "rds:CreateTenantDatabase",
                       "rds:ModifyTenantDatabase"],
            "Resource": "*",
            "Condition": {
                "Bool": {
```

```
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

> **Note**
>
> This policy enforces password management in AWS Secrets Manager at creation. However, you can still disable Secrets Manager integration and manually set a master password by modifying the instance.
> To prevent this, include `rds:ModifyDBInstance`, `rds:ModifyDBCluster` in the action block of the policy. Be aware, this prevents the user from applying any further modifications to existing instances that don't have Secrets Manager integration enabled.

For more information about using condition keys in IAM policies, see Policy condition keys for Amazon RDS and Example policies: Using condition keys.

Managing the master user password for a DB instance with Secrets Manager

You can configure RDS management of the master user password in Secrets Manager when you perform the following actions:

- Creating an Amazon RDS DB instance
- Modifying an Amazon RDS DB instance
- Restoring a backup into an Amazon RDS for MySQL DB instance
- Restoring to a DB instance (RDS for Oracle only)
- Restoring a DB instance to a specified time for Amazon RDS (RDS for Oracle only)

You can perform the preceding operations using the RDS console, the AWS CLI, or the RDS API.

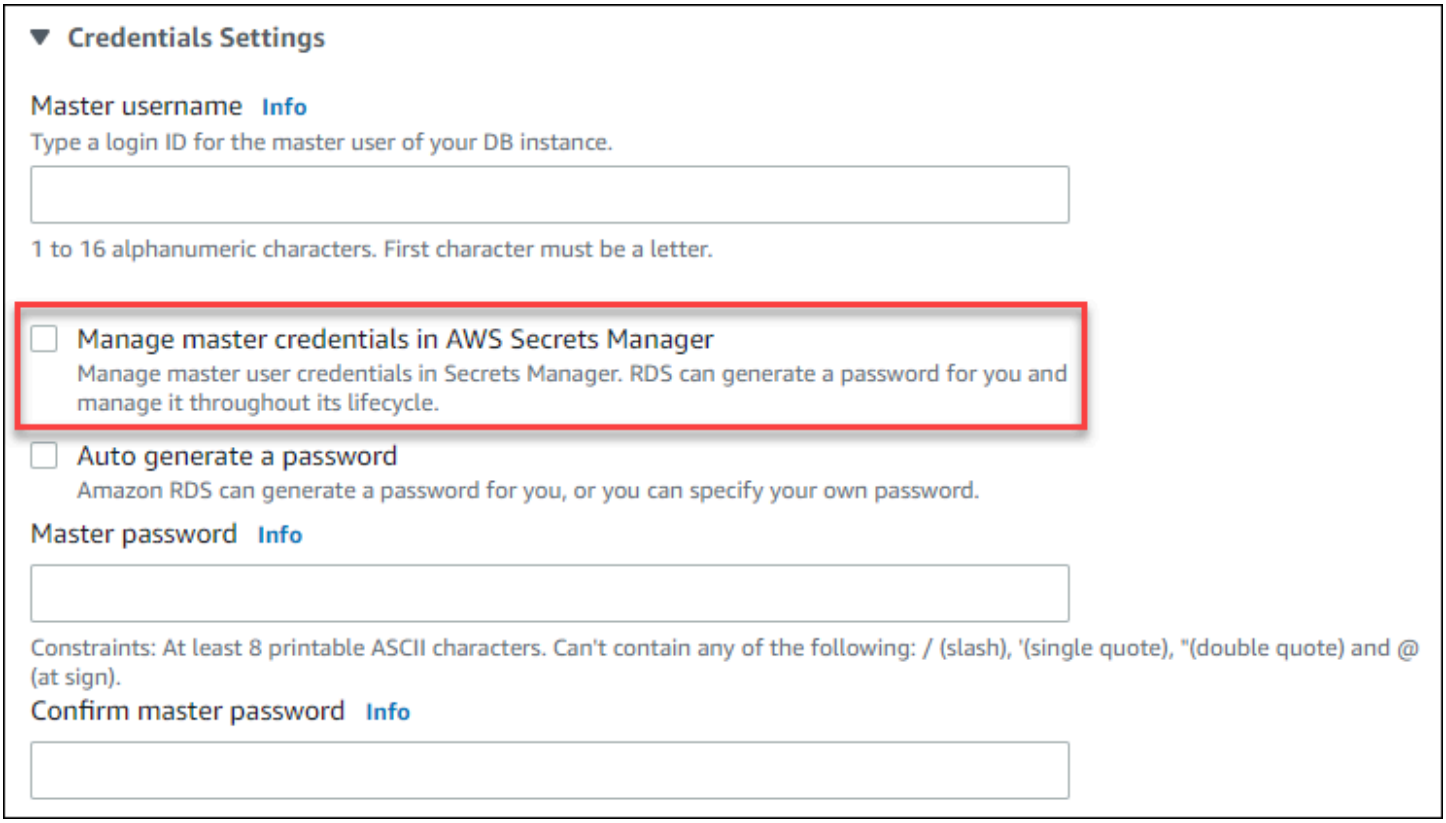
Console

Follow the instructions for creating or modifying a DB instance with the RDS console:

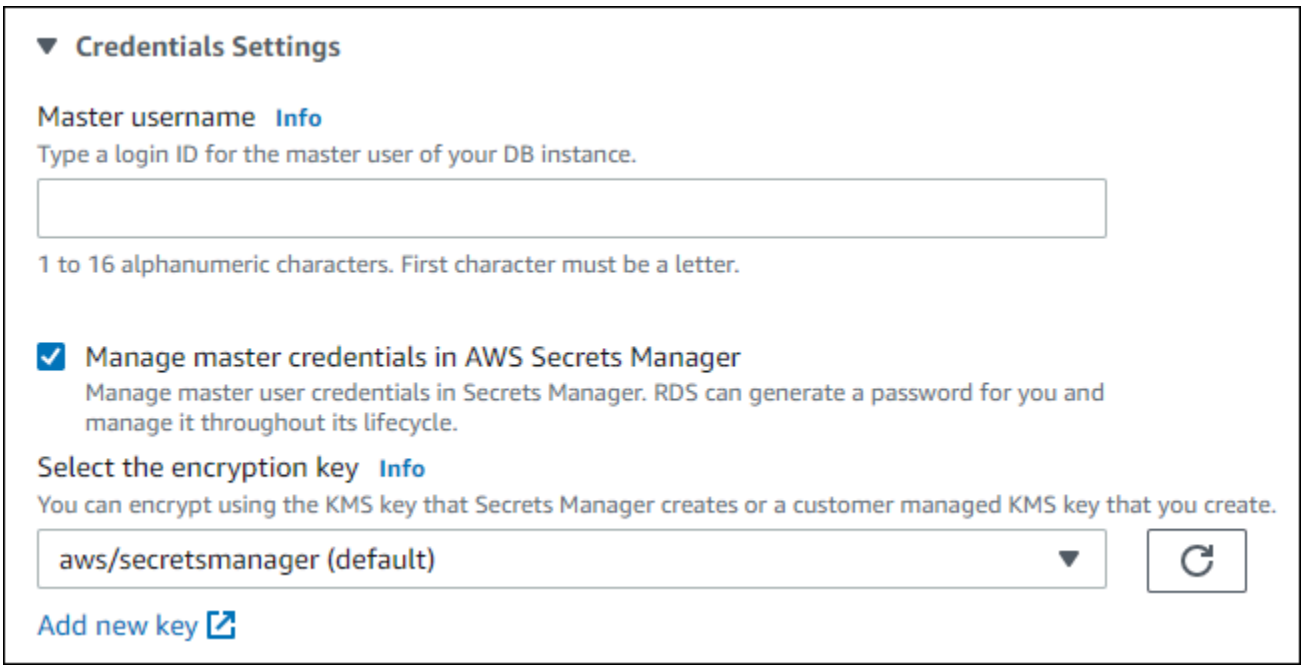
- Creating a DB instance
- Modifying an Amazon RDS DB instance
- Importing data from Amazon S3 to a new MySQL DB instance

When you use the RDS console to perform one of these operations, you can specify that the master user password is managed by RDS in Secrets Manager. When you're creating or restoring a DB instance, select **Manage master credentials in AWS Secrets Manager** in **Credential settings**. When you're modifying a DB instance, select **Manage master credentials in AWS Secrets Manager** in **Settings**.

The following image is an example of the **Manage master credentials in AWS Secrets Manager** setting when you are creating or restoring a DB instance.



When you select this option, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.



You can choose to encrypt the secret with a KMS key that Secrets Manager provides or with a customer managed key that you create. After RDS is managing the database credentials for a DB instance, you can't change the KMS key used to encrypt the secret.

You can choose other settings to meet your requirements. For more information about the available settings when you're creating a DB instance, see Settings for DB instances. For more information about the available settings when you're modifying a DB instance, see Settings for DB instances.

AWS CLI

To manage the master user password with RDS in Secrets Manager, specify the `--manage-master-user-password` option in one of the following AWS CLI commands:

- create-db-instance
- modify-db-instance
- restore-db-instance-from-s3
- restore-db-instance-from-db-snapshot (RDS for Oracle only)
- restore-db-instance-to-point-in-time (RDS for Oracle only)

When you specify the `--manage-master-user-password` option in these commands, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.

To encrypt the secret, you can specify a customer managed key or use the default KMS key that is provided by Secrets Manager. Use the `--master-user-secret-kms-key-id` option to specify a customer managed key. The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key. To use a KMS key in a different AWS account, specify the key ARN or alias ARN. After RDS is managing the database credentials for a DB instance, you can't change the KMS key that is used to encrypt the secret.

You can choose other settings to meet your requirements. For more information about the available settings when you are creating a DB instance, see Settings for DB instances. For more information about the available settings when you are modifying a DB instance, see Settings for DB instances.

The following example creates a DB instance and specifies that RDS manages the master user password in Secrets Manager. The secret is encrypted using the KMS key that is provided by Secrets Manager.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --engine mysql \
    --engine-version 8.0.39 \
    --db-instance-class db.r5b.large \
    --allocated-storage 200 \
    --master-username testUser \
    --manage-master-user-password
```

For Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier mydbinstance ^
    --engine mysql ^
    --engine-version 8.0.39 ^
    --db-instance-class db.r5b.large ^
    --allocated-storage 200 ^
    --master-username testUser ^
    --manage-master-user-password
```

RDS API

To specify that RDS manages the master user password in Secrets Manager, set the `ManageMasterUserPassword` parameter to `true` in one of the following RDS API operations:

- CreateDBInstance
- ModifyDBInstance
- RestoreDBInstanceFromS3
- RestoreDBInstanceFromSnapshot (RDS for Oracle only)
- RestoreDBInstanceToPointInTime (RDS for Oracle only)

When you set the `ManageMasterUserPassword` parameter to `true` in one of these operations, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.

To encrypt the secret, you can specify a customer managed key or use the default KMS key that is provided by Secrets Manager. Use the `MasterUserSecretKmsKeyId` parameter to specify a customer managed key. The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key. To use a KMS key in a different AWS account, specify the key ARN or alias ARN. After RDS is managing the database credentials for a DB instance, you can't change the KMS key that is used to encrypt the secret.

Managing the master user password for an RDS for Oracle tenant database with Secrets Manager

You can configure RDS management of the master user password in Secrets Manager when you perform the following actions:

- Adding an RDS for Oracle tenant database to your CDB instance
- Modifying an RDS for Oracle tenant database

You can use the RDS console, the AWS CLI, or the RDS API to perform the preceding actions.

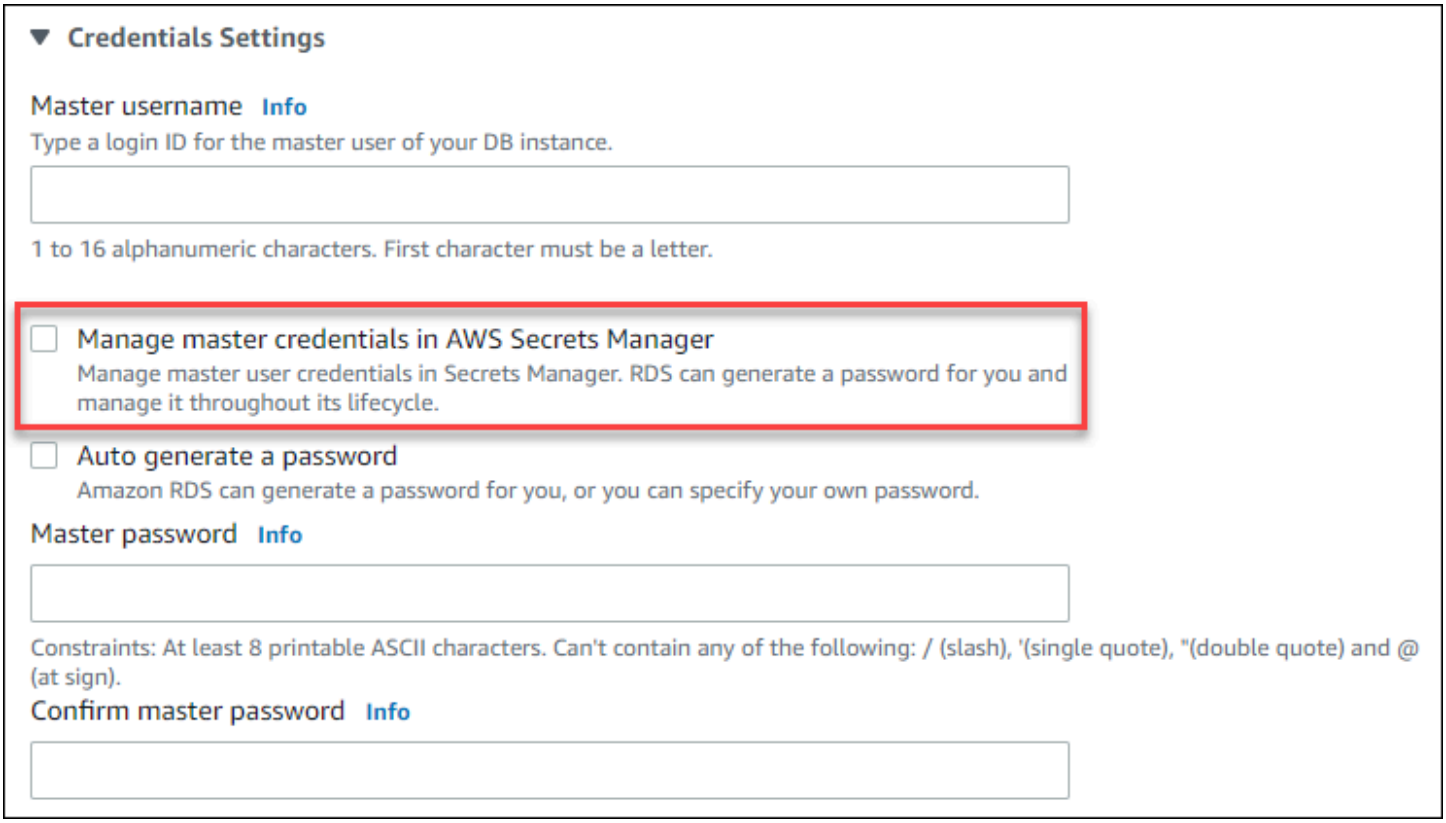
Console

Follow the instructions for creating or modifying an RDS for Oracle tenant database with the RDS console:

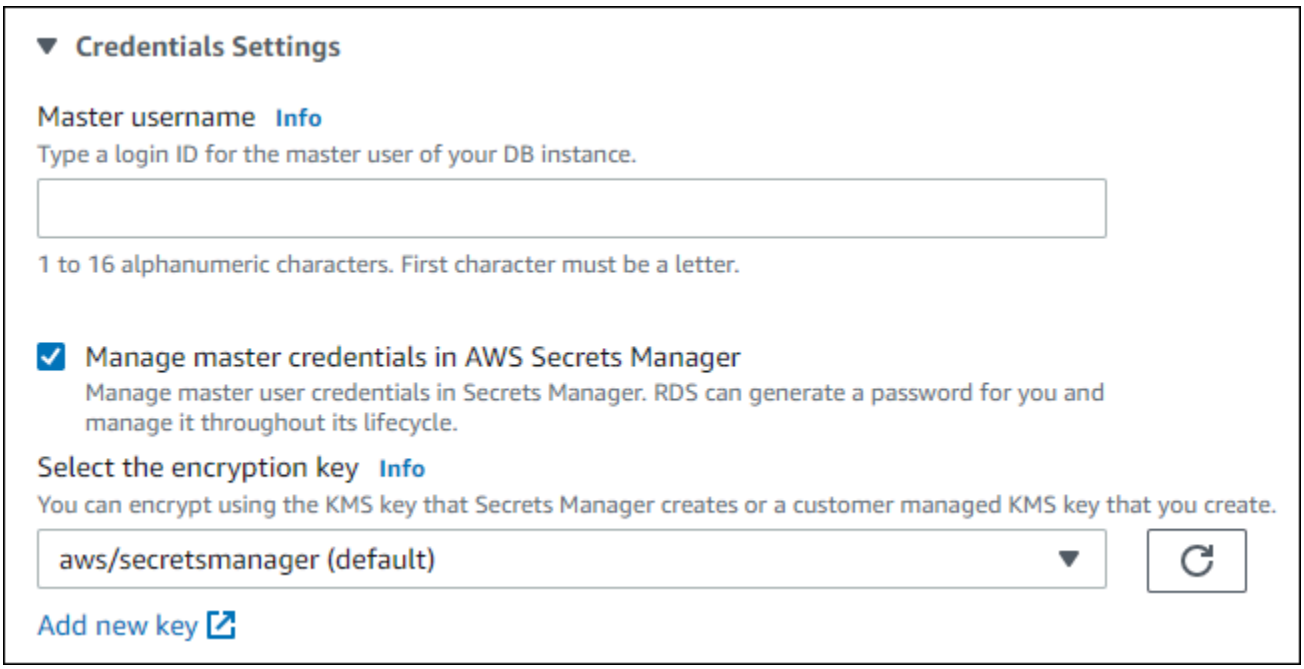
- Adding an RDS for Oracle tenant database to your CDB instance
- Modifying an RDS for Oracle tenant database

When you use the RDS console to perform one of the preceding operations, you can specify that RDS manage the master password in Secrets Manager. When you create a tenant database, select **Manage master credentials in AWS Secrets Manager** in **Credential settings**. When you modify a tenant database, select **Manage master credentials in AWS Secrets Manager** in **Settings**.

The following image is an example of the **Manage master credentials in AWS Secrets Manager** setting when you are creating a tenant database.



When you select this option, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.



You can choose to encrypt the secret with a KMS key that Secrets Manager provides or with a customer managed key that you create. After RDS is managing the database credentials for a tenant database, you can't change the KMS key that is used to encrypt the secret.

You can choose other settings to meet your requirements. For more information about the available settings when you are creating a tenant database, see Settings for DB instances. For more information about the available settings when you are modifying a tenant database, see Settings for DB instances.

AWS CLI

To manage the master user password with RDS in Secrets Manager, specify the `--manage-master-user-password` option in one of the following AWS CLI commands:

- create-tenant-database
- modify-tenant-database

When you specify the `--manage-master-user-password` option in the preceding commands, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.

To encrypt the secret, you can specify a customer managed key or use the default KMS key that is provided by Secrets Manager. Use the `--master-user-secret-kms-key-id` option to specify a

customer managed key. The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key. To use a KMS key in a different AWS account, specify the key ARN or alias ARN. After RDS is managing the database credentials for a tenant database, you can't change the KMS key that is used to encrypt the secret.

You can choose other settings to meet your requirements. For more information about the available settings when you are creating a tenant database, see create-tenant-database. For more information about the available settings when you are modifying a tenant database, see modify-tenant-database.

The following example creates an RDS for Oracle tenant database and specifies that RDS manages the master user password in Secrets Manager. The secret is encrypted using the KMS key that is provided by Secrets Manager.

Example

For Linux, macOS, or Unix:

```
aws rds create-tenant-database --region us-east-1 \
    --db-instance-identifier my-cdb-inst \
    --tenant-db-name mypdb2 \
    --master-username mypdb2-admin \
    --character-set-name UTF-16 \
    --manage-master-user-password
```

For Windows:

```
aws rds create-tenant-database --region us-east-1 ^
    --db-instance-identifier my-cdb-inst ^
    --tenant-db-name mypdb2 ^
    --master-username mypdb2-admin ^
    --character-set-name UTF-16 ^
    --manage-master-user-password
```

RDS API

To specify that RDS manages the master user password in Secrets Manager, set the `ManageMasterUserPassword` parameter to `true` in one of the following RDS API operations:

- CreateTenantDatabase
- ModifyTenantDatabase

When you set the `ManageMasterUserPassword` parameter to `true` in one of these operations, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.

To encrypt the secret, you can specify a customer managed key or use the default KMS key that is provided by Secrets Manager. Use the `MasterUserSecretKmsKeyId` parameter to specify a customer managed key. The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key. To use a KMS key in a different AWS account, specify the key ARN or alias ARN. After RDS is managing the database credentials for a tenant database, you can't change the KMS key that is used to encrypt the secret.

Managing the master user password for a Multi-AZDB cluster with Secrets Manager

You can configure RDS management of the master user password in Secrets Manager when you perform the following actions:

- Creating a Multi-AZ DB cluster for Amazon RDS
- Modifying a Multi-AZ DB cluster for Amazon RDS

You can use the RDS console, the AWS CLI, or the RDS API to perform these actions.

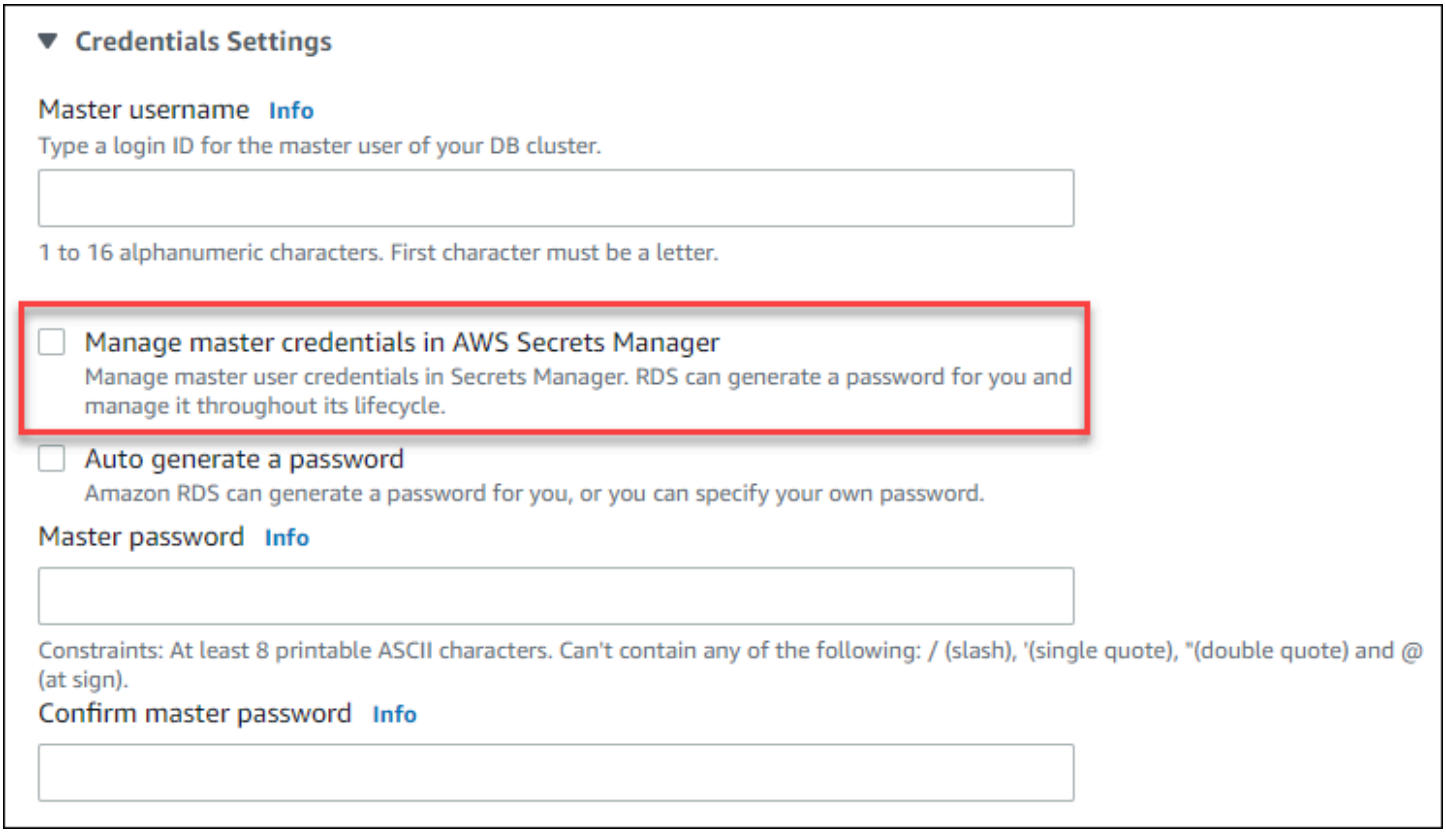
Console

Follow the instructions for creating or modifying a Multi-AZ DB cluster with the RDS console:

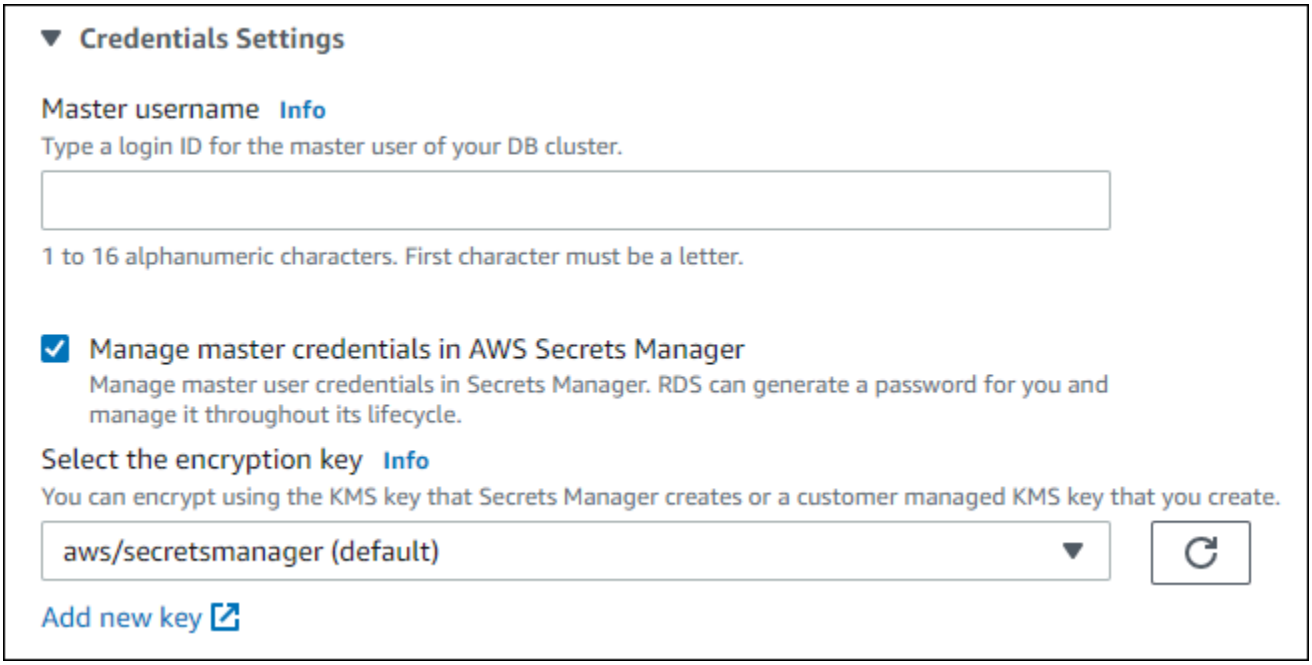
- Creating a DB cluster
- Modifying a Multi-AZ DB cluster for Amazon RDS

When you use the RDS console to perform one of these operations, you can specify that the master user password is managed by RDS in Secrets Manager. To do so when you are creating a DB cluster, select **Manage master credentials in AWS Secrets Manager** in **Credential settings**. When you are modifying a DB cluster, select **Manage master credentials in AWS Secrets Manager** in **Settings**.

The following image is an example of the **Manage master credentials in AWS Secrets Manager** setting when you are creating a DB cluster.



When you select this option, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.



You can choose to encrypt the secret with a KMS key that Secrets Manager provides or with a customer managed key that you create. After RDS is managing the database credentials for a DB cluster, you can't change the KMS key that is used to encrypt the secret.

You can choose other settings to meet your requirements.

For more information about the available settings when you are creating a Multi-AZ DB cluster, see Settings for creating Multi-AZ DB clusters. For more information about the available settings when you are modifying a Multi-AZ DB cluster, see Settings for modifying Multi-AZ DB clusters.

AWS CLI

To specify that RDS manages the master user password in Secrets Manager, specify the `--manage-master-user-password` option in one of the following commands:

- create-db-cluster
- modify-db-cluster

When you specify the `--manage-master-user-password` option in these commands, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.

To encrypt the secret, you can specify a customer managed key or use the default KMS key that is provided by Secrets Manager. Use the `--master-user-secret-kms-key-id` option to specify a customer managed key. The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key. To use a KMS key in a different AWS account, specify the key ARN or alias ARN. After RDS is managing the database credentials for a DB cluster, you can't change the KMS key that is used to encrypt the secret.

You can choose other settings to meet your requirements.

For more information about the available settings when you are creating a Multi-AZ DB cluster, see Settings for creating Multi-AZ DB clusters. For more information about the available settings when you are modifying a Multi-AZ DB cluster, see Settings for modifying Multi-AZ DB clusters.

This example creates a Multi-AZ DB cluster and specifies that RDS manages the password in Secrets Manager. The secret is encrypted using the KMS key that is provided by Secrets Manager.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
   --db-cluster-identifier mysql-multi-az-db-cluster \
   --engine mysql \
   --engine-version 8.0.39  \
   --backup-retention-period 1  \
   --allocated-storage 4000 \
   --storage-type io1 \
   --iops 10000 \
   --db-cluster-instance-class db.r6gd.xlarge \
   --master-username testUser \
   --manage-master-user-password
```

For Windows:

```
aws rds create-db-cluster ^
   --db-cluster-identifier mysql-multi-az-db-cluster ^
   --engine mysql ^
   --engine-version 8.0.39 ^
   --backup-retention-period 1 ^
   --allocated-storage 4000 ^
   --storage-type io1 ^
   --iops 10000 ^
   --db-cluster-instance-class db.r6gd.xlarge ^
   --master-username testUser ^
   --manage-master-user-password
```

RDS API

To specify that RDS manages the master user password in Secrets Manager, set the `ManageMasterUserPassword` parameter to `true` in one of the following operations:

- CreateDBCluster
- ModifyDBCluster

When you set the `ManageMasterUserPassword` parameter to `true` in one of these operations, RDS generates the master user password and manages it throughout its lifecycle in Secrets Manager.

To encrypt the secret, you can specify a customer managed key or use the default KMS key that is provided by Secrets Manager. Use the `MasterUserSecretKmsKeyId` parameter to specify a

customer managed key. The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the KMS key. To use a KMS key in a different AWS account, specify the key ARN or alias ARN. After RDS is managing the database credentials for a DB cluster, you can't change the KMS key that is used to encrypt the secret.

Rotating the master user password secret for a DB instance

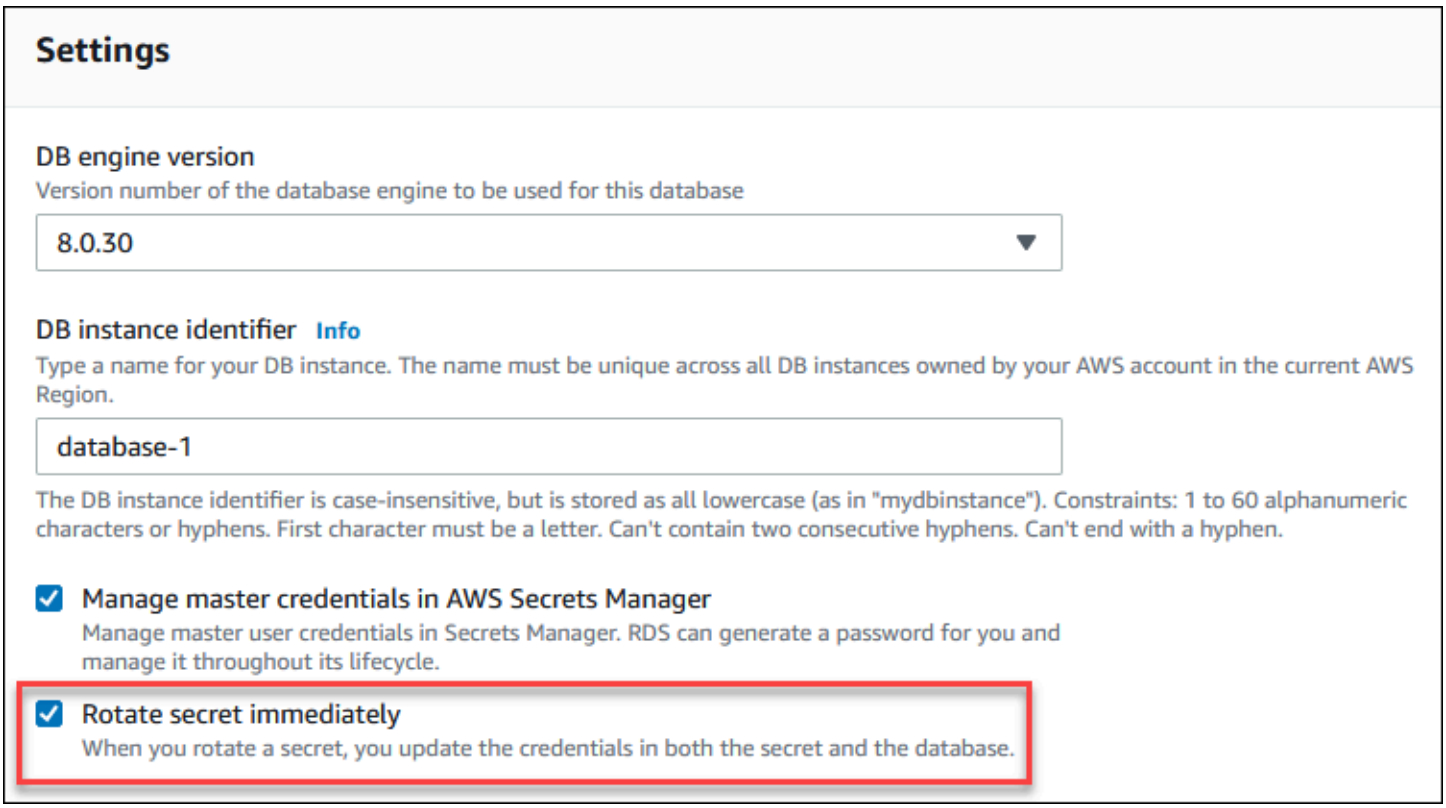
When RDS rotates a master user password secret, Secrets Manager generates a new secret version for the existing secret. The new version of the secret contains the new master user password. Amazon RDS changes the master user password for the DB instance to match the password for the new secret version.

You can rotate a secret immediately instead of waiting for a scheduled rotation. To rotate a master user password secret in Secrets Manager, modify the DB instance. For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

You can rotate a master user password secret immediately with the RDS console, the AWS CLI, or the RDS API. The new password is always 28 characters long and contains at least one upper and lowercase character, one number, and one punctuation.

Console

To rotate a master user password secret using the RDS console, modify the DB instance and select **Rotate secret immediately** in **Settings**.



Follow the instructions for modifying a DB instance with the RDS console in Modifying an Amazon RDS DB instance. You must choose **Apply immediately** on the confirmation page.

AWS CLI

To rotate a master user password secret using the AWS CLI, use the modify-db-instance command and specify the `--rotate-master-user-password` option. You must specify the `--apply-immediately` option when you rotate the master password.

This example rotates a master user password secret.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --rotate-master-user-password \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --rotate-master-user-password ^
    --apply-immediately
```

RDS API

You can rotate a master user password secret using the ModifyDBInstance operation and setting the `RotateMasterUserPassword` parameter to `true`. You must set the `ApplyImmediately` parameter to `true` when you rotate the master password.

Rotating the master user password secret for a Multi-AZDB cluster

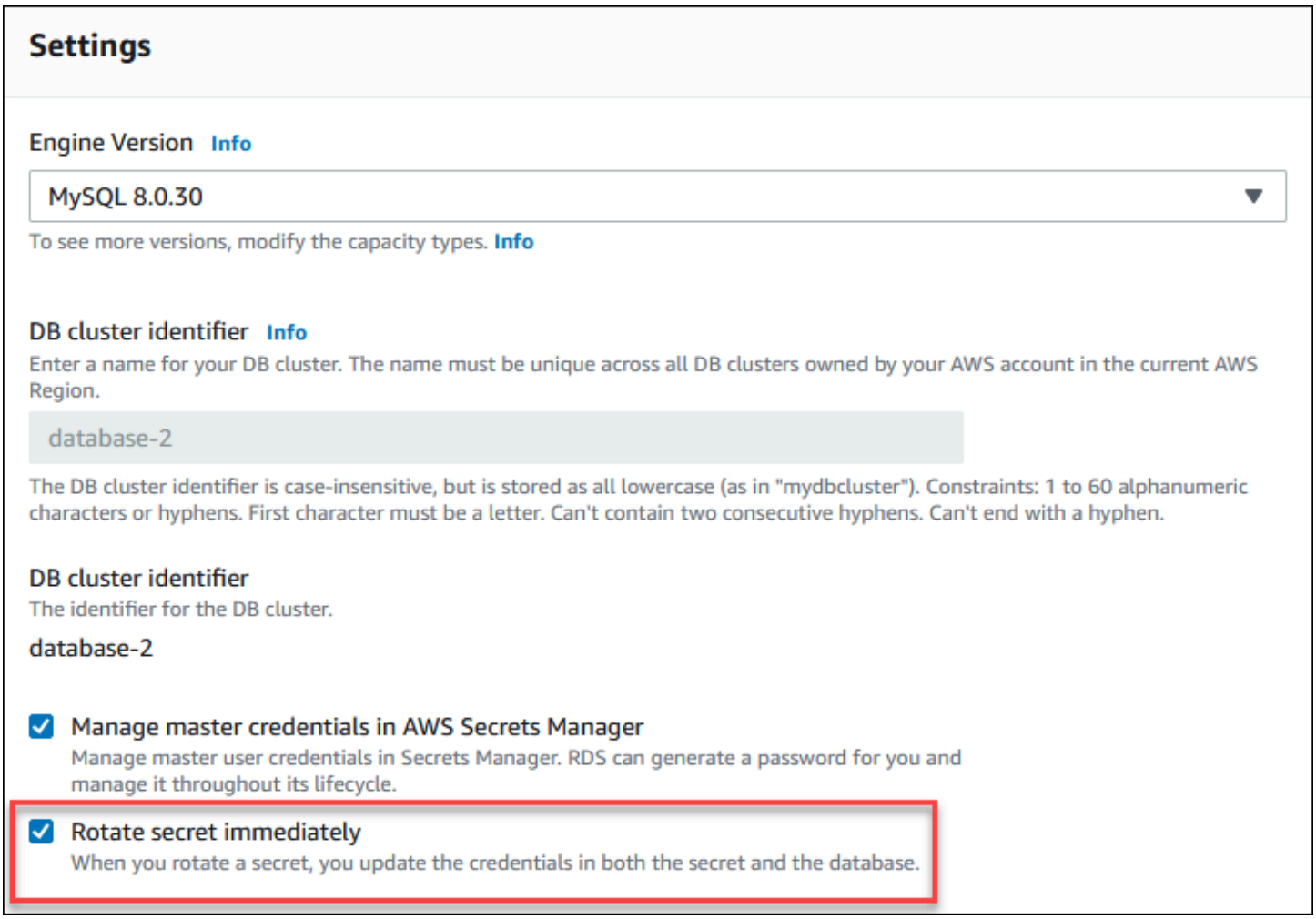
When RDS rotates a master user password secret, Secrets Manager generates a new secret version for the existing secret. The new version of the secret contains the new master user password. Amazon RDS changes the master user password for the Multi-AZ DB cluster to match the password for the new secret version.

You can rotate a secret immediately instead of waiting for a scheduled rotation. To rotate a master user password secret in Secrets Manager, modify the Multi-AZ DB cluster. For information about modifying a Multi-AZ DB cluster, see Modifying a Multi-AZ DB cluster for Amazon RDS.

You can rotate a master user password secret immediately with the RDS console, the AWS CLI, or the RDS API. The new password is always 28 characters long and contains atleast one upper and lowercase character, one number, and one punctuation.

Console

To rotate a master user password secret using the RDS console, modify the Multi-AZ DB cluster and select **Rotate secret immediately** in **Settings**.



Follow the instructions for modifying a Multi-AZ DB cluster with the RDS console in Modifying a Multi-AZ DB cluster for Amazon RDS. You must choose **Apply immediately** on the confirmation page.

AWS CLI

To rotate a master user password secret using the AWS CLI, use the modify-db-cluster command and specify the `--rotate-master-user-password` option. You must specify the `--apply-immediately` option when you rotate the master password.

This example rotates a master user password secret.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
```

```
    --db-cluster-identifier mydbcluster \
    --rotate-master-user-password \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --rotate-master-user-password ^
    --apply-immediately
```

RDS API

You can rotate a master user password secret using the ModifyDBCluster operation and setting the `RotateMasterUserPassword` parameter to `true`. You must set the `ApplyImmediately` parameter to `true` when you rotate the master password.

Viewing the details about a secret for a DB instance

You can retrieve your secrets using the console (https://console.aws.amazon.com/secretsmanager/) or the AWS CLI (get-secret-value Secrets Manager command).

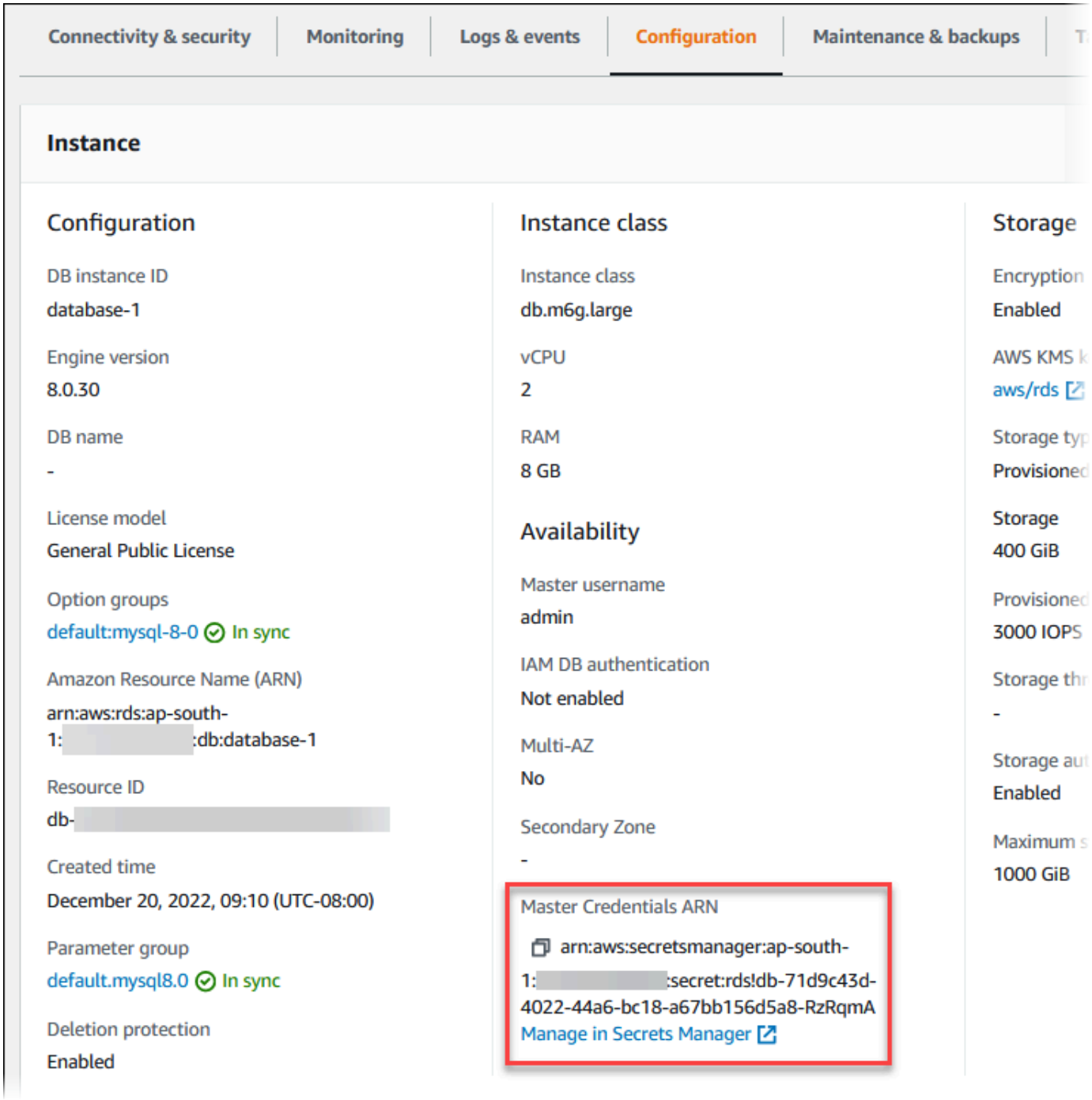
You can find the Amazon Resource Name (ARN) of a secret managed by RDS in Secrets Manager with the RDS console, the AWS CLI, or the RDS API.

Console

To view the details about a secret managed by RDS in Secrets Manager

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance to show its details.
4. Choose the **Configuration** tab.

   In **Master Credentials ARN**, you can view the secret ARN.



You can follow the **Manage in Secrets Manager** link to view and manage the secret in the Secrets Manager console.

AWS CLI

You can use the describe-db-instances RDS CLI command to find the following information about a secret managed by RDS in Secrets Manager:

- `SecretArn` – The ARN of the secret
- `SecretStatus` – The status of the secret

  The possible status values include the following:

  - `creating` – The secret is being created.
  - `active` – The secret is available for normal use and rotation.
  - `rotating` – The secret is being rotated.
  - `impaired` – The secret can be used to access database credentials, but it can't be rotated. A secret might have this status if, for example, permissions are changed so that RDS can no longer access the secret or the KMS key for the secret.

    When a secret has this status, you can correct the condition that caused the status. If you correct the condition that caused status, the status remains `impaired` until the next rotation. Alternatively, you can modify the DB instance to turn off automatic management of database credentials, and then modify the DB instance again to turn on automatic management of database credentials. To modify the DB instance, use the `--manage-master-user-password` option in the modify-db-instance command.
- `KmsKeyId` – The ARN of the KMS key that is used to encrypt the secret

Specify the `--db-instance-identifier` option to show output for a specific DB instance. This example shows the output for a secret that is used by a DB instance.

Example

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

Following is sample output for a secret:

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
db-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

When you have the secret ARN, you can view details about the secret using the get-secret-value Secrets Manager CLI command.

This example shows the details for the secret in the previous sample output.

Example

For Linux, macOS, or Unix:

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
db-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

For Windows:

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
db-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

RDS API

You can view the ARN, status, and KMS key for a secret managed by RDS in Secrets Manager by using the DescribeDBInstances operation and setting the `DBInstanceIdentifier` parameter to a DB instance identifier. Details about the secret are included in the output.

When you have the secret ARN, you can view details about the secret using the GetSecretValue Secrets Manager operation.

Viewing the details about a secret for a Multi-AZDB cluster

You can retrieve your secrets using the console (https://console.aws.amazon.com/secretsmanager/) or the AWS CLI (get-secret-value Secrets Manager command).

You can find the Amazon Resource Name (ARN) of a secret managed by RDS in Secrets Manager with the RDS console, the AWS CLI, or the RDS API.

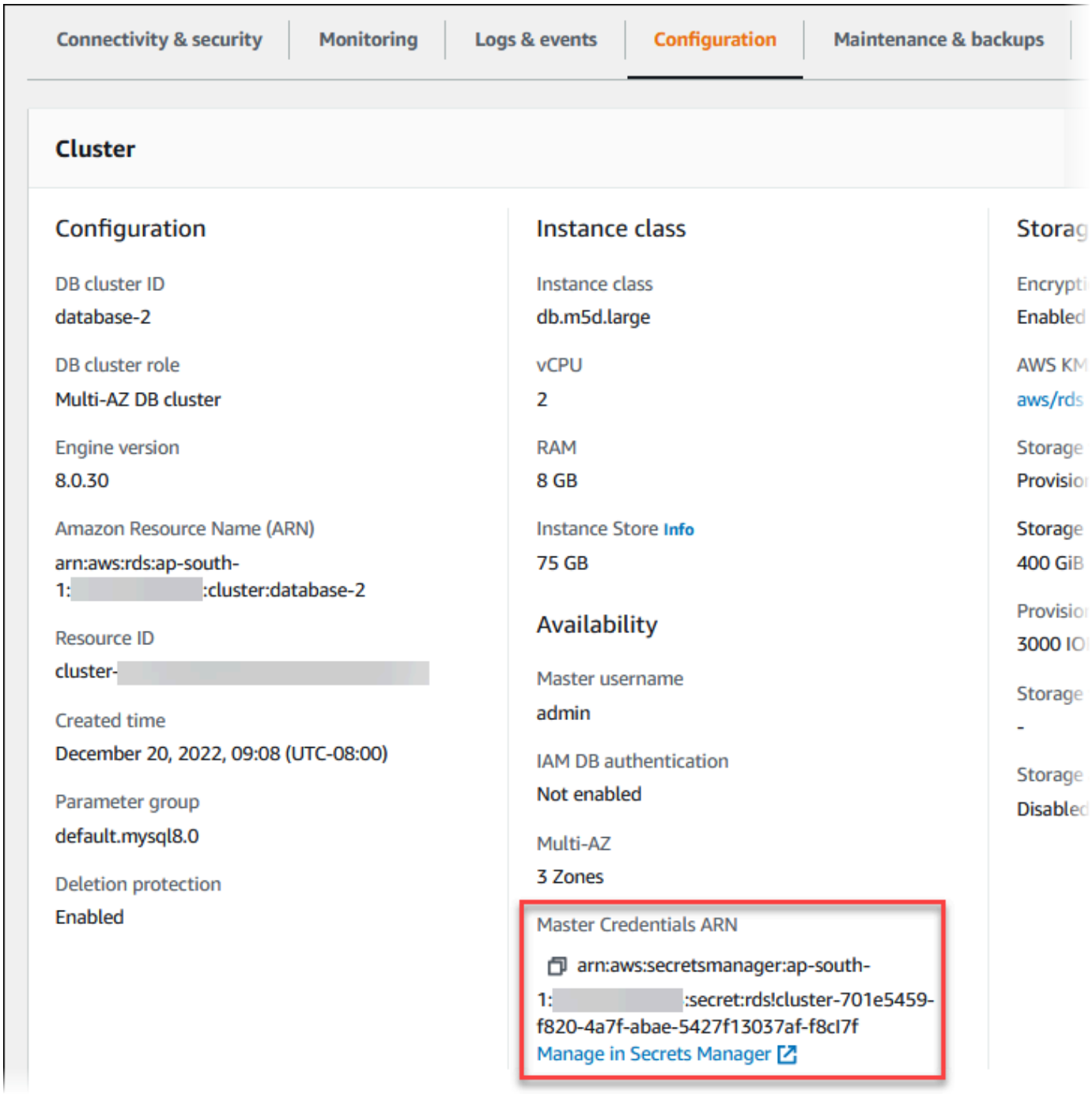
Console

To view the details about a secret managed by RDS in Secrets Manager

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. In the navigation pane, choose **Databases**.
3. Choose the name of the Multi-AZ DB cluster to show its details.
4. Choose the **Configuration** tab.

   In **Master Credentials ARN**, you can view the secret ARN.



You can follow the **Manage in Secrets Manager** link to view and manage the secret in the Secrets Manager console.

AWS CLI

You can use the RDS AWS CLI describe-db-clusters command to find the following information about a secret managed by RDS in Secrets Manager:

- `SecretArn` – The ARN of the secret
- `SecretStatus` – The status of the secret

  The possible status values include the following:

  - `creating` – The secret is being created.
  - `active` – The secret is available for normal use and rotation.
  - `rotating` – The secret is being rotated.
  - `impaired` – The secret can be used to access database credentials, but it can't be rotated. A secret might have this status if, for example, permissions are changed so that RDS can no longer access the secret or the KMS key for the secret.

    When a secret has this status, you can correct the condition that caused the status. If you correct the condition that caused status, the status remains `impaired` until the next rotation. Alternatively, you can modify the DB cluster to turn off automatic management of database credentials, and then modify the DB cluster again to turn on automatic management of database credentials. To modify the DB cluster, use the `--manage-master-user-password` option in the modify-db-cluster command.
- `KmsKeyId` – The ARN of the KMS key that is used to encrypt the secret

Specify the `--db-cluster-identifier` option to show output for a specific DB cluster. This example shows the output for a secret that is used by a DB cluster.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydbcluster
```

The following sample shows the output for a secret:

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

When you have the secret ARN, you can view details about the secret using the get-secret-value Secrets Manager CLI command.

This example shows the details for the secret in the previous sample output.

Example

For Linux, macOS, or Unix:

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

For Windows:

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

RDS API

You can view the ARN, status, and KMS key for a secret managed by RDS in Secrets Manager using the DescribeDBClusters RDS operation and setting the `DBClusterIdentifier` parameter to a DB cluster identifier. Details about the secret are included in the output.

When you have the secret ARN, you can view details about the secret using the GetSecretValue Secrets Manager operation.

Viewing the details about a secret for a tenant database

You can retrieve your secrets using the console (https://console.aws.amazon.com/secretsmanager/) or the AWS CLI (get-secret-value Secrets Manager command).

You can find the Amazon Resource Name (ARN) of a secret managed by Amazon RDS in AWS Secrets Manager with the Amazon RDS console, the AWS CLI, or the Amazon RDS API.

Console

To view the details about a secret managed by Amazon RDS in AWS Secrets Manager for a tenant database

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that contains the tenant database to show its details.
4. Choose the **Configuration** tab.

   In the **Tenant databases** section, find the tenant database and view its **Master Credentials ARN**.

   You can follow the **Manage in Secrets Manager** link to view and manage the secret in the Secrets Manager console.

AWS CLI

You can use the describe-tenant-databases Amazon RDS AWS CLI command to find the following information about a secret managed by Amazon RDS in AWS Secrets Manager for a tenant database:

- `SecretArn` – The ARN of the secret
- `SecretStatus` – The status of the secret

  The possible status values include the following:

  - `creating` – The secret is being created.
  - `active` – The secret is available for normal use and rotation.
  - `rotating` – The secret is being rotated.
  - `impaired` – The secret can be used to access database credentials, but it can't be rotated. A secret might have this status if, for example, permissions are changed so that Amazon RDS can no longer access the secret or the KMS key for the secret.

When a secret has this status, you can correct the condition that caused the status. If you correct the condition that caused status, the status remains `impaired` until the next rotation. Alternatively, you can modify the tenant database to turn off automatic management of database credentials, and then modify the tenant database again to turn on automatic management of database credentials. To modify the tenant database, use the `--manage-master-user-password` option in the modify-tenant-database command.

- `KmsKeyId` – The ARN of the KMS key that is used to encrypt the secret

Specify the `--db-instance-identifier` option to show output for tenant databases in a specific DB instance. You can also specify the `--tenant-db-name` option to show output for a specific tenant database. This example shows the output for a secret that is used by a tenant database.

Example

```
aws rds describe-tenant-databases \
    --db-instance-identifier database-3 \
    --query "TenantDatabases[0].MasterUserSecret"
```

Following is sample output for a secret:

```
{
    "SecretArn": "arn:aws:secretsmanager:us-east-2:123456789012:secret:rds!db-ABC123",
    "SecretStatus": "active",
    "KmsKeyId": "arn:aws:kms:us-east-2:123456789012:key/aa11bb22-####-####-####-
fedcba123456"
}
```

When you have the secret ARN, you can view details about the secret using the get-secret-value Secrets Manager AWS CLI command.

This example shows the details for the secret in the previous sample output.

Example

For Linux, macOS, or Unix:

```
aws secretsmanager get-secret-value \
```

```
    --secret-id 'arn:aws:secretsmanager:us-east-2:123456789012:secret:rds!db-ABC123'
```

For Windows:

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:us-east-2:123456789012:secret:rds!db-ABC123'
```

Amazon RDS API

You can view the ARN, status, and KMS key for a secret managed by Amazon RDS in AWS Secrets Manager by using the DescribeTenantDatabases operation and setting the `DBInstanceIdentifier` parameter to a DB instance identifier. You can also set the `TenantDBName` parameter to a specific tenant database name. Details about the secret are included in the output.

When you have the secret ARN, you can view details about the secret using the GetSecretValue Secrets Manager operation.

Region and version availability

Feature availability and support varies across specific versions of each database engine and across AWS Regions. For more information about version and Region availability with Secrets Manager integration with Amazon RDS, see Supported Regions and DB engines for the Secrets Manager integration with Amazon RDS.

Data protection in Amazon RDS

The AWS shared responsibility model applies to data protection in Amazon Relational Database Service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM).

That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon RDS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Topics

- Protecting data using encryption
- Internetwork traffic privacy

Protecting data using encryption

You can enable encryption for database resources. You can also encrypt connections to DB instances.

Topics

- Encrypting Amazon RDS resources
- AWS KMS key management
- Using SSL/TLS to encrypt a connection to a DB instance or cluster

- Rotating your SSL/TLS certificate

Encrypting Amazon RDS resources

Amazon RDS can encrypt your Amazon RDS DB instances. Data that is encrypted at rest includes the underlying storage for DB instances, its logs, automated backups, read replicas, and snapshots.

Amazon RDS encrypted DB instances use the industry standard AES-256 encryption algorithm to encrypt your data on the server that hosts your Amazon RDS DB instances.

After your data is encrypted, Amazon RDS handles authentication of access and decryption of your data transparently with a minimal impact on performance. You don't need to modify your database client applications to use encryption.

> **Note**
>
> For encrypted and unencrypted DB instances, data that is in transit between the source and the read replicas is encrypted, even when replicating across AWS Regions.

Topics

- Overview of encrypting Amazon RDS resources
- Encrypting a DB instance
- Determining whether encryption is turned on for a DB instance
- Availability of Amazon RDS encryption
- Encryption in transit
- Limitations of Amazon RDS encrypted DB instances

Overview of encrypting Amazon RDS resources

Amazon RDS encrypted DB instances provide an additional layer of data protection by securing your data from unauthorized access to the underlying storage. You can use Amazon RDS encryption to increase data protection of your applications deployed in the cloud, and to fulfill compliance requirements for encryption at rest. For an Amazon RDS encrypted DB instance, all logs, backups, and snapshots are encrypted. For more information about the availability and limitations of encryption, see Availability of Amazon RDS encryption and Limitations of Amazon RDS encrypted DB instances.

Amazon RDS uses an AWS Key Management Service key to encrypt these resources. AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. You can use an AWS managed key, or you can create customer managed keys.

When you create an encrypted DB instance, you can choose a customer managed key or the AWS managed key for Amazon RDS to encrypt your DB instance. If you don't specify the key identifier for a customer managed key, Amazon RDS uses the AWS managed key for your new DB instance. Amazon RDS creates an AWS managed key for Amazon RDS for your AWS account. Your AWS account has a different AWS managed key for Amazon RDS for each AWS Region.

To manage the customer managed keys used for encrypting and decrypting your Amazon RDS resources, you use the AWS Key Management Service (AWS KMS).

Using AWS KMS, you can create customer managed keys and define the policies that control the use of these customer managed keys. AWS KMS supports CloudTrail, so you can audit KMS key usage to verify that customer managed keys are being used appropriately. You can use your customer managed keys with Amazon Aurora and supported AWS services such as Amazon S3, Amazon EBS, and Amazon Redshift. For a list of services that are integrated with AWS KMS, see AWS Service Integration. Some considerations about using KMS keys:

- Once you have created an encrypted DB instance, you can't change the KMS key used by that DB instance. Therefore, be sure to determine your KMS key requirements before you create your encrypted DB instance.

  If you must change the encryption key for your DB instance, create a manual snapshot of your instance and enable encryption while copying the snapshot. For more information, see re:Post Knowledge article.
- If you copy an encrypted snapshot, you can use a different KMS key to encrypt the target snapshot than the one that was used to encrypt the source snapshot.
- A read replica of an Amazon RDS encrypted instance must be encrypted using the same KMS key as the primary DB instance when both are in the same AWS Region.
- If the primary DB instance and read replica are in different AWS Regions, you encrypt the read replica using the KMS key for that AWS Region.
- You can't share a snapshot that has been encrypted using the AWS managed key of the AWS account that shared the snapshot.
- Amazon RDS also supports encrypting an Oracle or SQL Server DB instance with Transparent Data Encryption (TDE). TDE can be used with RDS encryption at rest, although using TDE and RDS encryption at rest simultaneously might slightly affect the performance of your database.

You must manage different keys for each encryption method. For more information on TDE, see Oracle Transparent Data Encryption or Support for Transparent Data Encryption in SQL Server.

Important

Amazon RDS loses access to the KMS key for a DB instance when you disable the KMS key. If you lose access to a KMS key, the encrypted DB instance goes into the `inaccessible-encryption-credentials-recoverable` state 2 hours after detection in instances where backups are enabled. The DB instance remains in this state for seven days, during which the instance is stopped. API calls made to the DB instance during this time might not succeed. To recover the DB instance, enable the KMS key and restart this DB instance. Enable the KMS key from the AWS Management Console, AWS CLI, or RDS API. Restart the DB instance using the AWS CLI command start-db-instance or AWS Management Console. The `inaccessible-encryption-credentials-recoverable` state only applies to DB instances that can stop. This recoverable state is not applicable to instances that can't stop, such as read replicas and instances with read replicas. For more information, see Limitations of stopping your DB instance.

If the DB instance isn't recovered within seven days, it goes into the terminal `inaccessible-encryption-credentials` state. In this state, the DB instance is not usable anymore and you can only restore the DB instance from a backup. We strongly recommend that you always turn on backups for encrypted DB instances to guard against the loss of encrypted data in your databases.

During the creation of a DB instance, Amazon RDS checks if the calling principal has access to the KMS key and generates a grant from the KMS key that it uses for the entire lifetime of the DB instance. Revoking the calling principal's access to the KMS key does not affect a running database. When using KMS keys in cross-account scenarios, such as copying a snapshot to another account, the KMS key needs to be shared with the other account. If you create a DB instance from the snapshot without specifying a different KMS key, the new instance uses the KMS key from the source account. Revoking access to the key after you create the DB instance does not affect the instance. However, disabling the key impacts all DB instances encrypted with that key. To prevent this, specify a different key during the snapshot copy operation.

DB instances with disabled backups remain available until the volumes are detached from the host during an instance modification or a recovery. RDS moves the instances into

`inaccessible-encryption-credentials-recoverable` state or `inaccessible-encryption-credentials` state as applicable.

For more information about KMS keys, see AWS KMS keys in the *AWS Key Management Service Developer Guide* and AWS KMS key management.

Encrypting a DB instance

To encrypt a new DB instance, choose **Enable encryption** on the Amazon RDS console. For information on creating a DB instance, see Creating an Amazon RDS DB instance.

If you use the create-db-instance AWS CLI command to create an encrypted DB instance, set the `--storage-encrypted` parameter. If you use the CreateDBInstance API operation, set the `StorageEncrypted` parameter to true.

If you use the AWS CLI `create-db-instance` command to create an encrypted DB instance with a customer managed key, set the `--kms-key-id` parameter to any key identifier for the KMS key. If you use the Amazon RDS API `CreateDBInstance` operation, set the `KmsKeyId` parameter to any key identifier for the KMS key. To use a customer managed key in a different AWS account, specify the key ARN or alias ARN.

Determining whether encryption is turned on for a DB instance

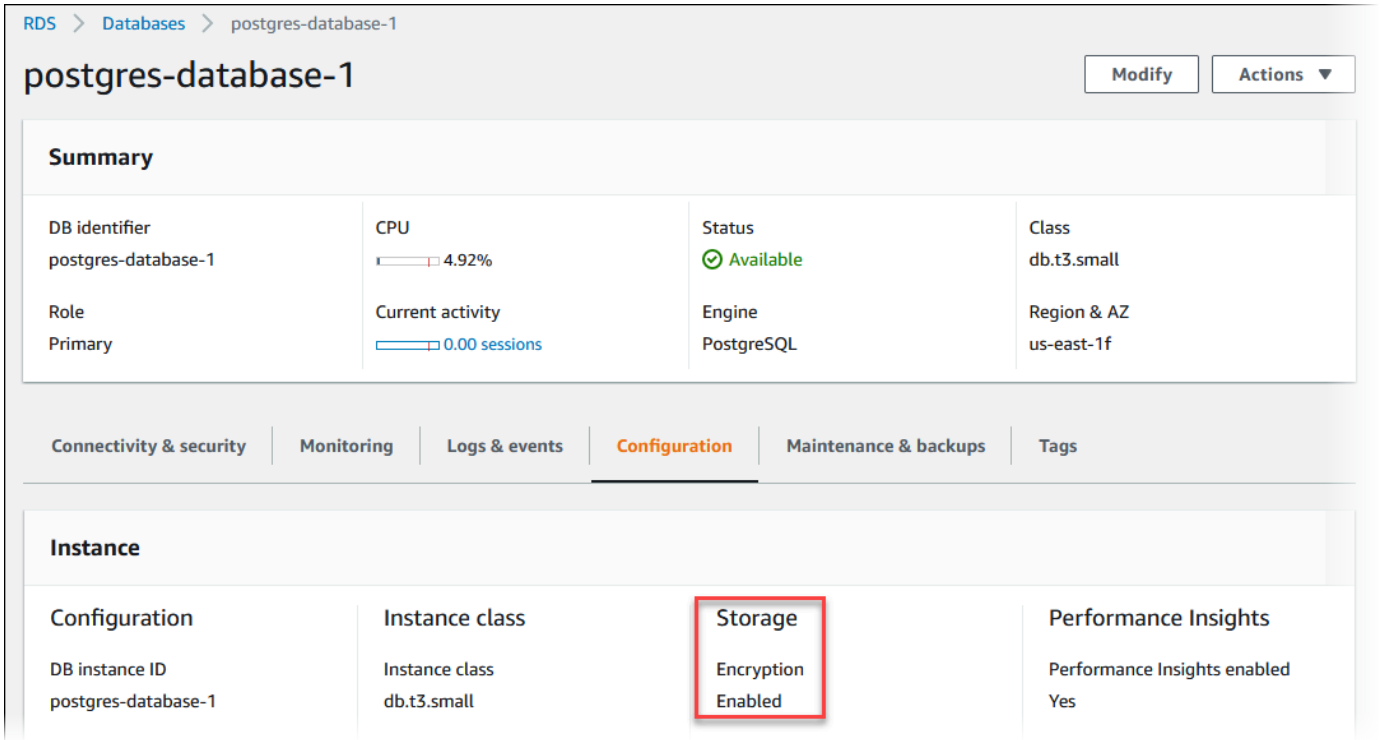
You can use the AWS Management Console, AWS CLI, or RDS API to determine whether encryption at rest is turned on for a DB instance.

Console

To determine whether encryption at rest is turned on for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that you want to check to view its details.
4. Choose the **Configuration** tab, and check the **Encryption** value under **Storage**.

   It shows either **Enabled** or **Not enabled**.



AWS CLI

To determine whether encryption at rest is turned on for a DB instance by using the AWS CLI, call the describe-db-instances command with the following option:

- `--db-instance-identifier` – The name of the DB instance.

The following example uses a query to return either TRUE or FALSE regarding encryption at rest for the mydb DB instance.

Example

```
aws rds describe-db-instances --db-instance-identifier mydb --query "*[].
{StorageEncrypted:StorageEncrypted}" --output text
```

RDS API

To determine whether encryption at rest is turned on for a DB instance by using the Amazon RDS API, call the DescribeDBInstances operation with the following parameter:

- `DBInstanceIdentifier` – The name of the DB instance.

Availability of Amazon RDS encryption

Amazon RDS encryption is currently available for all database engines and storage types.

Amazon RDS encryption is available for most DB instance classes. The following table lists DB instance classes that *don't support* Amazon RDS encryption:

Instance type	Instance class
General purpose (M1)	db.m1.small db.m1.medium db.m1.large db.m1.xlarge
Memory optimized (M2)	db.m2.xlarge db.m2.2xlarge db.m2.4xlarge
Burstable (T2)	db.t2.micro

Encryption in transit

Encryption at the physical layer

All data flowing accross AWS Regions over the AWS global network is automatically encrypted at the physical layer before it leaves AWS secured facilities. All traffic between AZs is encrypted. Additional layers of encryption, including those listed in this section may provide additional protections.

Encryption provided by Amazon VPC peering and Transit Gateway cross-Region peering

All cross-Region traffic that uses Amazon VPC and Transit Gateway peering is automatically bulk-encrypted when it exits a Region. An additional layer of encryption is automatically provided at the physical layer for all traffic before it leaves AWS secured facilities.

Encryption between instances

AWS provides secure and private connectivity between DB instances of all types. In addition, some instance types use the offload capabilities of the underlying Nitro System hardware to automatically encrypt in-transit traffic between instances. This encryption uses Authenticated Encryption with Associated Data (AEAD) algorithms, with 256-bit encryption. There is no impact on network performance. To support this additional in-transit traffic encryption between instances, the following requirements must be met:

- The instances use the following instance types:
  - **General purpose**: M6i, M6id, M6in, M6idn, M7g
  - **Memory optimized**: R6i, R6id, R6in, R6idn, R7g, X2idn, X2iedn, X2iezn
- The instances are in the same AWS Region.
- The instances are in the same VPC or peered VPCs, and the traffic does not pass through a virtual network device or service, such as a load balancer or a transit gateway.

Limitations of Amazon RDS encrypted DB instances

The following limitations exist for Amazon RDS encrypted DB instances:

- You can only encrypt an Amazon RDS DB instance when you create it, not after the DB instance is created.

  However, because you can encrypt a copy of an unencrypted snapshot, you can effectively add encryption to an unencrypted DB instance. That is, you can create a snapshot of your DB instance, and then create an encrypted copy of that snapshot. You can then restore a DB instance from the encrypted snapshot, and thus you have an encrypted copy of your original DB instance. For more information, see Copying a DB snapshot for Amazon RDS.
- You can't turn off encryption on an encrypted DB instance.
- You can't create an encrypted snapshot of an unencrypted DB instance.
- A snapshot of an encrypted DB instance must be encrypted using the same KMS key as the DB instance.
- You can't have an encrypted read replica of an unencrypted DB instance or an unencrypted read replica of an encrypted DB instance.
- Encrypted read replicas must be encrypted with the same KMS key as the source DB instance when both are in the same AWS Region.

- You can't restore an unencrypted backup or snapshot to an encrypted DB instance.
- To copy an encrypted snapshot from one AWS Region to another, you must specify the KMS key in the destination AWS Region. This is because KMS keys are specific to the AWS Region that they are created in.

  The source snapshot remains encrypted throughout the copy process. Amazon RDS uses envelope encryption to protect data during the copy process. For more information about envelope encryption, see Envelope encryption in the *AWS Key Management Service Developer Guide*.
- You can't unencrypt an encrypted DB instance. However, you can export data from an encrypted DB instance and import the data into an unencrypted DB instance.

AWS KMS key management

Amazon RDS automatically integrates with AWS Key Management Service (AWS KMS) for key management. Amazon RDS uses envelope encryption. For more information about envelope encryption, see Envelope encryption in the *AWS Key Management Service Developer Guide*.

You can use two types of AWS KMS keys to encrypt your DB instances .

- If you want full control over a KMS key, you must create a *customer managed key*. For more information about customer managed keys, see Customer managed keys in the *AWS Key Management Service Developer Guide*.
- *AWS managed keys* are KMS keys in your account that are created, managed, and used on your behalf by an AWS service that is integrated with AWS KMS. By default, the RDS AWS managed key (`aws/rds`) is used for encryption. You can't manage, rotate, or delete the RDS AWS managed key. For more information about AWS managed keys, see AWS managed keys in the *AWS Key Management Service Developer Guide*.

To manage KMS keys used for Amazon RDS encrypted DB instances , use the AWS Key Management Service (AWS KMS) in the AWS KMS console, the AWS CLI, or the AWS KMS API. To view audit logs of every action taken with an AWS managed or customer managed key, use AWS CloudTrail. For more information about key rotation, see Rotating AWS KMS keys.

Authorizing use of a customer managed key

When RDS uses a customer managed key in cryptographic operations, it acts on behalf of the user who is creating or changing the RDS resource.

To create an RDS resource using a customer managed key, a user must have permissions to call the following operations on the customer managed key:

- `kms:CreateGrant`
- `kms:DescribeKey`

You can specify these required permissions in a key policy, or in an IAM policy if the key policy allows it.

Important

When you use explicit deny statements for all resources (*) in AWS KMS key policies with managed services like Amazon RDS, you must specify a condition to allow the resource owning account. Operations might fail without this condition, even if the deny rule includes exceptions for your IAM user.

Tip

To follow the principle of least privilege, do not allow full access to `kms:CreateGrant`. Instead, use the kms:ViaService condition key to allow the user to create grants on the KMS key only when the grant is created on the user's behalf by an AWS service.

You can make the IAM policy stricter in various ways. For example, if you want to allow the customer managed key to be used only for requests that originate in RDS , use the kms:ViaService condition key with the `rds.<region>.amazonaws.com` value. Also, you can use the keys or values in the Amazon RDS encryption context as a condition for using the customer managed key for encryption.

For more information, see Allowing users in other accounts to use a KMS key in the *AWS Key Management Service Developer Guide* and Key policies in AWS KMS.

Amazon RDS encryption context

When RDS uses your KMS key, or when Amazon EBS uses the KMS key on behalf of RDS , the service specifies an encryption context. The encryption context is additional authenticated data (AAD) that AWS KMS uses to ensure data integrity. When an encryption context is specified for an

encryption operation, the service must specify the same encryption context for the decryption operation. Otherwise, decryption fails. The encryption context is also written to your AWS CloudTrail logs to help you understand why a given KMS key was used. Your CloudTrail logs might contain many entries describing the use of a KMS key, but the encryption context in each log entry can help you determine the reason for that particular use.

At minimum, Amazon RDS always uses the DB instance ID for the encryption context, as in the following JSON-formatted example:

```
{ "aws:rds:db-id": "db-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

This encryption context can help you identify the DB instance for which your KMS key was used.

When your KMS key is used for a specific DB instance and a specific Amazon EBS volume, both the DB instance ID and the Amazon EBS volume ID are used for the encryption context, as in the following JSON-formatted example:

```
{
  "aws:rds:db-id": "db-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

Using SSL/TLS to encrypt a connection to a DB instance or cluster

You can use Secure Socket Layer (SSL) or Transport Layer Security (TLS) from your application to encrypt a connection to a database running Db2, MariaDB, Microsoft SQL Server, MySQL, Oracle, or PostgreSQL.

SSL/TLS connections provide a layer of security by encrypting data that moves between your client and DB instance or cluster . Optionally, your SSL/TLS connection can perform server identity verification by validating the server certificate installed on your database. To require server identity verification, follow this general process:

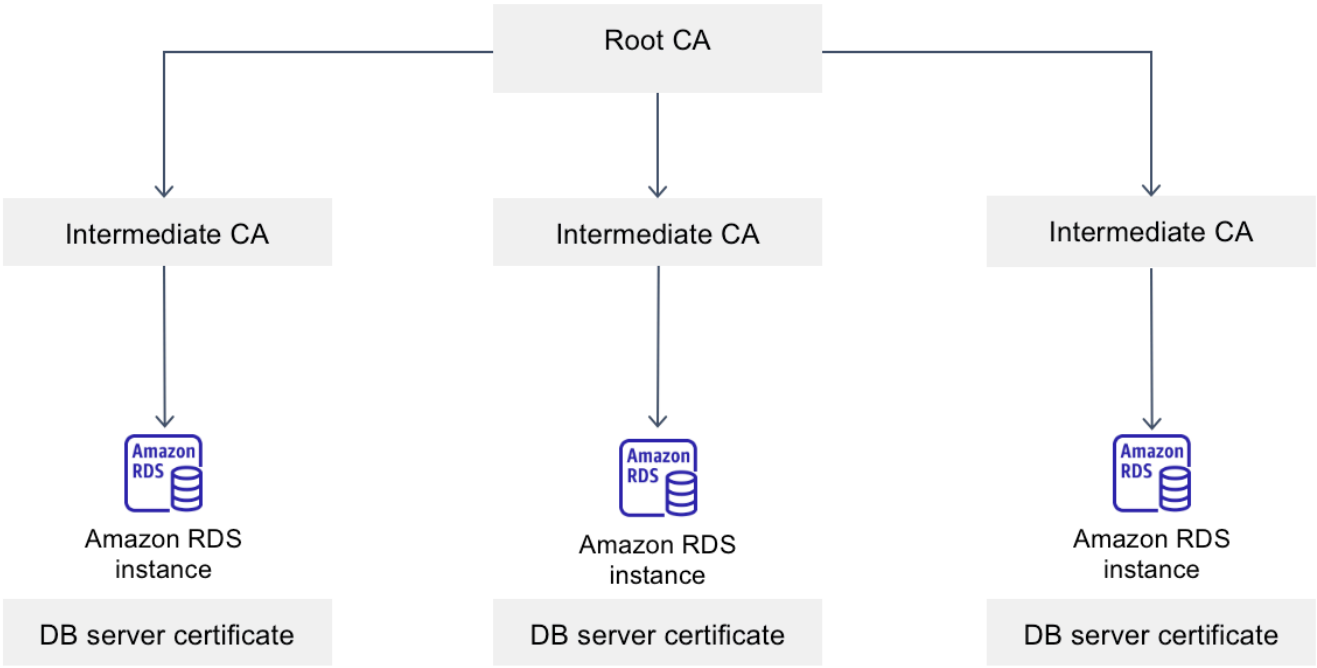
1. Choose the **certificate authority (CA)** that signs the **DB server certificate,** for your database. For more information about certificate authorities, see Certificate authorities .
2. Download a certificate bundle to use when you are connecting to the database. To download a certificate bundle, see Certificate bundles by AWS Region .

> **Note**
>
> All certificates are only available for download using SSL/TLS connections.

3. Connect to the database using your DB engine's process for implementing SSL/TLS connections. Each DB engine has its own process for implementing SSL/TLS. To learn how to implement SSL/TLS for your database, follow the link that corresponds to your DB engine:
   - Using SSL/TLS with an Amazon RDS for Db2 DB instance
   - SSL/TLS support for MariaDB DB instances on Amazon RDS
   - Using SSL with a Microsoft SQL Server DB instance
   - SSL/TLS support for MySQL DB instances on Amazon RDS
   - Using SSL with an RDS for Oracle DB instance
   - Using SSL with a PostgreSQL DB instance

Certificate authorities

The **certificate authority (CA)** is the certificate that identifies the root CA at the top of the certificate chain. The CA signs the **DB server certificate,** which is installed on each DB instance. The DB server certificate identifies the DB instance as a trusted server.



Amazon RDS provides the following CAs to sign the DB server certificate for a database.

Certificate authority (CA)	Description	Common name (CN)
rds-ca-rsa2048-g1	Uses a certificate authority with RSA 2048 private key algorithm and SHA256 signing algorithm in most AWS Regions. In the AWS GovCloud (US) Regions, this CA uses a certificate authority with RSA 2048 private key algorithm and SHA384 signing algorithm. This CA supports automatic server certificate rotation.	Amazon RDS *region-id entifier* Root CA RSA2048 G1
rds-ca-rsa4096-g1	Uses a certificate authority with RSA 4096 private key algorithm and SHA384 signing algorithm. This CA supports automatic server certificate rotation.	Amazon RDS *region-id entifier* Root CA RSA4096 G1
rds-ca-ecc384-g1	Uses a certificate authority with ECC 384 private key algorithm and SHA384 signing algorithm. This CA supports automatic server certificate rotation.	Amazon RDS *region-id entifier* Root CA ECC384 G1

Note

If you are using the AWS CLI, you can see the validities of the certificate authorities listed above by using describe-certificates.

These CA certificates are included in the regional and global certificate bundle. When you use the rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, or rds-ca-ecc384-g1 CA with a database, RDS manages the DB server certificate on the database. RDS rotates the DB server certificate automatically before it expires.

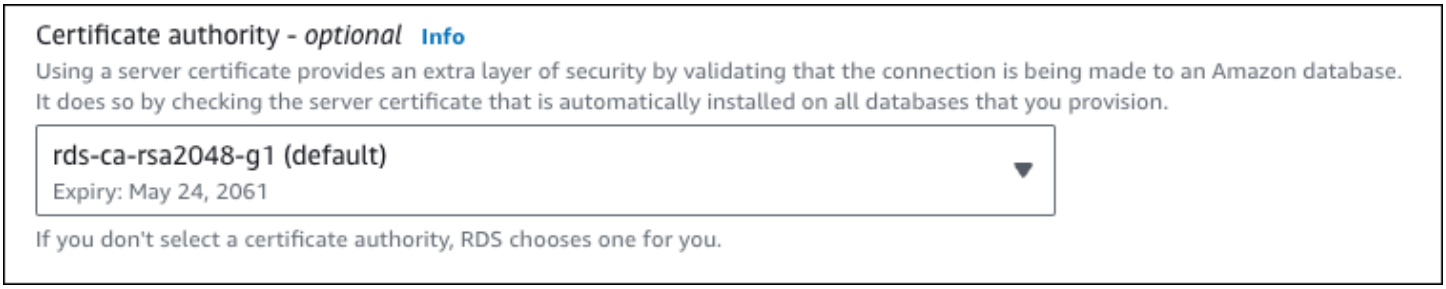
Setting the CA for your database

You can set the CA for a database when you perform the following tasks:

- Create a DB instance or Multi-AZ DB cluster – You can set the CA when you create a DB instance or cluster. For instructions, see the section called "Creating a DB instance" or the section called "Creating a Multi-AZ DB cluster" .
- Modify a DB instance or Multi-AZ DB cluster – You can set the CA for a DB instance or cluster by modifying it. For instructions, see the section called "Modifying a DB instance" or the section called "Modifying a Multi-AZ DB cluster" .

> **Note**
>
> The default CA is set to rds-ca-rsa2048-g1. You can override the default CA for your AWS account by using the modify-certificates command.

The available CAs depend on the DB engine and DB engine version. When you use the AWS Management Console, you can choose the CA using the **Certificate authority** setting, as shown in the following image.



The console only shows the CAs that are available for the DB engine and DB engine version. If you're using the AWS CLI, you can set the CA for a DB instance using the create-db-instance or modify-db-instance command. You can set the CA for a Multi-AZ DB cluster using the create-db-cluster or modify-db-cluster command.

If you're using the AWS CLI, you can see the available CAs for your account by using the describe-certificates command. This command also shows the expiration date for each CA in `ValidTill` in the output. You can find the CAs that are available for a specific DB engine and DB engine version using the describe-db-engine-versions command.

The following example shows the CAs available for the default RDS for PostgreSQL DB engine version.

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

Your output is similar to the following. The available CAs are listed in `SupportedCACertificateIdentifiers`. The output also shows whether the DB engine version supports rotating the certificate without restart in `SupportsCertificateRotationWithoutRestart`.

```
{
    "DBEngineVersions": [
        {
            "Engine": "postgres",
            "MajorEngineVersion": "13",
            "EngineVersion": "13.4",
            "DBParameterGroupFamily": "postgres13",
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 13.4-R1",
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [
                "Lambda"
            ],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "SupportsBabelfish": false,
            "SupportsCertificateRotationWithoutRestart": true,
            "SupportedCACertificateIdentifiers": [
                "rds-ca-rsa2048-g1",
                "rds-ca-ecc384-g1",
                "rds-ca-rsa4096-g1"
            ]
        }
    ]
}
```

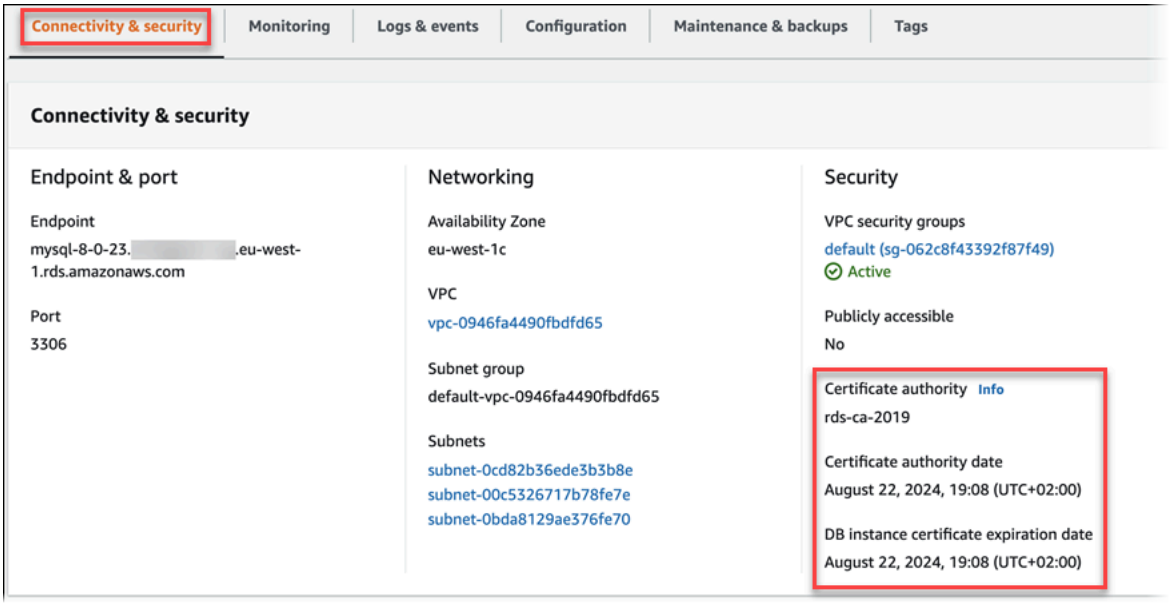
DB server certificate validities

The validity of DB server certificate depends on the DB engine and DB engine version. If the DB engine version supports rotating the certificate without restart, the validity of the DB server certificate is 1 year. Otherwise the validity is 3 years.

For more information about DB server certificate rotation, see Automatic server certificate rotation .

Viewing the CA for your DB instance

You can view the details about the CA for a database by viewing the **Connectivity & security** tab in the console, as in the following image.



If you're using the AWS CLI, you can view the details about the CA for a DB instance by using the describe-db-instances command. You can view the details about the CA for a Multi-AZ DB cluster by using the describe-db-clusters command.

Download certificate bundles for Amazon RDS

When you connect to your database with SSL or TLS, the database instance requires a trust certificate from Amazon RDS. Select the appropriate link in the following table to download the bundle that corresponds with the AWS Region where you host your database.

Certificate bundles by AWS Region

The certificate bundles for all AWS Regions and GovCloud (US) Regions contain the following root CA certificates:

- `rds-ca-rsa2048-g1`
- `rds-ca-rsa4096-g1`
- `rds-ca-ecc384-g1`

The `rds-ca-rsa4096-g1` and `rds-ca-ecc384-g1` certificates are not available in the following Regions:

- Asia Pacific (Mumbai)
- Asia Pacific (Melbourne)
- Canada West (Calgary)
- Europe (Zurich)
- Europe (Spain)
- Israel (Tel Aviv)

Your application trust store needs to only register the root CA certificate. Do not register the intermediate CA certificates to your trust store as this might cause connection issues when RDS automatically rotates your DB server certificate.

> **Note**
>
> Amazon RDS Proxy uses certificates from the AWS Certificate Manager (ACM). If you're using RDS Proxy, you don't need to download Amazon RDS certificates or update applications that use RDS Proxy connections. For more information, see Using TLS/SSL with RDS Proxy .

To download a certificate bundle for an AWS Region, select the link for the AWS Region that hosts your database in the following table.

AWS Region	Certificate bundle (PEM)	Certificate bundle (PKCS7)
Any commercial AWS Region	global-bundle.pem	global-bundle.p7b
US East (N. Virginia)	us-east-1-bundle.pem	us-east-1-bundle.p7b
US East (Ohio)	us-east-2-bundle.pem	us-east-2-bundle.p7b

AWS Region	Certificate bundle (PEM)	Certificate bundle (PKCS7)
US West (N. California)	us-west-1-bundle.pem	us-west-1-bundle.p7b
US West (Oregon)	us-west-2-bundle.pem	us-west-2-bundle.p7b
Africa (Cape Town)	af-south-1-bundle.pem	af-south-1-bundle.p7b
Asia Pacific (Hong Kong)	ap-east-1-bundle.pem	ap-east-1-bundle.p7b
Asia Pacific (Hyderabad)	ap-south-2-bundle.pem	ap-south-2-bundle.p7b
Asia Pacific (Jakarta)	ap-southeast-3-bundle.pem	ap-southeast-3-bundle.p7b
Asia Pacific (Malaysia)	ap-southeast-5-bundle.pem	ap-southeast-5-bundle.p7b
Asia Pacific (Melbourne)	ap-southeast-4-bundle.pem	ap-southeast-4-bundle.p7b
Asia Pacific (Mumbai)	ap-south-1-bundle.pem	ap-south-1-bundle.p7b
Asia Pacific (Osaka)	ap-northeast-3-bundle.pem	ap-northeast-3-bundle.p7b
Asia Pacific (Thailand)	ap-southeast-7-bundle.pem	ap-southeast-7-bundle.p7b
Asia Pacific (Tokyo)	ap-northeast-1-bundle.pem	ap-northeast-1-bundle.p7b
Asia Pacific (Seoul)	ap-northeast-2-bundle.pem	ap-northeast-2-bundle.p7b
Asia Pacific (Singapore)	ap-southeast-1-bundle.pem	ap-southeast-1-bundle.p7b
Asia Pacific (Sydney)	ap-southeast-2-bundle.pem	ap-southeast-2-bundle.p7b
Canada (Central)	ca-central-1-bundle.pem	ca-central-1-bundle.p7b
Canada West (Calgary)	ca-west-1-bundle.pem	ca-west-1-bundle.p7b
Europe (Frankfurt)	eu-central-1-bundle.pem	eu-central-1-bundle.p7b
Europe (Ireland)	eu-west-1-bundle.pem	eu-west-1-bundle.p7b
Europe (London)	eu-west-2-bundle.pem	eu-west-2-bundle.p7b

AWS Region	Certificate bundle (PEM)	Certificate bundle (PKCS7)
Europe (Milan)	eu-south-1-bundle.pem	eu-south-1-bundle.p7b
Europe (Paris)	eu-west-3-bundle.pem	eu-west-3-bundle.p7b
Europe (Spain)	eu-south-2-bundle.pem	eu-south-2-bundle.p7b
Europe (Stockholm)	eu-north-1-bundle.pem	eu-north-1-bundle.p7b
Europe (Zurich)	eu-central-2-bundle.pem	eu-central-2-bundle.p7b
Israel (Tel Aviv)	il-central-1-bundle.pem	il-central-1-bundle.p7b
Mexico (Central)	mx-central-1-bundle.pem	mx-central-1-bundle.p7b
Middle East (Bahrain)	me-south-1-bundle.pem	me-south-1-bundle.p7b
Middle East (UAE)	me-central-1-bundle.pem	me-central-1-bundle.p7b
South America (São Paulo)	sa-east-1-bundle.pem	sa-east-1-bundle.p7b
Any AWS GovCloud (US) Regions	global-bundle.pem	global-bundle.p7b
AWS GovCloud (US-East)	us-gov-east-1-bundle.pem	us-gov-east-1-bundle.p7b
AWS GovCloud (US-West)	us-gov-west-1-bundle.pem	us-gov-west-1-bundle.p7b

Viewing the contents of your CA certificate

To check the contents of your CA certificate bundle, use the following command:

```
keytool -printcert -v -file global-bundle.pem
```

Rotating your SSL/TLS certificate

Amazon RDS Certificate Authority certificates rds-ca-2019 expired in August, 2024. If you use or plan to use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) with certificate verification to connect to your RDS DB instances or Multi-AZ DB clusters,consider using one of the new CA

certificates rds-ca-rsa2048-g1, rds-ca-rsa4096-g1 or rds-ca-ecc384-g1. If you currently do not use SSL/TLS with certificate verification, you might still have an expired CA certificate and must update them to a new CA certificate if you plan to use SSL/TLS with certificate verification to connect to your RDS databases.

Amazon RDS provides new CA certificates as an AWS security best practice. For information about the new certificates and the supported AWS Regions, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

To update the CA certificate for your database, use the following methods:

- Updating your CA certificate by modifying your DB instance or cluster
- Updating your CA certificate by applying maintenance

Before you update your DB instances or Multi-AZ DB clusters to use the new CA certificate, make sure that you update your clients or applications connecting to your RDS databases.

Considerations for rotating certificates

Consider the following situations before rotating your certificate:

- Amazon RDS Proxy uses certificates from the AWS Certificate Manager (ACM). If you're using RDS Proxy, when you rotate your SSL/TLS certificate, you don't need to update applications that use RDS Proxy connections. For more information, see Using TLS/SSL with RDS Proxy .
- If you're using a Go version 1.15 application with a DB instance or Multi-AZ DB cluster that was created or updated to the rds-ca-2019 certificate prior to July 28, 2020, you must update the certificate again. Update the certificate to rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, or rds-ca-ecc384-g1 depending on your engine .

  Use the `modify-db-instance` command for a DB instance, or the `modify-db-cluster` command for a Multi-AZ DB cluster, using the new CA certificate identifier. You can find the CAs that are available for a specific DB engine and DB engine version using the `describe-db-engine-versions` command.

  If you created your database or updated its certificate after July 28, 2020, no action is required. For more information, see Go GitHub issue #39568.

Updating your CA certificate by modifying your DB instance or cluster

The following example updates your CA certificate from *rds-ca-2019* to *rds-ca-rsa2048-g1*.You can choose a different certificate. For more information, see Certificate authorities .

Update your application trust store to reduce any down time associated with updating your CA certificate. For more information about restarts associated with CA certificate rotation, see Automatic server certificate rotation .

To update your CA certificate by modifying your DB instance or cluster

1. Download the new SSL/TLS certificate as described in Using SSL/TLS to encrypt a connection to a DB instance or cluster .
2. Update your applications to use the new SSL/TLS certificate.

   The methods for updating applications for new SSL/TLS certificates depend on your specific applications. Work with your application developers to update the SSL/TLS certificates for your applications.

   For information about checking for SSL/TLS connections and updating applications for each DB engine, see the following topics:

   - Updating applications to connect to MariaDB instances using new SSL/TLS certificates
   - Updating applications to connect to Microsoft SQL Server DB instances using new SSL/TLS certificates
   - Updating applications to connect to MySQL DB instances using new SSL/TLS certificates
   - Updating applications to connect to Oracle DB instances using new SSL/TLS certificates
   - Updating applications to connect to PostgreSQL DB instances using new SSL/TLS certificates

   For a sample script that updates a trust store for a Linux operating system, see Sample script for importing certificates into your trust store .

   > **Note**
   >
   > The certificate bundle contains certificates for both the old and new CA, so you can upgrade your application safely and maintain connectivity during the transition period. If you are using the AWS Database Migration Service to migrate a database to a DB

instance or cluster , we recommend using the certificate bundle to ensure connectivity during the migration.

3. Modify the DB instance or Multi-AZ DB cluster to change the CA from **rds-ca-2019** to **rds-ca-rsa2048-g1**. To check if your database requires a restart to update the CA certificates, use the describe-db-engine-versions command and check the `SupportsCertificateRotationWithoutRestart` flag.

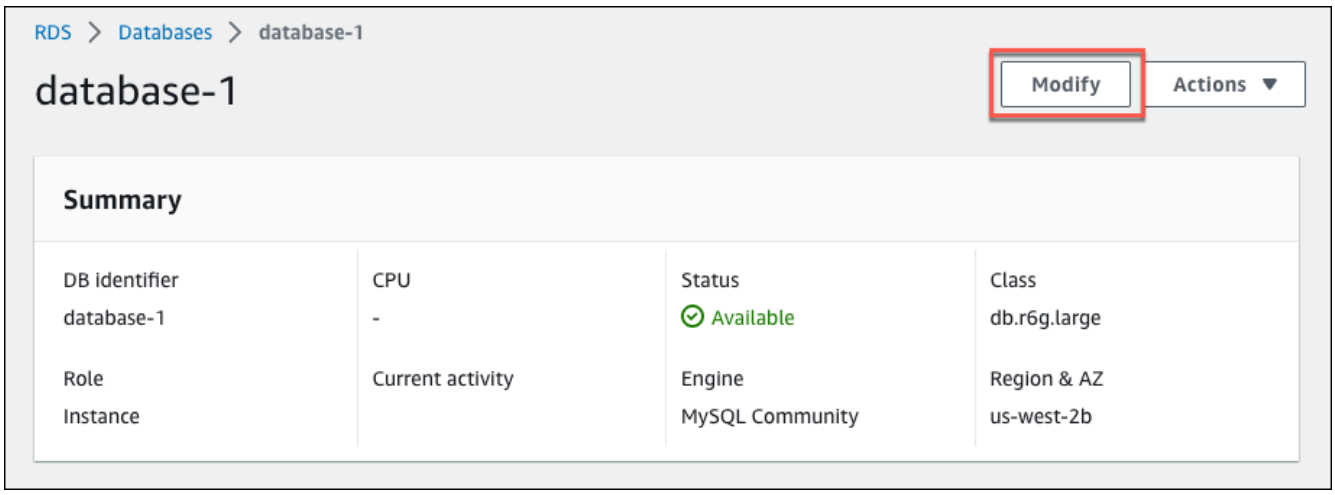
   Important

   If you are experiencing connectivity issues after certificate expiry, use the apply immediately option by specifying **Apply immediately** in the console or by specifying the `--apply-immediately` option using the AWS CLI. By default, this operation is scheduled to run during your next maintenance window.
   For RDS for Oracle DB instances, we recommend that you restart your Oracle DB to prevent any connection errors.
   For RDS for SQL Server Multi-AZ instances with AlwaysOn or Mirroring option enabled, a failover is expected when instance is rebooted after the certificate rotation.
   To set an override for your instance CA that's different from the default RDS CA, use the modify-certificates CLI command.

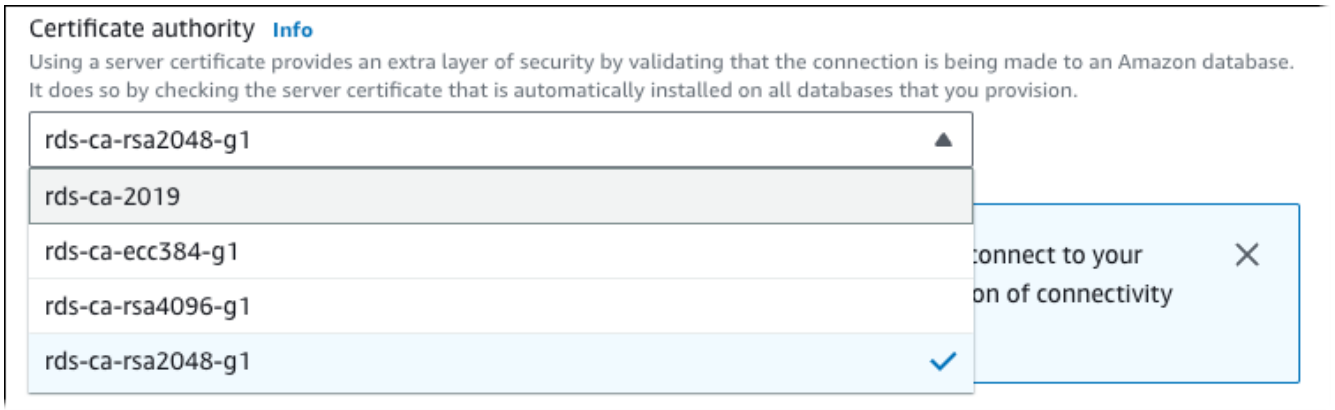
You can use the AWS Management Console or the AWS CLI to change the CA certificate from **rds-ca-2019** to **rds-ca-rsa2048-g1** for a DB instance or Multi-AZ DB cluster.

Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**, and then choose the DB instance or Multi-AZ DB cluster that you want to modify.
3. Choose **Modify**.



4. In the **Connectivity** section, choose **rds-ca-rsa2048-g1**.



5. Choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, choose **Apply immediately**.
7. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** or **Modify cluster** to save your changes.

> **Important**
>
> When you schedule this operation, make sure that you have updated your client-side trust store beforehand.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To use the AWS CLI to change the CA from **rds-ca-2019** to **rds-ca-rsa2048-g1** for a DB instance or Multi-AZ DB cluster, call the modify-db-instance or modify-db-cluster command. Specify the DB instance or cluster identifier and the `--ca-certificate-identifier` option.

Use the `--apply-immediately` parameter to apply the update immediately. By default, this operation is scheduled to run during your next maintenance window.

> **Important**
>
> When you schedule this operation, make sure that you have updated your client-side trust store beforehand.

Example

DB instance

The following example modifies `mydbinstance` by setting the CA certificate to `rds-ca-rsa2048-g1`.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

> **Note**
>
> If your instance requires reboot, you can use the modify-db-instance CLI command and specify the `--no-certificate-rotation-restart` option.

Example

Multi-AZ DB cluster

The following example modifies `mydbcluster` by setting the CA certificate to `rds-ca-rsa2048-g1`.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

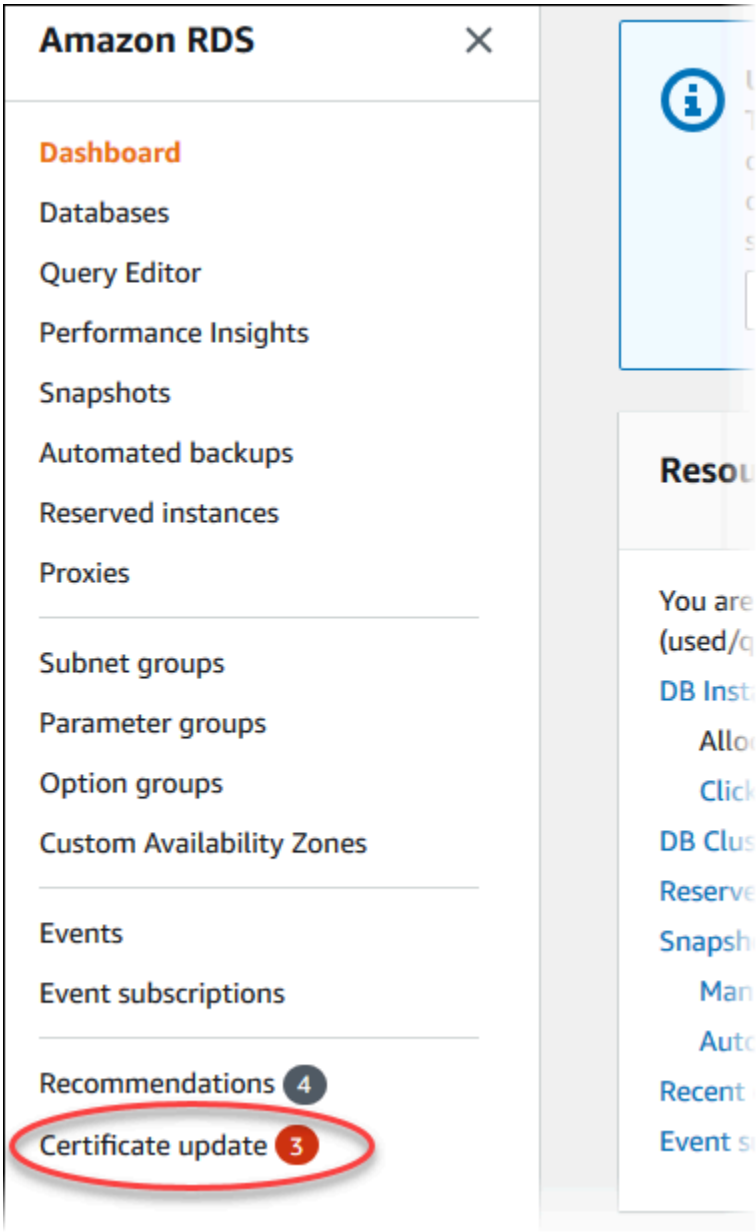
Updating your CA certificate by applying maintenance

Perform the following steps to update your CA certificate by applying maintenance.

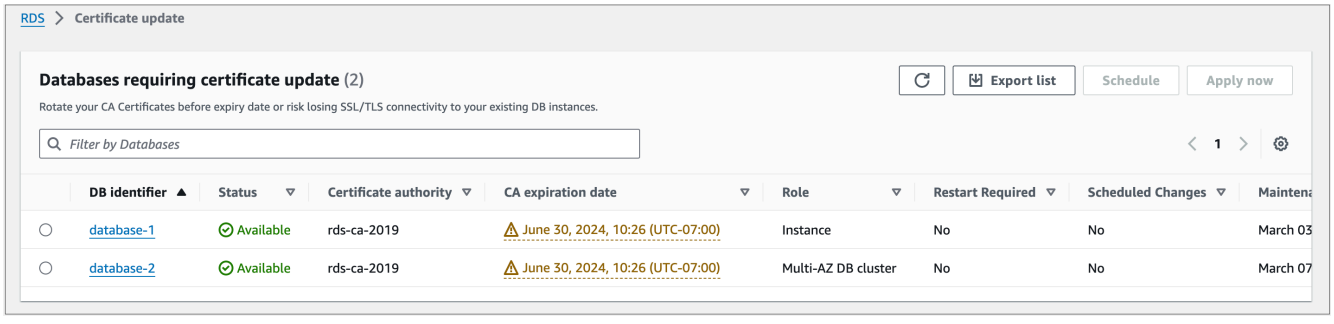
Console

To update your CA certificate by applying maintenance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Certificate update**.



The **Databases requiring certificate update** page appears.



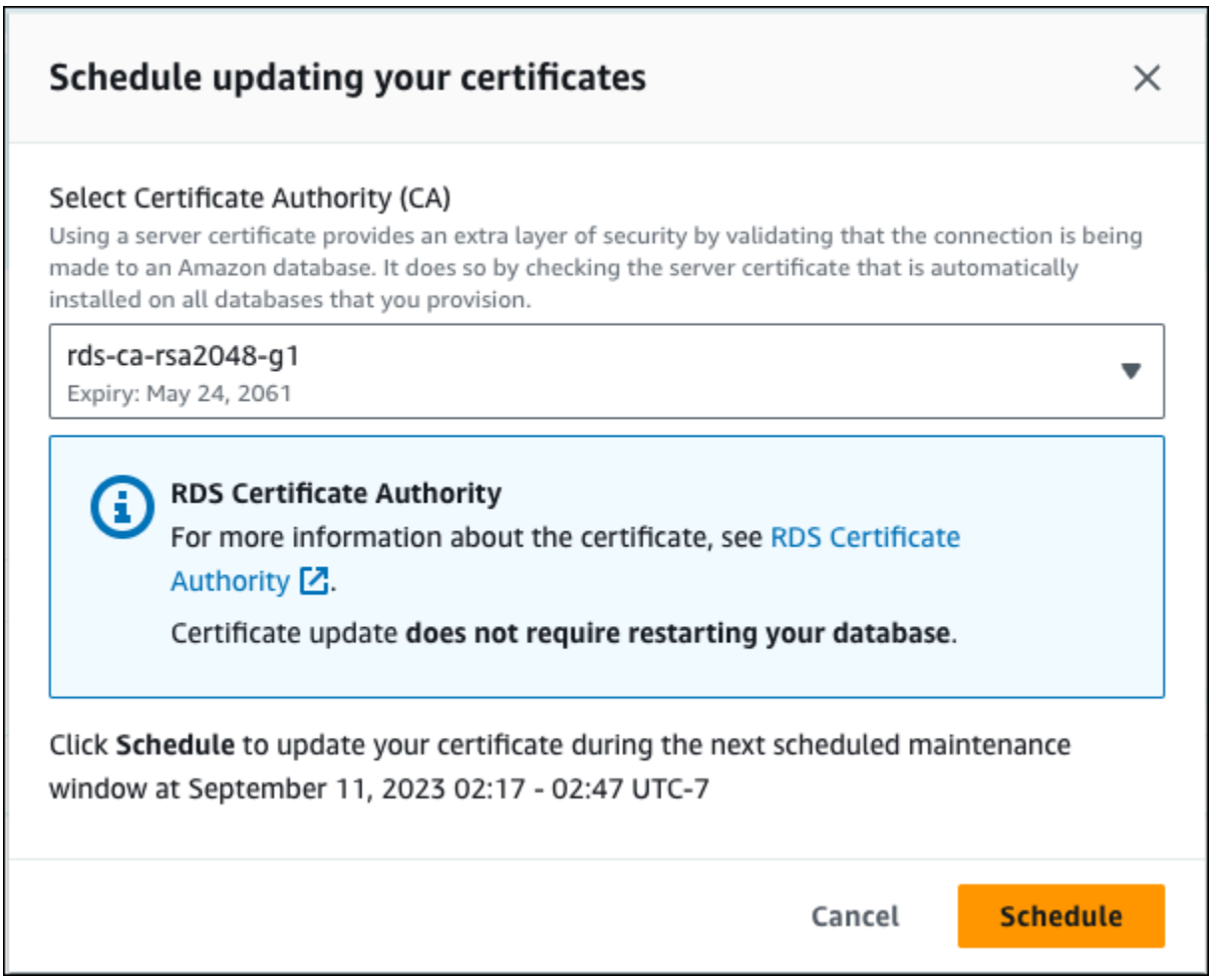
> **Note**
>
> This page only shows the DB instances and clusters for the current AWS Region. If you have databases in more than one AWS Region, check this page in each AWS Region to see all DB instances with old SSL/TLS certificates.

3. Choose the DB instance or Multi-AZ DB cluster that you want to update.

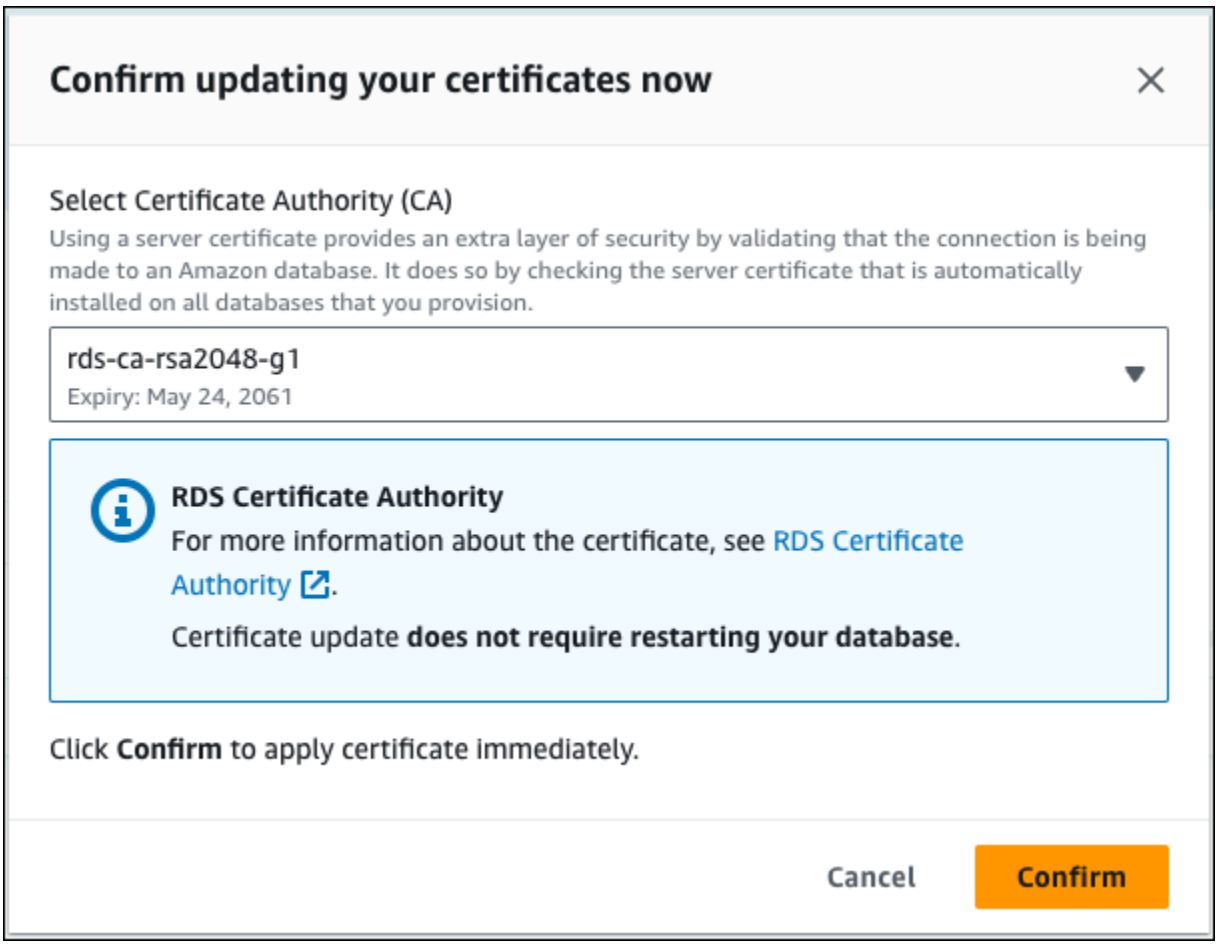
   You can schedule the certificate rotation for your next maintenance window by choosing **Schedule**. Apply the rotation immediately by choosing **Apply now**.

   > **Important**
   >
   > If you experience connectivity issues after certificate expiry, use the **Apply now** option.

4. a. If you choose **Schedule**, you are prompted to confirm the CA certificate rotation. This prompt also states the scheduled window for your update.



b. If you choose **Apply now**, you are prompted to confirm the CA certificate rotation.



> **Important**
>
> Before scheduling the CA certificate rotation on your database, update any client applications that use SSL/TLS and the server certificate to connect. These updates are specific to your DB engine. After you have updated these client applications, you can confirm the CA certificate rotation.

To continue, choose the check box, and then choose **Confirm**.

5. Repeat steps 3 and 4 for each DB instance and cluster that you want to update.

Automatic server certificate rotation

If your root CA supports automatic server certificate rotation, RDS automatically handles the rotation of the DB server certificate. RDS uses the same root CA for this automatic rotation, so you don't need to download a new CA bundle. See Certificate authorities .

The rotation and validity of your DB server certificate depend on your DB engine:

- If your DB engine supports rotation without restart, RDS automatically rotates the DB server certificate without requiring any action from you. RDS attempts to rotate your DB server certificate in your preferred maintenance window at the DB server certificate half life. The new DB server certificate is valid for 12 months.
- If your DB engine doesn't support rotation without restart, Amazon RDS makes a `server-certificate-rotation` Pending Maintenance Action visible via Describe-pending-maintenance-actions API, at the half life of the certificate, or at least 3 months before expiry. You can apply the rotation using the apply-pending-maintenance-action API. The new DB server certificate is valid for 36 months.

Use the describe-db-engine-versions command and inspect the `SupportsCertificateRotationWithoutRestart` flag to identify whether the DB engine version supports rotating the certificate without restart. For more information, see Setting the CA for your database .

Sample script for importing certificates into your trust store

The following are sample shell scripts that import the certificate bundle into a trust store.

Each sample shell script uses keytool, which is part of the Java Development Kit (JDK). For information about installing the JDK, see JDK Installation Guide.

Linux

The following is a sample shell script that imports the certificate bundle into a trust store on a Linux operating system.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
```

```
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/
 {split_after=1}{print > "rds-ca-" n+1 ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

macOS

The following is a sample shell script that imports the certificate bundle into a trust store on macOS.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi truststore=${mydir}/rds-truststore.jks storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem">
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-
```

```
for CERT in rds-ca-*; do alias=$(openssl x509 -noout -text -in $CERT | perl -ne
 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -
keystore ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias |
 cut -d " " -f3- | while read alias
do expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -
alias "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

Internetwork traffic privacy

Connections are protected both between Amazon RDS and on-premises applications and between Amazon RDS and other AWS resources within the same AWS Region.

Traffic between service and on-premises clients and applications

You have two connectivity options between your private network and AWS:

- An AWS Site-to-Site VPN connection. For more information, see What is AWS Site-to-Site VPN?
- An Direct Connect connection. For more information, see What is Direct Connect?

You get access to Amazon RDS through the network by using AWS-published API operations. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Identity and access management for Amazon RDS

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon RDS resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon RDS works with IAM
- Identity-based policy examples for Amazon RDS
- AWS managed policies for Amazon RDS
- Amazon RDS updates to AWS managed policies
- Preventing cross-service confused deputy problems
- IAM database authenticationfor MariaDB, MySQL, and PostgreSQL
- Troubleshooting Amazon RDS identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon RDS.

Service user – If you use the Amazon RDS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon RDS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon RDS, see Troubleshooting Amazon RDS identity and access.

Service administrator – If you're in charge of Amazon RDS resources at your company, you probably have full access to Amazon RDS. It's your job to determine which Amazon RDS features and resources your employees should access. You must then submit requests to your administrator

to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon RDS, see How Amazon RDS works with IAM.

Administrator – If you're an administrator, you might want to learn details about how you can write policies to manage access to Amazon RDS. To view example Amazon RDS identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon RDS.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

You can authenticate to your DB instance using IAM database authentication.

IAM database authentication works with the following DB engines:

- RDS for MariaDB
- RDS for MySQL
- RDS for PostgreSQL

For more information about authenticating to your DB instance using IAM, see IAM database authenticationfor MariaDB, MySQL, and PostgreSQL.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to a user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary user permissions** – A user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates

the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions** – Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

An administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (permission set or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as a permission set or role. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single permission set or role. Managed policies are standalone policies that you can attach to multiple permission sets and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

For information about AWS managed policies that are specific to Amazon RDS, see AWS managed policies for Amazon RDS.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (permission set or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the permission set or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the permission sets or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon RDS works with IAM

Before you use IAM to manage access to Amazon RDS, you should understand what IAM features are available to use with Amazon RDS.

The following table lists IAM features you can use with Amazon RDS:

IAM feature	Amazon RDS support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
Attribute-based access control (ABAC) (tags in policies)	Yes
Temporary credentials	Yes
Forward access sessions	Yes
Service roles	Yes
Service-linked roles	Yes

To get a high-level view of how Amazon RDS and other AWS services work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

Topics

- Amazon RDS identity-based policies
- Resource-based policies within Amazon RDS
- Policy actions for Amazon RDS
- Policy resources for Amazon RDS
- Policy condition keys for Amazon RDS
- Access control lists (ACLs) in Amazon RDS
- Attribute-based access control (ABAC) in policies with Amazon RDS tags

- Using temporary credentials with Amazon RDS
- Forward access sessions for Amazon RDS
- Service roles for Amazon RDS
- Service-linked roles for Amazon RDS

Amazon RDS identity-based policies

Supports identity-based policies: Yes.

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon RDS

To view examples of Amazon RDS identity-based policies, see Identity-based policy examples for Amazon RDS.

Resource-based policies within Amazon RDS

Supports resource-based policies: No.

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon RDS

Supports policy actions: Yes.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon RDS use the following prefix before the action: `rds:`. For example, to grant someone permission to describe DB instances with the Amazon RDS `DescribeDBInstances` API operation, you include the `rds:DescribeDBInstances` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon RDS defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [
      "rds:action1",
      "rds:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action.

```
"Action": "rds:Describe*"
```

To see a list of Amazon RDS actions, see Actions Defined by Amazon RDS in the *Service Authorization Reference*.

Policy resources for Amazon RDS

Supports policy resources: Yes.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't

support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

The DB instance resource has the following Amazon Resource Name (ARN).

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS service namespaces.

For example, to specify the `dbtest` DB instance in your statement, use the following ARN.

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

To specify all DB instances that belong to a specific account, use the wildcard (*).

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:*"
```

Some RDS API operations, such as those for creating resources, can't be performed on a specific resource. In those cases, use the wildcard (*).

```
"Resource": "*"
```

Many Amazon RDS API operations involve multiple resources. For example, `CreateDBInstance` creates a DB instance. You can specify that an user must use a specific security group and parameter group when creating a DB instance. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "resource1",
      "resource2"
```

To see a list of Amazon RDS resource types and their ARNs, see Resources Defined by Amazon RDS in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon RDS.

Policy condition keys for Amazon RDS

Supports service-specific policy condition keys: Yes.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Amazon RDS defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

All RDS API operations support the `aws:RequestedRegion` condition key.

To see a list of Amazon RDS condition keys, see Condition Keys for Amazon RDS in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon RDS.

Access control lists (ACLs) in Amazon RDS

Supports access control lists (ACLs): No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) in policies with Amazon RDS tags

Supports attribute-based access control (ABAC) tags in policies: Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For more information about tagging Amazon RDS resources, see Specifying conditions: Using custom tags. To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Grant permission for actions on a resource with a specific tag with two different values.

Using temporary credentials with Amazon RDS

Supports temporary credentials: Yes.

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Forward access sessions for Amazon RDS

Supports forward access sessions: Yes.

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon RDS

Supports service roles: Yes.

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Amazon RDS functionality. Edit service roles only when Amazon RDS provides guidance to do so.

Service-linked roles for Amazon RDS

Supports service-linked roles: Yes.

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about using Amazon RDS service-linked roles, see Using service-linked roles for Amazon RDS.

Identity-based policy examples for Amazon RDS

By default, permission sets and roles don't have permission to create or modify Amazon RDS resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An administrator must create IAM policies that grant permission sets and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the permission sets or roles that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

- Policy best practices
- Using the Amazon RDS console
- Permissions required to use the console
- Allow users to view their own permissions
- Permission policies to create, modify and, delete resources in Amazon RDS
- Example policies: Using condition keys

- Specifying conditions: Using custom tags
- Grant permission to tag Amazon RDS resources during creation

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon RDS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon RDS console

To access the Amazon RDS console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon RDS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

To ensure that those entities can still use the Amazon RDS console, also attach the following AWS managed policy to the entities.

```
AmazonRDSReadOnlyAccess
```

For more information, see Adding permissions to a user in the *IAM User Guide*.

Permissions required to use the console

For a user to work with the console, that user must have a minimum set of permissions. These permissions allow the user to describe the Amazon RDS resources for their AWS account and to provide other related information, including Amazon EC2 security and network information.

If you create an IAM policy that is more restrictive than the minimum required permissions, the console doesn't function as intended for users with that IAM policy. To ensure that those users can still use the console, also attach the `AmazonRDSReadOnlyAccess` managed policy to the user, as described in Managing access using policies.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the Amazon RDS API.

The following policy grants full access to all Amazon RDS resources for the root AWS account:

```
AmazonRDSFullAccess
```

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Permission policies to create, modify and, delete resources in Amazon RDS

The following sections present examples of permission policies that grant and restrict access to resources:

Allow a user to create DB instances in an AWS account

The following is an example policy that allows the account with the ID `123456789012` to create DB instances for your AWS account. The policy requires that the name of the new DB instance begin with `test`. The new DB instance must also use the MySQL database engine and the `db.t2.micro` DB instance class. In addition, the new DB instance must use an option group and a DB parameter group that starts with `default`, and it must use the `default` subnet group.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCreateDBInstanceOnly",
         "Effect": "Allow",
         "Action": [
            "rds:CreateDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:db:test*",
            "arn:aws:rds:*:123456789012:og:default*",
            "arn:aws:rds:*:123456789012:pg:default*",
            "arn:aws:rds:*:123456789012:subgrp:default"
         ],
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql",
               "rds:DatabaseClass": "db.t2.micro"
            }
         }
      }
   ]
}
```

The policy includes a single statement that specifies the following permissions for the user:

- The policy allows the account to create a DB instance using the CreateDBInstance API operation (this also applies to the create-db-instance AWS CLI command and the AWS Management Console).

- The `Resource` element specifies that the user can perform actions on or with resources. You specify resources using an Amazon Resources Name (ARN). This ARN includes the name of the service that the resource belongs to (`rds`), the AWS Region (`*` indicates any region in this example), the AWS account number (`123456789012` is the account number in this example), and the type of resource. For more information about creating ARNs, see Amazon Resource Names (ARNs) in Amazon RDS.

  The `Resource` element in the example specifies the following policy constraints on resources for the user:

  - The DB instance identifier for the new DB instance must begin with `test` (for example, `testCustomerData1`, `test-region2-data`).
  - The option group for the new DB instance must begin with `default`.
  - The DB parameter group for the new DB instance must begin with `default`.
  - The subnet group for the new DB instance must be the `default` subnet group.
- The `Condition` element specifies that the DB engine must be MySQL and the DB instance class must be `db.t2.micro`. The `Condition` element specifies the conditions when a policy should take effect. You can add additional permissions or restrictions by using the `Condition` element. For more information about specifying conditions, see Policy condition keys for Amazon RDS. This example specifies the `rds:DatabaseEngine` and `rds:DatabaseClass` conditions. For information about the valid condition values for `rds:DatabaseEngine`, see the list under the `Engine` parameter in CreateDBInstance. For information about the valid condition values for `rds:DatabaseClass`, see Supported DB engines for DB instance classes .

The policy doesn't specify the `Principal` element because in an identity-based policy you don't specify the principal who gets the permission. When you attach policy to a user, the user is the implicit principal. When you attach a permission policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

To see a list of Amazon RDS actions, see Actions Defined by Amazon RDS in the *Service Authorization Reference*.

Allow a user to perform any describe action on any RDS resource

The following permissions policy grants permissions to a user to run all of the actions that begin with `Describe`. These actions show information about an RDS resource, such as a DB instance. The wildcard character (*) in the `Resource` element indicates that the actions are allowed for all Amazon RDS resources owned by the account.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowRDSDescribe",
         "Effect": "Allow",
         "Action": "rds:Describe*",
         "Resource": "*"
      }
   ]
}
```

Allow a user to create a DB instance that uses the specified DB parameter group and subnet group

The following permissions policy grants permissions to allow a user to only create a DB instance that must use the `mydbpg` DB parameter group and the `mydbsubnetgroup` DB subnet group.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": [
            "arn:aws:rds:*:*:pg:mydbpg",
            "arn:aws:rds:*:*:subgrp:mydbsubnetgroup"
         ]
      }
   ]
}
```

Grant permission for actions on a resource with a specific tag with two different values

You can use conditions in your identity-based policy to control access to Amazon RDS resources based on tags. The following policy allows permission to perform the `CreateDBSnapshot` API operation on DB instances with either the `stage` tag set to `development` or `test`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

The following policy allows permission to perform the `ModifyDBInstance` API operation on DB instances with either the `stage` tag set to `development` or `test`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

Prevent a user from deleting a DB instance

The following permissions policy grants permissions to prevent a user from deleting a specific DB instance. For example, you might want to deny the ability to delete your production DB instances to any user that is not an administrator.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyDelete1",
         "Effect": "Deny",
         "Action": "rds:DeleteDBInstance",
         "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
      }
   ]
}
```

Deny all access to a resource

You can explicitly deny access to a resource. Deny policies take precedence over allow policies. The following policy explicitly denies a user the ability to manage a resource:

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": "rds:*",
         "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
      }
   ]
}
```

Example policies: Using condition keys

Following are examples of how you can use condition keys in Amazon RDS IAM permissions policies.

Example 1: Grant permission to create a DB instance that uses a specific DB engine and isn't MultiAZ

The following policy uses an RDS condition key and allows a user to create only DB instances that use the MySQL database engine and don't use MultiAZ. The `Condition` element indicates the requirement that the database engine is MySQL.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AllowMySQLCreate",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql"
            },
            "Bool": {
               "rds:MultiAz": false
            }
         }
      }
   ]
}
```

Example 2: Explicitly deny permission to create DB instances for certain DB instance classes and create DB instances that use Provisioned IOPS

The following policy explicitly denies permission to create DB instances that use the DB instance classes `r3.8xlarge` and `m4.10xlarge`, which are the largest and most expensive DB instance

classes. This policy also prevents users from creating DB instances that use Provisioned IOPS, which incurs an additional cost.

Explicitly denying permission supersedes any other permissions granted. This ensures that identities to not accidentally get permission that you never want to grant.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "DenyLargeCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseClass": [
                  "db.r3.8xlarge",
                  "db.m4.10xlarge"
               ]
            }
         }
      },
      {
         "Sid": "DenyPIOPSCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "NumericNotEquals": {
               "rds:Piops": "0"
            }
         }
      }
   ]
}
```

Example 3: Limit the set of tag keys and values that can be used to tag a resource

The following policy uses an RDS condition key and allows the addition of a tag with the key `stage` to be added to a resource with the values `test`, `qa`, and `production`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagEdits",
      "Effect": "Allow",
      "Action": [
        "rds:AddTagsToResource",
        "rds:RemoveTagsFromResource"
      ],
      "Resource": "arn:aws:rds:us-east-1:123456789012:db:db-123456",
      "Condition": {
        "StringEquals": {
          "rds:req-tag/stage": [
            "test",
            "qa",
            "production"
          ]
        }
      }
    }
  ]
}
```

Specifying conditions: Using custom tags

Amazon RDS supports specifying conditions in an IAM policy using custom tags.

For example, suppose that you add a tag named `environment` to your DB instances with values such as `beta`, `staging`, `production`, and so on. If you do, you can create a policy that restricts certain users to DB instances based on the `environment` tag value.

> **Note**
>
> Custom tag identifiers are case-sensitive.

The following table lists the RDS tag identifiers that you can use in a `Condition` element.

RDS tag identifier	Applies to
`db-tag`	DB instances, including read replicas
`snapshot-tag`	DB snapshots
`ri-tag`	Reserved DB instances
`og-tag`	DB option groups
`pg-tag`	DB parameter groups
`subgrp-tag`	DB subnet groups
`es-tag`	Event subscriptions
`cluster-tag`	DB clusters
`cluster-pg-tag`	DB cluster parameter groups
`cluster-snapshot-tag`	DB cluster snapshots

The syntax for a custom tag condition is as follows:

```
"Condition":{"StringEquals":{"rds:rds-tag-identifier/tag-name":
["value"]} }
```

For example, the following `Condition` element applies to DB instances with a tag named `environment` and a tag value of `production`.

```
"Condition":{"StringEquals":{"rds:db-tag/environment": ["production"]} }
```

For information about creating tags, see Tagging Amazon RDS resources.

Important

If you manage access to your RDS resources using tagging, we recommend that you secure access to the tags for your RDS resources. You can manage access to tags by creating policies for the `AddTagsToResource` and `RemoveTagsFromResource` actions. For example, the following policy denies users the ability to add or remove tags for all resources. You can then create policies to allow specific users to add or remove tags.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyTagUpdates",
         "Effect":"Deny",
         "Action":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"*"
      }
   ]
}
```

To see a list of Amazon RDS actions, see Actions Defined by Amazon RDS in the *Service Authorization Reference*.

Example policies: Using custom tags

Following are examples of how you can use custom tags in Amazon RDS IAM permissions policies. For more information about adding tags to an Amazon RDS resource, see Amazon Resource Names (ARNs) in Amazon RDS.

Note

All examples use the us-west-2 region and contain fictitious account IDs.

Example 1: Grant permission for actions on a resource with a specific tag with two different values

The following policy allows permission to perform the `CreateDBSnapshot` API operation on DB instances with either the `stage` tag set to `development` or `test`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

The following policy allows permission to perform the `ModifyDBInstance` API operation on DB instances with either the `stage` tag set to `development` or `test`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
            ],
          "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
            ]
       },
       {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
            ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
               "rds:db-tag/stage":[
                  "development",
                  "test"
                  ]
               }
            }
       }
    ]
}
```

Example 2: Explicitly deny permission to create a DB instance that uses specified DB parameter groups

The following policy explicitly denies permission to create a DB instance that uses DB parameter groups with specific tag values. You might apply this policy if you require that a specific customer-created DB parameter group always be used when creating DB instances. Policies that use `Deny` are most often used to restrict access that was granted by a broader policy.

Explicitly denying permission supersedes any other permissions granted. This ensures that identities to not accidentally get permission that you never want to grant.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyProductionCreate",
         "Effect":"Deny",
         "Action":"rds:CreateDBInstance",
         "Resource":"arn:aws:rds:*:123456789012:pg:*",
         "Condition":{
            "StringEquals":{
               "rds:pg-tag/usage":"prod"
            }
         }
      }
   ]
}
```

Example 3: Grant permission for actions on a DB instance with an instance name that is prefixed with a user name

The following policy allows permission to call any API (except to `AddTagsToResource` or `RemoveTagsFromResource`) on a DB instance that has a DB instance name that is prefixed with the user's name and that has a tag called `stage` equal to `devo` or that has no tag called `stage`.

The `Resource` line in the policy identifies a resource by its Amazon Resource Name (ARN). For more information about using ARNs with Amazon RDS resources, see Amazon Resource Names (ARNs) in Amazon RDS.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowFullDevAccessNoTags",
         "Effect":"Allow",
         "NotAction":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:${aws:username}*",
         "Condition":{
            "StringEqualsIfExists":{
               "rds:db-tag/stage":"devo"
            }
         }
      }
   ]
}
```

Grant permission to tag Amazon RDS resources during creation

Some RDS API operations allow you to specify tags when you create the resource. You can use resource tags to implement attribute-based control (ABAC). For more information, see What is ABAC for AWS? and Controlling access to AWS resources using tags.

To enable users to tag resources on creation, they must have permissions to use the action that creates the resource, such as `rds:CreateDBInstance`. If tags are specified in the create action, RDS performs additional authorization on the `rds:AddTagsToResource` action to verify if users have permissions to create tags. Therefore, users must also have explicit permissions to use the `rds:AddTagsToResource` action.

In the IAM policy definition for the `rds:AddTagsToResource` action, you can use the `aws:RequestTag` condition key to require tags in a request to tag a resource.

For example, the following policy allows users to create DB instances and apply tags during DB instance creation, but only with specific tag keys (`environment` or `project`):

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBInstance"
           ],
           "Resource": "*"
       },
       {
           "Effect": "Allow",
           "Action": [
               "rds:AddTagsToResource"
           ],
           "Resource": "*",
           "Condition": {
               "StringEquals": {
                   "aws:RequestTag/environment": ["production", "development"],
                   "aws:RequestTag/project": ["dataanalytics", "webapp"]
               },
               "ForAllValues:StringEquals": {
                   "aws:TagKeys": ["environment", "project"]
               }
           }
       }
   ]
}
```

This policy denies any create DB instance request that includes tags other than the `environment` or `project` tags, or that doesn't specify either of these tags. Additionally, users must specify values for the tags that match the allowed values in the policy.

The following policy allows users to create DB clusters and apply any tags during creation except the `environment=prod` tag:

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "rds:CreateDBCluster"
           ],
           "Resource": "*"
       },
       {
           "Effect": "Allow",
           "Action": [
               "rds:AddTagsToResource"
           ],
           "Resource": "*",
           "Condition": {
               "StringNotEquals": {
                   "aws:RequestTag/environment": "prod"
               }
           }
       }
   ]
}
```

Supported RDS API actions for tagging on creation

The following RDS API actions support tagging when you create a resource. For these actions, you can specify tags when you create the resource:

- `CreateBlueGreenDeployment`
- `CreateCustomDBEngineVersion`
- `CreateDBCluster`
- `CreateDBClusterEndpoint`
- `CreateDBClusterParameterGroup`
- `CreateDBClusterSnapshot`
- `CreateDBInstance`

- CreateDBInstanceReadReplica
- CreateDBParameterGroup
- CreateDBProxy
- CreateDBProxyEndpoint
- CreateDBSecurityGroup
- CreateDBShardGroup
- CreateDBSnapshot
- CreateDBSubnetGroup
- CreateEventSubscription
- CreateGlobalCluster
- CreateIntegration
- CreateOptionGroup
- CreateTenantDatabase
- CopyDBClusterParameterGroup
- CopyDBClusterSnapshot
- CopyDBParameterGroup
- CopyDBSnapshot
- CopyOptionGroup
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- RestoreDBInstanceFromDBSnapshot
- RestoreDBInstanceFromS3
- RestoreDBInstanceToPointInTime
- PurchaseReservedDBInstancesOffering

If you use the AWS CLI or API to create a resource with tags, the `Tags` parameter is used to apply tags to resources during creation.

For these API actions, if tagging fails, the resource is not created, and the request fails with an error. This ensures that resources are either created with tags or not created at all, preventing resources from being created without the intended tags.

AWS managed policies for Amazon RDS

To add permissions to permission sets and roles, it's easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (permission sets and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services don't remove permissions from an AWS managed policy, so policy updates don't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ReadOnlyAccess` AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

Topics

- AWS managed policy: AmazonRDSReadOnlyAccess
- AWS managed policy: AmazonRDSFullAccess
- AWS managed policy: AmazonRDSDataFullAccess
- AWS managed policy: AmazonRDSEnhancedMonitoringRole
- AWS managed policy: AmazonRDSPerformanceInsightsReadOnly
- AWS managed policy: AmazonRDSPerformanceInsightsFullAccess
- AWS managed policy: AmazonRDSDirectoryServiceAccess
- AWS managed policy: AmazonRDSServiceRolePolicy
- AWS managed policy: AmazonRDSCustomServiceRolePolicy
- AWS managed policy: AmazonRDSCustomInstanceProfileRolePolicy
- AWS managed policy: AmazonRDSPreviewServiceRolePolicy
- AWS managed policy: AmazonRDSBetaServiceRolePolicy

AWS managed policy: AmazonRDSReadOnlyAccess

This policy allows read-only access to Amazon RDS through the AWS Management Console.

Permissions details

This policy includes the following permissions:

- `rds` – Allows principals to describe Amazon RDS resources and list the tags for Amazon RDS resources.
- `cloudwatch` – Allows principals to get Amazon CloudWatch metric statistics.
- `ec2` – Allows principals to describe Availability Zones and networking resources.
- `logs` – Allows principals to describe CloudWatch Logs log streams of log groups, and get CloudWatch Logs log events.
- `devops-guru` – Allows principals to describe resources that have Amazon DevOps Guru coverage, which is specified either by CloudFormation stack names or resource tags.

For more information about this policy, including the JSON policy document, see AmazonRDSReadOnlyAccess in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSFullAccess

This policy provides full access to Amazon RDS through the AWS Management Console.

Permissions details

This policy includes the following permissions:

- `rds` – Allows principals full access to Amazon RDS.
- `application-autoscaling` – Allows principals describe and manage Application Auto Scaling scaling targets and policies.
- `cloudwatch` – Allows principals get CloudWatch metric statics and manage CloudWatch alarms.
- `ec2` – Allows principals to describe Availability Zones and networking resources.
- `logs` – Allows principals to describe CloudWatch Logs log streams of log groups, and get CloudWatch Logs log events.
- `outposts` – Allows principals to get AWS Outposts instance types.

- `pi` – Allows principals to get Performance Insights metrics.
- `sns` – Allows principals to Amazon Simple Notification Service (Amazon SNS) subscriptions and topics, and to publish Amazon SNS messages.
- `devops-guru` – Allows principals to describe resources that have Amazon DevOps Guru coverage, which is specified either by CloudFormation stack names or resource tags.

For more information about this policy, including the JSON policy document, see AmazonRDSFullAccess in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSDataFullAccess

This policy allows full access to use the Data API and the query editor on Aurora Serverless clusters in a specific AWS account. This policy allows the AWS account to get the value of a secret from AWS Secrets Manager.

You can attach the `AmazonRDSDataFullAccess` policy to your IAM identities.

Permissions details

This policy includes the following permissions:

- `dbqms` – Allows principals to access, create, delete, describe, and update queries. The Database Query Metadata Service (`dbqms`) is an internal-only service. It provides your recent and saved queries for the query editor on the AWS Management Console for multiple AWS services, including Amazon RDS.
- `rds-data` – Allows principals to run SQL statements on Aurora Serverless databases.
- `secretsmanager` – Allows principals to get the value of a secret from AWS Secrets Manager.

For more information about this policy, including the JSON policy document, see AmazonRDSDataFullAccess in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSEnhancedMonitoringRole

This policy provides access to Amazon CloudWatch Logs for Amazon RDS Enhanced Monitoring.

Permissions details

This policy includes the following permissions:

- `logs` – Allows principals to create CloudWatch Logs log groups and retention policies, and to create and describe CloudWatch Logs log streams of log groups. It also allows principals to put and get CloudWatch Logs log events.

For more information about this policy, including the JSON policy document, see AmazonRDSEnhancedMonitoringRole in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSPerformanceInsightsReadOnly

This policy provides read-only access to Amazon RDS Performance Insights for Amazon RDS DB instances and Amazon Aurora DB clusters.

This policy now includes `Sid` (statement ID) as an identifier for the policy statement.

Permissions details

This policy includes the following permissions:

- `rds` – Allows principals to describe Amazon RDS DB instances and Amazon Aurora DB clusters.
- `pi` – Allows principals make calls to the Amazon RDS Performance Insights API and access Performance Insights metrics.

For more information about this policy, including the JSON policy document, see AmazonRDSPerformanceInsightsReadOnly in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSPerformanceInsightsFullAccess

This policy provides full access to Amazon RDS Performance Insights for Amazon RDS DB instances and Amazon Aurora DB clusters.

This policy now includes `Sid` (statement ID) as an identifier for the policy statement.

Permissions details

This policy includes the following permissions:

- `rds` – Allows principals to describe Amazon RDS DB instances and Amazon Aurora DB clusters.
- `pi` – Allows principals make calls to the Amazon RDS Performance Insights API, and create, view, and delete performance analysis reports.

- `cloudwatch` – Allows principals to list all the Amazon CloudWatch metrics, and get metric data and statistics.

For more information about this policy, including the JSON policy document, see AmazonRDSPerformanceInsightsFullAccess in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSDirectoryServiceAccess

This policy allows Amazon RDS to make calls to the Directory Service.

Permissions details

This policy includes the following permission:

- `ds` – Allows principals to describe Directory Service directories and control authorization to Directory Service directories.

For more information about this policy, including the JSON policy document, see AmazonRDSDirectoryServiceAccess in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSServiceRolePolicy

You can't attach the `AmazonRDSServiceRolePolicy` policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon RDS to perform actions on your behalf. For more information, see Service-linked role permissions for Amazon RDS.

AWS managed policy: AmazonRDSCustomServiceRolePolicy

You can't attach the `AmazonRDSCustomServiceRolePolicy` policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon RDS to call AWS services on behalf of your RDS DB resources.

This policy includes the following permissions:

- `ec2` - Allows RDS Custom to perform backup operations on the DB instance that provides point-in-time restore capabilities.
- `secretsmanager` - Allows RDS Custom to manage DB instance specific secrets created by RDS Custom.

- `cloudwatch` - Allows RDS Custom to upload DB instance metrics and logs to CloudWatch through CloudWatch agent.
- `events`, `sqs` - Allows RDS Custom to send and receive status information about the DB instance.
- `cloudtrail` - Allows RDS Custom to recieve change events about the DB instance
- `servicequotas` - Allows RDS Custom to read service quotas related to the DB instance
- `ssm` - Allows RDS Custom to manage the DB instance's underlying EC2 instance.
- `rds` - Allows RDS Custom to manage RDS resources for your DB instance
- `iam` - Allows RDS Custom to validate and attach the instance profile to a DB instance's underlying EC2 instance.

For more information about this policy, including the JSON policy document, see AmazonRDSCustomServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSCustomInstanceProfileRolePolicy

You shouldn't attach `AmazonRDSCustomInstanceProfileRolePolicy` to your IAM entities. It should only be attached to an instance profile role that is used to grant permissions to your Amazon RDS Custom DB instance to perform various automation actions and database management tasks. Pass the instance profile as `custom-iam-instance-profile` parameter during the RDS Custom instance creation and RDS Custom associates this instance profile to your DB instance.

Permissions details

This policy includes the following permissions:

- `ssm`, `ssmmessages`, `ec2messages` - Allows RDS Custom to communicate, execute automation and maintain agents on the DB instance through Systems Manager.
- `ec2`, `s3` - Allows RDS Custom to perform backup operations on the DB instance that provides point-in-time restore capabilities.
- `secretsmanager` - Allows RDS Custom to manage DB instance specific secrets created by RDS Custom.
- `cloudwatch`, `logs` - Allows RDS Custom to upload DB instance metrics and logs to CloudWatch through CloudWatch agent.
- `events`, `sqs` - Allows RDS Custom to send and receive status information about the DB instance.

- `kms` - Allows RDS Custom to use an instance-specific KMS key to perform encryption of secrets and S3 objects that RDS Custom manages.

For more information about this policy, including the JSON policy document, see AmazonRDSCustomInstanceProfileRolePolicy in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSPreviewServiceRolePolicy

You shouldn't attach `AmazonRDSPreviewServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Amazon RDS to call AWS services on behalf of your RDS DB resources. For more information, see Service-linked role for Amazon RDS Preview.

Permissions details

This policy includes the following permissions:

- `ec2` - Allows principals to describe Availability Zones and networking resources.
- `secretsmanager` – Allows principals to get the value of a secret from AWS Secrets Manager.
- `cloudwatch`, `logs` - Allows Amazon RDS to upload DB instance metrics and logs to CloudWatch through CloudWatch agent.

For more information about this policy, including the JSON policy document, see AmazonRDSPreviewServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

AWS managed policy: AmazonRDSBetaServiceRolePolicy

You shouldn't attach `AmazonRDSBetaServiceRolePolicy` to your IAM entities. This policy is attached to a service-linked role that allows Amazon RDS to call AWS services on behalf of your RDS DB resources. For more information, see Service-linked role permissions for Amazon RDS Beta.

Permissions details

This policy includes the following permissions:

- `ec2` - Allows Amazon RDS to perform backup operations on the DB instance that provides point-in-time restore capabilities.
- `secretsmanager` - Allows Amazon RDS to manage DB instance specific secrets created by Amazon RDS.

- `cloudwatch`, `logs` - Allows Amazon RDS to upload DB instance metrics and logs to CloudWatch through CloudWatch agent.

For more information about this policy, including the JSON policy document, see AmazonRDSBetaServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

Amazon RDS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon RDS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon RDS Document history page.

Change	Description	Date
Service-linked role permissions for Amazon RDS Custom – Update to existing policy	Amazon RDS updated permissions on the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. The update removes `ec2:CopySnapshot` from one statement and adds two new statements for source and destination snapshot permissions. These updates comply with a Change in EBS CopySnapshot authorization behavior while keeping effective permissions unchanged. For more information, see Service-linked role permissions for Amazon RDS Custom.	August 7, 2025
Service-linked role permissions for Amazon RDS Custom – Update to existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. These permissions allow	March 25, 2025

Change	Description	Date
	RDS Custom to manage EC2 key-pairs and allow RDS Custom to integrate with Amazon SQS. For more information, see Service-linked role permissions for Amazon RDS Custom.	
AWS managed policy: AmazonRDSCustomInstanceProfileRolePolicy – Update to existing policy	Amazon RDS added new permissions to the managed policy `AmazonRDSCustomInstanceProfileRolePolicy` to allow the usage of RDS Custom managed secrets on an RDS Custom instance. For more information, see AWS managed policy: AmazonRDSCustomInstanceProfileRolePolicy.	March 20, 2025
Service-linked role permissions for Amazon RDS Custom – Update to existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. These new permissions allow RDS Custom to list and restore Secrets Manager secrets. For more information, see Service-linked role permissions for Amazon RDS Custom.	March 6, 2025

Change	Description	Date
AWS managed policy: AmazonRDSPreviewServiceRolePolicy – Update to existing policy	Amazon RDS removed `sns:Publish` permission from the `AmazonRDSPreviewServiceRolePolicy` of the `AWSServiceRoleForRDSPreview` service-linked role. For more information, see AWS managed policy: AmazonRDSPreviewServiceRolePolicy.	August 7, 2024
AWS managed policy: AmazonRDSBetaServiceRolePolicy – Update to existing policy	Amazon RDS removed `sns:Publish` permission from the `AmazonRDSBetaServiceRolePolicy` of the `AWSServiceRoleForRDSBeta` service-linked role. For more information, see AWS managed policy: AmazonRDSBetaServiceRolePolicy.	August 7, 2024

Change	Description	Date
Service-linked role permissions for Amazon RDS Custom – Update to existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. The permissions allow RDS Custom to communicate with Amazon RDS services in another AWS Region and copy EC2 images. For more information, see Service-linked role permissions for Amazon RDS Custom.	July 18, 2024
AWS managed policy: AmazonRDSServiceRolePolicy – Update to existing policy	Amazon RDS removed `sns:Publish` permission from the `AmazonRDSServiceRolePolicy` of the `AWSServiceRoleForRDS` service-linked role. For more information, see AWS managed policy: AmazonRDSServiceRolePolicy.	July 2, 2024

Change	Description	Date
Service-linked role permissions for Amazon RDS Custom – Update to existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. This new permission allow RDS Custom to associate a service-role as an instance profile to an RDS Custom instance. For more information, see Service-linked role permissions for Amazon RDS Custom.	April 19, 2024
AWS managed policies for Amazon RDS – Update to existing policy	Amazon RDS added a new permission to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role to allow RDS Custom for SQL Server to modify the underlying database host instance type. RDS also added the `ec2:DescribeInstanceTypes` permission to get instance type information for database host. For more information, see AWS managed policies for Amazon RDS.	April 8, 2024

Change	Description	Date
AWS managed policies for Amazon RDS – New policy	Amazon RDS added a new managed policy named `AmazonRDSCustomInstanceProfileRolePolicy` to allow RDS Custom to perform automation actions and database management tasks through an EC2 instance profile. For more information, see AWS managed policies for Amazon RDS.	February 27, 2024
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new statement IDs to the `AmazonRDSServiceRolePolicy` of the `AWSServiceRoleForRDS` service-linked role. For more information, see Service-linked role permissions for Amazon RDS.	January 19, 2024

Change	Description	Date
AWS managed policies for Amazon RDS – Update to existing policies	The `AmazonRDSPerformanceInsightsReadOnly` and `AmazonRDSPerformanceInsightsFullAccess` managed policies now includes `Sid` (statement ID) as an identifier in the policy statement. For more information, see AWS managed policy: AmazonRDSPerformanceInsightsReadOnly and AWS managed policy: AmazonRDSPerformanceInsightsFullAccess	October 23, 2023
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. These new permissions allow RDS Custom for Oracle to create, modify, and delete EventBridge Managed Rules. For more information, see Service-linked role permissions for Amazon RDS Custom.	September 20, 2023

Change	Description	Date
AWS managed policies for Amazon RDS – Update to existing policy	Amazon RDS added new permissions to `AmazonRDSFullAccess` managed policy. The permissions allow you to generate, view, and delete the performance analysis report for a time period. For more information about configuring access policies for Performance Insights, see Configuring access policies for Performance Insights	August 17, 2023
AWS managed policies for Amazon RDS – New policy and update to existing policy	Amazon RDS added new permissions to `AmazonRDSPerformanceInsightsReadOnly` managed policy and a new managed policy named `AmazonRDSPerformanceInsightsFullAccess`. These permissions allow you to analyse the Performance Insights for a time period, view the analysis results along with the recommendations, and delete the reports. For more information about configuring access policies for Performance Insights, see Configuring access policies for Performance Insights	August 16, 2023

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. These new permissions allow RDS Custom for Oracle to use DB snapshots. For more information, see Service-linked role permissions for Amazon RDS Custom.	June 23, 2023
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. These new permissions allow RDS Custom for Oracle to use DB snapshots. For more information, see Service-linked role permissions for Amazon RDS Custom.	June 23, 2023

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. These new permissions allow RDS Custom to create network interfaces. For more information, see Service-linked role permissions for Amazon RDS Custom.	May 30, 2023
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. These new permissions allow RDS Custom to call Amazon EBS to check the storage quota. For more information, see Service-linked role permissions for Amazon RDS Custom.	April 18, 2023

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS Custom added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role for integration with Amazon SQS. RDS Custom requires integration with Amazon SQS to create and manage SQS queues in the customer account. The SQS queue names follow the format `do-not-delete-rds-custom-[identifier]` and are tagged with `Amazon RDS Custom`. The permission for `ec2:CreateSnapshot` was also added to allow RDS Custom to create backups for volumes attached to the instance. For more information, see Service-linked role permissions for Amazon RDS Custom.	April 6, 2023

Change	Description	Date
AWS managed policies for Amazon RDS – Update to an existing policy	Amazon RDS added a new Amazon CloudWatch namespace `ListMetrics` to `AmazonRDSFullAccess` and `AmazonRDSReadOnlyAccess`. This namespace is required for Amazon RDS to list specific resource usage metrics. For more information, see Overview of managing access permissions to your CloudWatch resources in the *Amazon CloudWatch User Guide*.	April 4, 2023
AWS managed policies for Amazon RDS – Update to an existing policy	Amazon RDS added a new permission to `AmazonRDSFullAccess` and `AmazonRDSReadOnlyAccess` managed policies to allow the display of Amazon DevOps Guru findings in the RDS console. This permission is required to allow the display of DevOps Guru findings. For more information, see Amazon RDS updates to AWS managed policies.	March, 30 2023

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AmazonRDSServiceRolePolicy` of the `AWSServiceRoleForRDS` service-linked role for integration with AWS Secrets Manager. RDS requires integration with Secrets Manager for managing master user passwords in Secrets Manager. The secret uses a reserved naming convention and restricts customer updates. For more information, see Password management with Amazon RDS and AWS Secrets Manager.	December 22, 2022

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role. RDS Custom supports DB clusters. These new permissions in the policy allow RDS Custom to call AWS services on behalf of your DB clusters. For more information, see Service-linked role permissions for Amazon RDS Custom.	November 9, 2022

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AWSServiceRoleForRDS` service-linked role for integration with AWS Secrets Manager. Integration with Secrets Manager is required for SQL Server Reporting Services (SSRS) Email to function on RDS. SSRS Email creates a secret on behalf of the customer. The secret uses a reserved naming convention and restricts customer updates. For more information, see Using SSRS Email to send reports.	August 26, 2022

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added a new Amazon CloudWatch namespace to `AmazonRDSPreviewServiceRolePolicy` for `PutMetricData`. This namespace is required for Amazon RDS to publish resource usage metrics. For more information, see Using condition keys to limit access to CloudWatch namespaces in the *Amazon CloudWatch User Guide*.	June 7, 2022
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added a new Amazon CloudWatch namespace to `AmazonRDSBetaServiceRolePolicy` for `PutMetricData`. This namespace is required for Amazon RDS to publish resource usage metrics. For more information, see Using condition keys to limit access to CloudWatch namespaces in the *Amazon CloudWatch User Guide*.	June 7, 2022

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added a new Amazon CloudWatch namespace to `AWSServiceRoleForRDS` for `PutMetricData` . This namespace is required for Amazon RDS to publish resource usage metrics. For more information, see Using condition keys to limit access to CloudWatch namespaces in the *Amazon CloudWatch User Guide*.	April 22, 2022
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new permissions to the `AWSServiceRoleForRDS` service-linked role to manage permissions for customer-owned IP pools and local gateway route tables (LGW-RTBs). These permissions are required for RDS on Outposts to perform Multi-AZ replication across the Outposts' local network. For more information, see Working with Multi-AZ deployments for Amazon RDS on AWS Outposts.	April 19, 2022

Change	Description	Date
Identity-based policies – Update to an existing policy	Amazon RDS added a new permission to the `AmazonRDSFullAccess` managed policy to describe permissions on LGW-RTBs. This permission is required to describe permissions for RDS on Outposts to perform Multi-AZ replication across the Outposts' local network. For more information, see Working with Multi-AZ deployments for Amazon RDS on AWS Outposts.	April 19, 2022
AWS managed policies for Amazon RDS – New policy	Amazon RDS added a new managed policy named `AmazonRDSPerformanceInsightsReadOnly` to allow Amazon RDS to call AWS services on behalf of your DB instances. For more information about configuring access policies for Performance Insights, see Configuring access policies for Performance Insights	March 10, 2022

Change	Description	Date
Service-linked role permissions for Amazon RDS – Update to an existing policy	Amazon RDS added new Amazon CloudWatch namespaces to `AWSServiceRoleForRDS` for `PutMetricData` . These namespaces are required for Amazon DocumentDB (with MongoDB compatibility) and Amazon Neptune to publish CloudWatch metrics. For more information, see Using condition keys to limit access to CloudWatch namespaces in the *Amazon CloudWatch User Guide*.	March 4, 2022
Service-linked role permissions for Amazon RDS Custom – New policy	Amazon RDS added a new service-linked role named `AWSServiceRoleForRDSCustom` to allow RDS Custom to call AWS services on behalf of your DB instances.	October 26, 2021
Amazon RDS started tracking changes	Amazon RDS started tracking changes for its AWS managed policies.	October 26, 2021

Preventing cross-service confused deputy problems

The *confused deputy problem* is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem.

Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way that it shouldn't have permission to access. To prevent this, AWS provides tools that can help you protect your data for all services with service principals that have been given access to resources in your account. For more information, see The confused deputy problem in the *IAM User Guide*.

To limit the permissions that Amazon RDS gives another service for a specific resource, we recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies.

In some cases, the `aws:SourceArn` value doesn't contain the account ID, for example when you use the Amazon Resource Name (ARN) for an Amazon S3 bucket. In these cases, make sure to use both global condition context keys to limit permissions. In some cases, you use both global condition context keys and the `aws:SourceArn` value contains the account ID. In these cases, make sure that the `aws:SourceAccount` value and the account in the `aws:SourceArn` use the same account ID when they're used in the same policy statement. If you want only one resource to be associated with the cross-service access, use `aws:SourceArn`. If you want to allow any resource in the specified AWS account to be associated with the cross-service use, use `aws:SourceAccount`.

Make sure that the value of `aws:SourceArn` is an ARN for an Amazon RDS resource type. For more information, see Amazon Resource Names (ARNs) in Amazon RDS.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. In some cases, you might not know the full ARN of the resource or you might be specifying multiple resources. In these cases, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. An example is `arn:aws:rds:*:123456789012:*`.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Amazon RDS to prevent the confused deputy problem.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "rds.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

For more examples of policies that use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys, see the following sections:

- Granting permissions to publish notifications to an Amazon SNS topic
- Manually creating an IAM role for native backup and restore
- Setting up Windows Authentication for SQL Server DB instances
- Prerequisites for integrating RDS for SQL Server with S3
- Manually creating an IAM role for SQL Server Audit
- Configuring IAM permissions for RDS for Oracle integration with Amazon S3
- Setting up access to an Amazon S3 bucket (PostgreSQL import)
- Setting up access to an Amazon S3 bucket (PostgreSQL export)

IAM database authenticationfor MariaDB, MySQL, and PostgreSQL

You can authenticate to your DB instance using AWS Identity and Access Management (IAM) database authentication. IAM database authentication works with MariaDB, MySQL, and PostgreSQL. With this authentication method, you don't need to use a password when you connect to a DB instance. Instead, you use an authentication token.

An *authentication token* is a unique string of characters that Amazon RDS generates on request. Authentication tokens are generated using AWS Signature Version 4. Each token has a lifetime of 15 minutes. You don't need to store user credentials in the database, because authentication is managed externally using IAM. You can also still use standard database authentication. The token is only used for authentication and doesn't affect the session after it is established.

IAM database authentication provides the following benefits:

- Network traffic to and from the database is encrypted using Secure Socket Layer (SSL) or Transport Layer Security (TLS). For more information about using SSL/TLS with Amazon RDS, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
- You can use IAM to centrally manage access to your database resources, instead of managing access individually on each DB instance.
- For applications running on Amazon EC2, you can use profile credentials specific to your EC2 instance to access your database instead of a password, for greater security.

In general, consider using IAM database authentication when your applications create fewer than 200 connections per second, and you don't want to manage usernames and passwords directly in your application code.

The Amazon Web Services (AWS) JDBC Driver supports IAM database authentication. For more information, see AWS IAM Authentication Plugin in the Amazon Web Services (AWS) JDBC Driver GitHub repository.

The Amazon Web Services (AWS) Python Driver supports IAM database authentication. For more information, see AWS IAM Authentication Plugin in the Amazon Web Services (AWS) Python Driver GitHub repository.

Navigate through the following topics to learn the process to set IAM for DB authentication:

- Enabling and disabling IAM database authentication

- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication
- Connecting to your DB instance using IAM authentication

Region and version availability

Feature availability and support varies across specific versions of each database engine. For more information on engine, version, and Region availability with Amazon RDS and IAM database authentication, see Supported Regions and DB engines for IAM database authentication in Amazon RDS.

CLI and SDK support

IAM database authentication is available for the AWS CLI and for the following language-specific AWS SDKs:

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto3)
- AWS SDK for Ruby

Limitations for IAM database authentication

When using IAM database authentication, the following limitations apply:

- Currently, IAM database authentication doesn't support all global condition context keys.

  For more information about global condition context keys, see AWS global condition context keys in the *IAM User Guide*.
- For PostgreSQL, if the IAM role (`rds_iam`) is added to a user (including the RDS master user), IAM authentication takes precedence over password authentication, so the user must log in as an IAM user.

- For PostgreSQL, Amazon RDS does not support enabling both IAM and Kerberos authentication methods at the same time.
- For PostgreSQL, you can't use IAM authentication to establish a replication connection.
- You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.
- CloudWatch and CloudTrail don't log IAM authentication. These services do not track `generate-db-auth-token` API calls that authorize the IAM role to enable database connection.
- IAM DB authentication requires compute resources on the database instance. You must have between 300 and 1000 MiB extra memory on your database for reliable connectivity. To see the memory needed for your workload, compare the RES column for RDS processes in the Enhanced Monitoring processlist before and after enabling IAM DB authentication. See Viewing OS metrics in the RDS console.

  If you are using a burstable class instance, avoid running out of memory by reducing the memory used by other parameters like buffers and cache by the same amount.
-
- IAM DB authentication is not supported for RDS on Outposts for any engine.

Recommendations for IAM database authentication

We recommend the following when using IAM database authentication:

- Use IAM database authentication when your application requires fewer than 200 new IAM database authentication connections per second.

  The database engines that work with Amazon RDS don't impose any limits on authentication attempts per second. However, when you use IAM database authentication, your application must generate an authentication token. Your application then uses that token to connect to the DB instance. If you exceed the limit of maximum new connections per second, then the extra overhead of IAM database authentication can cause connection throttling.

  Consider using connection pooling in your applications to mitigate constant connection creation. This can reduce the overhead from IAM DB authentication and allow your applications to reuse existing connections. Alternatively, consider using RDS Proxy for these use cases. RDS Proxy has additional costs. See RDS Proxy pricing.

- The size of an IAM database authentication token depends on many things including the number of IAM tags, IAM service policies, ARN lengths, as well as other IAM and database properties. The minimum size of this token is generally about 1 KB but can be larger. Since this token is used as the password in the connection string to the database using IAM authentication, you should ensure that your database driver (e.g., ODBC) and/or any tools do not limit or otherwise truncate this token due to its size. A truncated token will cause the authentication validation done by the database and IAM to fail.
- If you are using temporary credentials when creating an IAM database authentication token, the temporary credentials must still be valid when using the IAM database authentication token to make a connection request.

Unsupported AWS global condition context keys

IAM database authentication does not support the following subset of AWS global condition context keys.

- `aws:Referer`
- `aws:SourceIp`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserAgent`
- `aws:VpcSourceIp`

For more information, see AWS global condition context keys in the *IAM User Guide*.

Enabling and disabling IAM database authentication

By default, IAM database authentication is disabled on DB instances. You can enable or disable IAM database authentication using the AWS Management Console, AWS CLI, or the API.

You can enable IAM database authentication when you perform one of the following actions:

- To create a new DB instance with IAM database authentication enabled, see Creating an Amazon RDS DB instance.
- To modify a DB instance to enable IAM database authentication, see Modifying an Amazon RDS DB instance.

- To restore a DB instance from a snapshot with IAM database authentication enabled, see Restoring to a DB instance.
- To restore a DB instance to a point in time with IAM database authentication enabled, see Restoring a DB instance to a specified time for Amazon RDS.

Console

Each creation or modification workflow has a **Database authentication** section, where you can enable or disable IAM database authentication. In that section, choose **Password and IAM database authentication** to enable IAM database authentication.

To enable or disable IAM database authentication for an existing DB instance

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the DB instance that you want to modify.

   > **Note**
   >
   > Make sure that the DB instance is compatible with IAM authentication. Check the compatibility requirements in Region and version availability.

4. Choose **Modify**.
5. In the **Database authentication** section, choose **Password and IAM database authentication** to enable IAM database authentication. Choose **Password authentication** or **Password and Kerberos authentication** to disable IAM authentication.
6. You can also choose to enable publishing IAM DB authentication logs to CloudWatch Logs. Under **Log exports**, choose the **iam-db-auth-error log** option. Publishing your logs to CloudWatch Logs consumes storage and you incur charges for that storage. Be sure to delete any CloudWatch Logs that you no longer need.
7. Choose **Continue**.
8. To apply the changes immediately, choose **Immediately** in the **Scheduling of modifications** section.
9. Choose **Modify DB instance** .

AWS CLI

To create a new DB instance with IAM authentication by using the AWS CLI, use the `create-db-instance` command. Specify the `--enable-iam-database-authentication` option, as shown in the following example.

```
aws rds create-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.m3.medium \
    --engine MySQL \
    --allocated-storage 20 \
    --master-username masterawsuser \
    --manage-master-user-password \
    --enable-iam-database-authentication
```

To update an existing DB instance to have or not have IAM authentication, use the AWS CLI command `modify-db-instance`. Specify either the `--enable-iam-database-authentication` or `--no-enable-iam-database-authentication` option, as appropriate.

> **Note**
>
> Make sure that the DB instance is compatible with IAM authentication. Check the compatibility requirements in Region and version availability.

By default, Amazon RDS performs the modification during the next maintenance window. If you want to override this and enable IAM DB authentication as soon as possible, use the `--apply-immediately` parameter.

The following example shows how to immediately enable IAM authentication for an existing DB instance.

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --apply-immediately \
    --enable-iam-database-authentication
```

If you are restoring a DB instance, use one of the following AWS CLI commands:

- `restore-db-instance-to-point-in-time`

- `restore-db-instance-from-db-snapshot`

The IAM database authentication setting defaults to that of the source snapshot. To change this setting, set the `--enable-iam-database-authentication` or `--no-enable-iam-database-authentication` option, as appropriate.

RDS API

To create a new DB instance with IAM authentication by using the API, use the API operation `CreateDBInstance`. Set the `EnableIAMDatabaseAuthentication` parameter to `true`.

To update an existing DB instance to have IAM authentication, use the API operation `ModifyDBInstance`. Set the `EnableIAMDatabaseAuthentication` parameter to `true` to enable IAM authentication, or `false` to disable it.

> **Note**
>
> Make sure that the DB instance is compatible with IAM authentication. Check the compatibility requirements in Region and version availability.

If you are restoring a DB instance, use one of the following API operations:

- `RestoreDBInstanceFromDBSnapshot`
- `RestoreDBInstanceToPointInTime`

The IAM database authentication setting defaults to that of the source snapshot. To change this setting, set the `EnableIAMDatabaseAuthentication` parameter to `true` to enable IAM authentication, or `false` to disable it.

Creating and using an IAM policy for IAM database access

To allow a user or role to connect to your DB instance, you must create an IAM policy. After that, you attach the policy to a permissions set or role.

> **Note**
>
> To learn more about IAM policies, see Identity and access management for Amazon RDS.

The following example policy allows a user to connect to a DB instance using IAM database authentication.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:111122223333:dbuser:db-
ABCDEFGHIJKL01234/db_user"
            ]
        }
    ]
}
```

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:111122223333:dbuser:cluster-
ABCDEFGHIJKL01234/db_user"
            ]
        }
    ]
}
```

Important

A user with administrator permissions can access DB instances without explicit permissions in an IAM policy. If you want to restrict administrator access to DB instances, you can create an IAM role with the appropriate, lesser privileged permissions and assign it to the administrator.

Note

Don't confuse the `rds-db:` prefix with other RDS API operation prefixes that begin with `rds:`. You use the `rds-db:` prefix and the `rds-db:connect` action only for IAM database authentication. They aren't valid in any other context.

The example policy includes a single statement with the following elements:

- `Effect` – Specify `Allow` to grant access to the DB instance. If you don't explicitly allow access, then access is denied by default.
- `Action` – Specify `rds-db:connect` to allow connections to the DB instance.
- `Resource` – Specify an Amazon Resource Name (ARN) that describes one database account in one DB instance. The ARN format is as follows.

```
arn:aws:rds-db:region:account-id:dbuser:DbiResourceId/db-user-name
```

In this format, replace the following:

- *`region`* is the AWS Region for the DB instance. In the example policy, the AWS Region is `us-east-2`.
- *`account-id`* is the AWS account number for the DB instance. In the example policy, the account number is `1234567890`. The user must be in the same account as the account for the DB instance.

  To perform cross-account access, create an IAM role with the policy shown above in the account for the DB instance and allow your other account to assume the role.

- *DbiResourceId* is the identifier for the DB instance. This identifier is unique to an AWS Region and never changes. In the example policy, the identifier is db-ABCDEFGHIJKL01234.

  To find a DB instance resource ID in the AWS Management Console for Amazon RDS, choose the DB instance to see its details. Then choose the **Configuration** tab. The **Resource ID** is shown in the **Configuration** section.

  Alternatively, you can use the AWS CLI command to list the identifiers and resource IDs for all of your DB instance in the current AWS Region, as shown following.

  ```
  aws rds describe-db-instances --query "DBInstances[*].
  [DBInstanceIdentifier,DbiResourceId]"
  ```

  If you are using Amazon Aurora, specify a DbClusterResourceId instead of a DbiResourceId. For more information, see Creating and using an IAM policy for IAM database access in the *Amazon Aurora User Guide*.

  > **Note**
  >
  > If you are connecting to a database through RDS Proxy, specify the proxy resource ID, such as prx-ABCDEFGHIJKL01234. For information about using IAM database authentication with RDS Proxy, see Connecting to a database using IAM authentication.

- *db-user-name* is the name of the database account to associate with IAM authentication. In the example policy, the database account is db_user.

You can construct other ARNs to support various access patterns. The following policy allows access to two different database accounts in a DB instance.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
```

```
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:db-ABCDEFGHIJKL01234/
mary_roe"
         ]
      }
   ]
}
```

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-
ABCDEFGHIJKL01234/mary_roe"
         ]
      }
   ]
}
```

The following policy uses the "*" character to match all DB instances and database accounts for a particular AWS account and AWS Region.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:111122223333:dbuser:*/*"
            ]
        }
    ]
}
```

The following policy matches all of the DB instances for a particular AWS account and AWS Region. However, the policy only grants access to DB instances that have a `jane_doe` database account.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
         ]
      }
   ]
}
```

The user or role has access to only those databases that the database user does. For example, suppose that your DB instance has a database named *dev*, and another database named *test*. If the database user `jane_doe` has access only to *dev*, any users or roles that access that DB instance with the `jane_doe` user also have access only to *dev*. This access restriction is also true for other database objects, such as tables, views, and so on.

An administrator must create IAM policies that grant entities permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the permission sets or roles that require those permissions. For examples of policies, see Identity-based policy examples for Amazon RDS.

Attaching an IAM policy to a permission set or role

After you create an IAM policy to allow database authentication, you need to attach the policy to a permission set or role. For a tutorial on this topic, see Create and attach your first customer managed policy in the *IAM User Guide*.

As you work through the tutorial, you can use one of the policy examples shown in this section as a starting point and tailor it to your needs. At the end of the tutorial, you have a permission set with an attached policy that can make use of the `rds-db:connect` action.

> **Note**
>
> You can map multiple permission sets or roles to the same database user account. For example, suppose that your IAM policy specified the following resource ARN.
>
> ```
> arn:aws:rds-db:us-east-2:123456789012:dbuser:db-12ABC34DEFG5HIJ6KLMNOP78QR/
> jane_doe
> ```
>
> If you attach the policy to *Jane*, *Bob*, and *Diego*, then each of those users can connect to the specified DB instance using the `jane_doe` database account.

Creating a database account using IAM authentication

With IAM database authentication, you don't need to assign database passwords to the user accounts you create. If you remove a user that is mapped to a database account, you should also remove the database account with the DROP USER statement.

> **Note**
>
> The user name used for IAM authentication must match the case of the user name in the database.

Topics

- Using IAM authentication with MariaDB and MySQL
- Using IAM authentication with PostgreSQL

Using IAM authentication with MariaDB and MySQL

With MariaDB and MySQL, authentication is handled by `AWSAuthenticationPlugin`—an AWS-provided plugin that works seamlessly with IAM to authenticate your users. Connect to the DB instance as the master user or a different user who can create users and grant privileges. After connecting, issue the `CREATE USER` statement, as shown in the following example.

```
CREATE USER 'jane_doe' IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

The `IDENTIFIED WITH` clause allows MariaDB and MySQL to use the `AWSAuthenticationPlugin` to authenticate the database account (`jane_doe`). The `AS 'RDS'` clause refers to the authentication method. Make sure the specified database user name is the same as a resource in the IAM policy for IAM database access. For more information, see Creating and using an IAM policy for IAM database access.

> **Note**
>
> If you see the following message, it means that the AWS-provided plugin is not available for the current DB instance.
> `ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded`
> To troubleshoot this error, verify that you are using a supported configuration and that you have enabled IAM database authentication on your DB instance. For more information, see Region and version availability and Enabling and disabling IAM database authentication.

After you create an account using `AWSAuthenticationPlugin`, you manage it in the same way as other database accounts. For example, you can modify account privileges with `GRANT` and `REVOKE` statements, or modify various account attributes with the `ALTER USER` statement.

Database network traffic is encrypted using SSL/TLS when using IAM. To allow SSL connections, modify the user account with the following command.

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

Using IAM authentication with PostgreSQL

To use IAM authentication with PostgreSQL, connect to the DB instance as the master user or a different user who can create users and grant privileges. After connecting, create database users and then grant them the `rds_iam` role as shown in the following example.

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

Make sure the specified database user name is the same as a resource in the IAM policy for IAM database access. For more information, see Creating and using an IAM policy for IAM database access. You must grant the `rds_iam` role to use IAM authentication. You can use nested memberships or indirect grants of the role as well.

Connecting to your DB instance using IAM authentication

With IAM database authentication, you use an authentication token when you connect to your DB instance. An *authentication token* is a string of characters that you use instead of a password. After you generate an authentication token, it's valid for 15 minutes before it expires. If you try to connect using an expired token, the connection request is denied.

Every authentication token must be accompanied by a valid signature, using AWS signature version 4. (For more information, see Signature Version 4 signing process in the *AWS General Reference*.) The AWS CLI and an AWS SDK, such as the AWS SDK for Java or AWS SDK for Python (Boto3), can automatically sign each token you create.

You can use an authentication token when you connect to Amazon RDS from another AWS service, such as AWS Lambda. By using a token, you can avoid placing a password in your code. Alternatively, you can use an AWS SDK to programmatically create and programmatically sign an authentication token.

After you have a signed IAM authentication token, you can connect to an Amazon RDS DB instance. Following, you can find out how to do this using either a command line tool or an AWS SDK, such as the AWS SDK for Java or AWS SDK for Python (Boto3).

For more information, see the following blog posts:

- Use IAM authentication to connect with SQL Workbench/J to Aurora MySQL or Amazon RDS for MySQL
- Using IAM authentication to connect with pgAdmin Amazon Aurora PostgreSQL or Amazon RDS for PostgreSQL

Prerequisites

The following are prerequisites for connecting to your DB instance using IAM authentication:

- Enabling and disabling IAM database authentication
- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication

Topics

- Connecting to your DB instance using IAM authentication with the AWS drivers
- Connecting to your DB instance using IAM authentication from the command line: AWS CLI and mysql client
- Connecting to your DB instance using IAM authentication from the command line: AWS CLI and psql client
- Connecting to your DB instance using IAM authentication and the AWS SDK for .NET
- Connecting to your DB instance using IAM authentication and the AWS SDK for Go
- Connecting to your DB instance using IAM authentication and the AWS SDK for Java
- Connecting to your DB instance using IAM authentication and the AWS SDK for Python (Boto3)

Connecting to your DB instance using IAM authentication with the AWS drivers

The AWS suite of drivers has been designed to provide support for faster switchover and failover times, and authentication with AWS Secrets Manager, AWS Identity and Access Management (IAM),

and Federated Identity. The AWS drivers rely on monitoring DB instance status and being aware of the instance topology to determine the new writer. This approach reduces switchover and failover times to single-digit seconds, compared to tens of seconds for open-source drivers.

For more information on the AWS drivers, see the corresponding language driver for your RDS for MariaDB, RDS for MySQL, or RDS for PostgreSQL DB instance.

> **Note**
>
> The only features supported for RDS for MariaDB are authentication with AWS Secrets Manager, AWS Identity and Access Management (IAM), and Federated Identity.

Connecting to your DB instance using IAM authentication from the command line: AWS CLI and mysql client

You can connect from the command line to an Amazon RDS DB instance with the AWS CLI and `mysql` command line tool as described following.

Prerequisites

The following are prerequisites for connecting to your DB instance using IAM authentication:

- Enabling and disabling IAM database authentication
- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication

> **Note**
>
> For information about connecting to your database using SQL Workbench/J with IAM authentication, see the blog post Use IAM authentication to connect with SQL Workbench/J to Aurora MySQL or Amazon RDS for MySQL.

Topics

- Generating an IAM authentication token
- Connecting to a DB instance

Generating an IAM authentication token

The following example shows how to get a signed authentication token using the AWS CLI.

```
aws rds generate-db-auth-token \
   --hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
   --port 3306 \
   --region us-west-2 \
   --username jane_doe
```

In the example, the parameters are as follows:

- `--hostname` – The host name of the DB instance that you want to access
- `--port` – The port number used for connecting to your DB instance
- `--region` – The AWS Region where the DB instance is running
- `--username` – The database account that you want to access

The first several characters of the token look like the following.

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.

Connecting to a DB instance

The general format for connecting is shown following.

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-
cleartext-plugin --user=userName --password=authToken
```

The parameters are as follows:

- `--host` – The host name of the DB instance that you want to access
- `--port` – The port number used for connecting to your DB instance

- `--ssl-ca` – The full path to the SSL certificate file that contains the public key

  For more information about SSL/TLS support for MariaDB, see SSL/TLS support for MariaDB DB instances on Amazon RDS.

  For more information about SSL/TLS support for MySQL, see SSL/TLS support for MySQL DB instances on Amazon RDS.

  To download an SSL certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
- `--enable-cleartext-plugin` – A value that specifies that `AWSAuthenticationPlugin` must be used for this connection

  If you are using a MariaDB client, the `--enable-cleartext-plugin` option isn't required.
- `--user` – The database account that you want to access
- `--password` – A signed IAM authentication token

The authentication token consists of several hundred characters. It can be unwieldy on the command line. One way to work around this is to save the token to an environment variable, and then use that variable when you connect. The following example shows one way to perform this workaround. In the example, */sample_dir/* is the full path to the SSL certificate file that contains the public key.

```
RDSHOST="mysqldb.123456789012.us-east-1.rds.amazonaws.com"
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-
west-2 --username jane_doe )"

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

When you connect using `AWSAuthenticationPlugin`, the connection is secured using SSL. To verify this, type the following at the `mysql>` command prompt.

```
show status like 'Ssl%';
```

The following lines in the output show more details.

```
+---------------+-------------+
```

```
| Variable_name | Value

                                                                    |
+---------------+-------------+
| ...           | ...
| Ssl_cipher    | AES256-SHA

                                                                    |
| ...           | ...
| Ssl_version   | TLSv1.1

                                                                    |
| ...           | ...
+-----------------------------+
```

If you want to connect to a DB instance through a proxy, see Connecting to a database using IAM authentication.

Connecting to your DB instance using IAM authentication from the command line: AWS CLI and psql client

You can connect from the command line to an Amazon RDS for PostgreSQL DB instance with the AWS CLI and psql command line tool as described following.

Prerequisites

The following are prerequisites for connecting to your DB instance using IAM authentication:

- Enabling and disabling IAM database authentication
- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication

> **Note**
>
> For information about connecting to your database using pgAdmin with IAM authentication, see the blog post Using IAM authentication to connect with pgAdmin Amazon Aurora PostgreSQL or Amazon RDS for PostgreSQL.

Topics

- Generating an IAM authentication token
- Connecting to an Amazon RDS PostgreSQL instance

Generating an IAM authentication token

The authentication token consists of several hundred characters so it can be unwieldy on the command line. One way to work around this is to save the token to an environment variable, and then use that variable when you connect. The following example shows how to use the AWS CLI to get a signed authentication token using the `generate-db-auth-token` command, and store it in a `PGPASSWORD` environment variable.

```
export RDSHOST="rdspostgres.123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

In the example, the parameters to the `generate-db-auth-token` command are as follows:

- `--hostname` – The host name of the DB instance that you want to access
- `--port` – The port number used for connecting to your DB instance
- `--region` – The AWS Region where the DB instance is running
- `--username` – The database account that you want to access

The first several characters of the generated token look like the following.

```
rdspostgres.123456789012.us-west-2.rds.amazonaws.com:5432/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.

Connecting to an Amazon RDS PostgreSQL instance

The general format for using psql to connect is shown following.

```
psql "host=hostName port=portNumber sslmode=verify-full
 sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName
 password=authToken"
```

The parameters are as follows:

- `host` – The host name of the DB instance that you want to access
- `port` – The port number used for connecting to your DB instance
- `sslmode` – The SSL mode to use

  When you use `sslmode=verify-full`, the SSL connection verifies the DB instance endpoint against the endpoint in the SSL certificate.
- `sslrootcert` – The full path to the SSL certificate file that contains the public key

  For more information, see Using SSL with a PostgreSQL DB instance.

  To download an SSL certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .
- dbname – The database that you want to access
- `user` – The database account that you want to access
- `password` – A signed IAM authentication token

> **Note**
>
> You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.

The following example shows using psql to connect. In the example, psql uses the environment variable RDSHOST for the host and the environment variable PGPASSWORD for the generated token. Also, */sample_dir/* is the full path to the SSL certificate file that contains the public key.

```
export RDSHOST="rdspostgres.123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

```
psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-
bundle.pem dbname=DBName user=jane_doe password=$PGPASSWORD"
```

If you want to connect to a DB instance through a proxy, see Connecting to a database using IAM authentication.

Connecting to your DB instance using IAM authentication and the AWS SDK for .NET

You can connect to an RDS for MariaDB, MySQL, or PostgreSQL DB instance with the AWS SDK for .NET as described following.

Prerequisites

The following are prerequisites for connecting to your DB instance using IAM authentication:

- Enabling and disabling IAM database authentication
- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication

Examples

The following code examples show how to generate an authentication token, and then use it to connect to a DB instance.

To run this code example, you need the AWS SDK for .NET, found on the AWS site. The `AWSSDK.CORE` and the `AWSSDK.RDS` packages are required. To connect to a DB instance, use the .NET database connector for the DB engine, such as MySqlConnector for MariaDB or MySQL, or Npgsql for PostgreSQL.

This code connects to a MariaDB or MySQL DB instance. Modify the values of the following variables as needed:

- `server` – The endpoint of the DB instance that you want to access
- `user` – The database account that you want to access
- `database` – The database that you want to access
- `port` – The port number used for connecting to your DB instance
- `SslMode` – The SSL mode to use

When you use `SslMode=Required`, the SSL connection verifies the DB instance endpoint against the endpoint in the SSL certificate.

- `SslCa` – The full path to the SSL certificate for Amazon RDS

  To download a certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

> **Note**
>
> You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
  class Program
  {
    static void Main(string[] args)
    {
      var pwd =
 Amazon.RDS.Util.RDSAuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
 "mysqldb.123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
      // for debug only Console.Write("{0}\n", pwd);  //this verifies the token is
 generated

      MySqlConnection conn = new MySqlConnection($"server=mysqldb.123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password={pwd};SslMode=Required;
      conn.Open();

      // Define a query
      MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

      // Execute a query
      MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();
```

```
      // Read all rows and output the first column in each row
      while (mysqlDataRdr.Read())
        Console.WriteLine(mysqlDataRdr[0]);

      mysqlDataRdr.Close();
      // Close connection
      conn.Close();
    }
  }
}
```

This code connects to a PostgreSQL DB instance.

Modify the values of the following variables as needed:

- `Server` – The endpoint of the DB instance that you want to access
- `User ID` – The database account that you want to access
- `Database` – The database that you want to access
- `Port` – The port number used for connecting to your DB instance
- `SSL Mode` – The SSL mode to use

  When you use `SSL Mode=Required`, the SSL connection verifies the DB instance endpoint against the endpoint in the SSL certificate.
- `Root Certificate` – The full path to the SSL certificate for Amazon RDS

  To download a certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

Note

You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
```

```
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
 RDSAuthTokenGenerator.GenerateAuthToken("postgresmydb.123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.Write("{0}\n", pwd);  //this verifies the token is generated

            NpgsqlConnection conn = new
 NpgsqlConnection($"Server=postgresmydb.123456789012.us-east-1.rds.amazonaws.com;User
 Id=jane_doe;Password={pwd};Database=mydb;SSL Mode=Require;Root
 Certificate=full_path_to_ssl_certificate");
            conn.Open();

            // Define a query
                   NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM
 pg_user", conn);

            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.Write("{0}\n", dr[0]);

            // Close connection
            conn.Close();
        }
    }
}
```

If you want to connect to a DB instance through a proxy, see Connecting to a database using IAM authentication.

Connecting to your DB instance using IAM authentication and the AWS SDK for Go

You can connect to an RDS for MariaDB, MySQL, or PostgreSQL DB instance with the AWS SDK for Go as described following.

Prerequisites

The following are prerequisites for connecting to your DB instance using IAM authentication:

- Enabling and disabling IAM database authentication
- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication

Examples

To run these code examples, you need the AWS SDK for Go, found on the AWS site.

Modify the values of the following variables as needed:

- `dbName` – The database that you want to access
- `dbUser` – The database account that you want to access
- `dbHost` – The endpoint of the DB instance that you want to access

> **Note**
>
> You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.

- `dbPort` – The port number used for connecting to your DB instance
- `region` – The AWS Region where the DB instance is running

In addition, make sure the imported libraries in the sample code exist on your system.

> **Important**
>
> The examples in this section use the following code to provide credentials that access a database from a local environment:
> `creds := credentials.NewEnvCredentials()`
> If you are accessing a database from an AWS service, such as Amazon EC2 or Amazon ECS, you can replace the code with the following code:
> `sess := session.Must(session.NewSession())`
> `creds := sess.Config.Credentials`
> If you make this change, make sure you add the following import:
> `"github.com/aws/aws-sdk-go/aws/session"`

Topics

- Connecting using IAM authentication and the AWS SDK for Go V2
- Connecting using IAM authentication and the AWS SDK for Go V1.

Connecting using IAM authentication and the AWS SDK for Go V2

You can connect to a DB instance using IAM authentication and the AWS SDK for Go V2.

The following code examples show how to generate an authentication token, and then use it to connect to a DB instance.

This code connects to a MariaDB or MySQL DB instance.

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/go-sql-driver/mysql"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "mysqldb.123456789012.us-east-1.rds.amazonaws.com"
     var dbPort int = 3306
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
```

```
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

This code connects to a PostgreSQL DB instance.

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/lib/pq"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "postgresmydb.123456789012.us-east-1.rds.amazonaws.com"
     var dbPort int = 5432
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
```

```
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

If you want to connect to a DB instance through a proxy, see Connecting to a database using IAM authentication.

Connecting using IAM authentication and the AWS SDK for Go V1.

You can connect to a DB instance using IAM authentication and the AWS SDK for Go V1

The following code examples show how to generate an authentication token, and then use it to connect to a DB instance.

This code connects to a MariaDB or MySQL DB instance.

```
package main

import (
    "database/sql"
    "fmt"
    "log"
```

```
    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqldb.123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

This code connects to a PostgreSQL DB instance.

```
package main

import (
 "database/sql"
 "fmt"

 "github.com/aws/aws-sdk-go/aws/credentials"
```

```
 "github.com/aws/aws-sdk-go/service/rds/rdsutils"
 _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmydb.123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

If you want to connect to a DB instance through a proxy, see Connecting to a database using IAM authentication.

Connecting to your DB instance using IAM authentication and the AWS SDK for Java

You can connect to an RDS for MariaDB, MySQL, or PostgreSQL DB instance with the AWS SDK for Java as described following.

Prerequisites

The following are prerequisites for connecting to your DB instance using IAM authentication:

- Enabling and disabling IAM database authentication
- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication
- Set up the AWS SDK for Java

For examples on how to use the SDK for Java 2.x, see Amazon RDS examples using SDK for Java 2.x. You can also use the AWS Advanced JDBC Wrapper, see AWS Advanced JDBC Wrapper documentation.

Topics

- Generating an IAM authentication token
- Manually constructing an IAM authentication token
- Connecting to a DB instance

Generating an IAM authentication token

If you are writing programs using the AWS SDK for Java, you can get a signed authentication token using the `RdsIamAuthTokenGenerator` class. Using this class requires that you provide AWS credentials. To do this, you create an instance of the `DefaultAWSCredentialsProviderChain` class. `DefaultAWSCredentialsProviderChain` uses the first AWS access key and secret key that it finds in the default credential provider chain. For more information about AWS access keys, see Managing access keys for users.

> **Note**
>
> You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.

After you create an instance of `RdsIamAuthTokenGenerator`, you can call the `getAuthToken` method to obtain a signed token. Provide the AWS Region, host name, port number, and user name. The following code example illustrates how to do this.

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
```

```
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

     String region = "us-west-2";
     String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
     String port = "3306";
     String username = "jane_doe";

     System.out.println(generateAuthToken(region, hostname, port, username));
    }

    static String generateAuthToken(String region, String hostName, String port, String
 username) {

     RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
      .credentials(new DefaultAWSCredentialsProviderChain())
      .region(region)
      .build();

     String authToken = generator.getAuthToken(
      GetIamAuthTokenRequest.builder()
      .hostname(hostName)
      .port(Integer.parseInt(port))
      .userName(username)
      .build());

     return authToken;
    }

}
```

Manually constructing an IAM authentication token

In Java, the easiest way to generate an authentication token is to use `RdsIamAuthTokenGenerator`. This class creates an authentication token for you, and then signs it using AWS signature version 4. For more information, see Signature version 4 signing process in the *AWS General Reference.*

However, you can also construct and sign an authentication token manually, as shown in the following code example.

```
package com.amazonaws.codesamples;

import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;
import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
    public static String action = "connect";
    public static String canonicalURIParameter = "/";
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new
 SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
```

```
       DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
    String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
    String awsSecretKey = creds.getCredentials().getAWSSecretKey();
       String expiryMinutes = "900";

       System.out.println("Step 1:  Create a canonical request:");
       String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
       System.out.println(canonicalString);
       System.out.println();

       System.out.println("Step 2:  Create a string to sign:");
       String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
       System.out.println(stringToSign);
       System.out.println();

       System.out.println("Step 3:  Calculate the signature:");
       String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
newSigningKey(awsSecretKey, date, region, serviceName)));
       System.out.println(signature);
       System.out.println();

       System.out.println("Step 4:  Add the signing info to the request");

       System.out.println(appendSignature(signature));
       System.out.println();

   }

   //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
should be in format YYYYMMDDTHHMMSSZ
   public static String createCanonicalString(String user, String accessKey, String
date, String dateTime, String region, String expiryPeriod, String hostName, String
port) throws Exception {
       canonicalQueryParameters.put("Action", action);
       canonicalQueryParameters.put("DBUser", user);
       canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
       canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date +
"%2F" + region + "%2F" + serviceName + "%2Faws4_request");
       canonicalQueryParameters.put("X-Amz-Date", dateTime);
       canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
       canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
```

```
        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
 canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
 currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/?" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString
 + '\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String dateTime, String canonicalRequest,
 String accessKey, String date, String region) throws Exception {
        String credentialScope = date + "/" + region + "/" + serviceName + "/
aws4_request";
        return algorithm + '\n' + dateTime + '\n' + credentialScope + '\n' +
 BinaryUtils.toHex(hash(canonicalRequest));

    }

    //Step 3: Calculate signature
    /**
     * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
     * the signing key and computing the signature. Refer to
     * http://docs.aws.amazon
     * .com/general/latest/gr/sigv4-calculate-signature.html
     */
    public static byte[] calculateSignature(String stringToSign,
                                            byte[] signingKey) {
        return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
                SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(byte[] data, byte[] key,
```

```
                        SigningAlgorithm algorithm) throws SdkClientException {
        try {
            Mac mac = algorithm.getMac();
            mac.init(new SecretKeySpec(key, algorithm.toString()));
            return mac.doFinal(data);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    public static byte[] newSigningKey(String secretKey,
                                    String dateStamp, String regionName, String
serviceName) {
        byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
        byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
        byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
        byte[] kService = sign(serviceName, kRegion,
                SigningAlgorithm.HmacSHA256);
        return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(String stringData, byte[] key,
                        SigningAlgorithm algorithm) throws SdkClientException {
        try {
            byte[] data = stringData.getBytes(UTF8);
            return sign(data, key, algorithm);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    //Step 4: append the signature
    public static String appendSignature(String signature) {
        return requestWithoutSignature + "&X-Amz-Signature=" + signature;
    }

    public static byte[] hash(String s) throws Exception {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(s.getBytes(UTF8));
```

```
            return md.digest();
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to compute hash while signing request: "
                            + e.getMessage(), e);
        }
    }
}
```

Connecting to a DB instance

The following code example shows how to generate an authentication token, and then use it to connect to an instance running MariaDB or MySQL.

To run this code example, you need the AWS SDK for Java, found on the AWS site. In addition, you need the following:

- MySQL Connector/J. This code example was tested with `mysql-connector-java-5.1.33-bin.jar`.
- An intermediate certificate for Amazon RDS that is specific to an AWS Region. (For more information, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .) At runtime, the class loader looks for the certificate in the same directory as this Java code example, so that the class loader can find it.
- Modify the values of the following variables as needed:
  - `RDS_INSTANCE_HOSTNAME` – The host name of the DB instance that you want to access.
  - `RDS_INSTANCE_PORT` – The port number used for connecting to your PostgreSQL DB instance.
  - `REGION_NAME` – The AWS Region where the DB instance is running.
  - `DB_USER` – The database account that you want to access.
  - `SSL_CERTIFICATE` – An SSL certificate for Amazon RDS that is specific to an AWS Region.

    To download a certificate for your AWS Region, see Using SSL/TLS to encrypt a connection to a DB instance or cluster . Place the SSL certificate in the same directory as this Java program file, so that the class loader can find the certificate at runtime.

This code example obtains AWS credentials from the default credential provider chain.

Note

Specify a password for `DEFAULT_KEY_STORE_PASSWORD` other than the prompt shown here as a security best practice.

```
package com.amazonaws.samples;

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

import java.net.URL;

public class IAMDatabaseAuthenticationTester {
    //&AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
 access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
 creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY =
 creds.getCredentials().getAWSSecretKey();

    //Configuration parameters for the generation of the IAM Database Authentication
 token
```

```
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME +
 ":" + RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-
cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
        while (rs.next()) {
             String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
        }

        //close the connection
        stmt.close();
        connection.close();

        clearSslProperties();

    }

    /**
     * This method returns a connection to the db instance authenticated using IAM
 Database Authentication
     * @return
     * @throws Exception
     */
    private static Connection getDBConnectionUsingIam() throws Exception {
```

```
        setSslProperties();
        return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
    }

    /**
     * This method sets the mysql connection properties which includes the IAM Database
 Authentication token
     * as the password. It also specifies that SSL verification is required.
     * @return
     */
    private static Properties setMySqlConnectionProperties() {
        Properties mysqlConnectionProperties = new Properties();
        mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
        mysqlConnectionProperties.setProperty("useSSL", "true");
        mysqlConnectionProperties.setProperty("user",DB_USER);
        mysqlConnectionProperties.setProperty("password",generateAuthToken());
        return mysqlConnectionProperties;
    }

    /**
     * This method generates the IAM Auth Token.
     * An example IAM Auth Token would look like follows:
     * btusi123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?
Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAPFXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
     * @return
     */
    private static String generateAuthToken() {
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
 AWS_SECRET_KEY);

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
                .credentials(new
 AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
        return generator.getAuthToken(GetIamAuthTokenRequest.builder()

 .hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
    }

    /**
     * This method sets the SSL properties which specify the key store file, its type
 and password:
```

```
     * @throws Exception
     */
    private static void setSslProperties() throws Exception {
        System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
        System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
        System.setProperty("javax.net.ssl.trustStorePassword",
DEFAULT_KEY_STORE_PASSWORD);
    }

    /**
     * This method returns the path of the Key Store File needed for the SSL
verification during the IAM Database Authentication to
     * the db instance.
     * @return
     * @throws Exception
     */
    private static String createKeyStoreFile() throws Exception {
        return createKeyStoreFile(createCertificate()).getPath();
    }

    /**
     *  This method generates the SSL certificate
     * @return
     * @throws Exception
     */
    private static X509Certificate createCertificate() throws Exception {
        CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
        URL url = new File(SSL_CERTIFICATE).toURI().toURL();
        if (url == null) {
            throw new Exception();
        }
        try (InputStream certInputStream = url.openStream()) {
            return (X509Certificate) certFactory.generateCertificate(certInputStream);
        }
    }

    /**
     * This method creates the Key Store File
     * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
     * @return
     * @throws Exception
     */
    private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
Exception {
```

```
        File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
 KEY_STORE_FILE_SUFFIX);
        try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
            KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
            ks.load(null);
            ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
            ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
        }
        return keyStoreFile;
    }

    /**
     * This method clears the SSL properties.
     * @throws Exception
     */
    private static void clearSslProperties() throws Exception {
           System.clearProperty("javax.net.ssl.trustStore");
           System.clearProperty("javax.net.ssl.trustStoreType");
           System.clearProperty("javax.net.ssl.trustStorePassword");
    }

}
```

If you want to connect to a DB instance through a proxy, see Connecting to a database using IAM authentication.

Connecting to your DB instance using IAM authentication and the AWS SDK for Python (Boto3)

You can connect to an RDS for MariaDB, MySQL, or PostgreSQL DB instance with the AWS SDK for Python (Boto3) as described following.

Prerequisites

The following are prerequisites for connecting to your DB instance using IAM authentication:

- Enabling and disabling IAM database authentication
- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication

In addition, make sure the imported libraries in the sample code exist on your system.

Examples

The code examples use profiles for shared credentials. For information about the specifying credentials, see Credentials in the AWS SDK for Python (Boto3) documentation.

The following code examples show how to generate an authentication token, and then use it to connect to a DB instance.

To run this code example, you need the AWS SDK for Python (Boto3), found on the AWS site.

Modify the values of the following variables as needed:

- `ENDPOINT` – The endpoint of the DB instance that you want to access
- `PORT` – The port number used for connecting to your DB instance
- `USER` – The database account that you want to access
- `REGION` – The AWS Region where the DB instance is running
- `DBNAME` – The database that you want to access
- `SSLCERTIFICATE` – The full path to the SSL certificate for Amazon RDS

  For `ssl_ca`, specify an SSL certificate. To download an SSL certificate, see Using SSL/TLS to encrypt a connection to a DB instance or cluster .

> **Note**
>
> You cannot use a custom Route 53 DNS record instead of the DB instance endpoint to generate the authentication token.

This code connects to a MariaDB or MySQL DB instance.

Before running this code, install the PyMySQL driver by following the instructions in the Python Package Index.

```
import pymysql
import sys
import boto3
import os

ENDPOINT="mysqldb.123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
```

```
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn =
 pymysql.connect(auth_plugin_map={'mysql_clear_password':None},host=ENDPOINT,
 user=USER, password=token, port=PORT, database=DBNAME, ssl_ca='SSLCERTIFICATE',
 ssl_verify_identity=True, ssl_verify_cert=True)
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

This code connects to a PostgreSQL DB instance.

Before running this code, install `psycopg2` by following the instructions in Psycopg documentation.

```
import psycopg2
import sys
import boto3
import os

ENDPOINT="postgresmydb.123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
```

```
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
 password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

If you want to connect to a DB instance through a proxy, see Connecting to a database using IAM authentication.

Troubleshooting for IAM DB authentication

Following, you can find troubleshooting ideas for some common IAM DB authentication issues and information on CloudWatch logs and metrics for IAM DB authentication.

Exporting IAM DB authentication error logs to CloudWatch Logs

IAM DB authentication error logs are stored on the database host, and you can export these logs your CloudWatch Logs account. Use the logs and remediation methods in this page to troubleshoot IAM DB authentication issues.

You can enable log exports to CloudWatch Logs from the console, AWS CLI, and RDS API. For console instructions, see Publishing database logs to Amazon CloudWatch Logs.

To export your IAM DB authentication error logs to CloudWatch Logs when creating a DB instance from the AWS CLI, use the following command:

```
aws rds create-db-instance --db-instance-identifier mydbinstance \
--region us-east-1 \
--db-instance-class db.t3.large \
--allocated-storage 50 \
```

```
--engine postgres \
--engine-version 16 \
--port 5432 \
--master-username master \
--master-user-password password \
--publicly-accessible \
--enable-iam-database-authentication \
--enable-cloudwatch-logs-exports=iam-db-auth-error
```

To export your IAM DB authentication error logs to CloudWatch Logs when modifying a DB instance from the AWS CLI, use the following command:

```
aws rds modify-db-instance --db-instance-identifier mydbinstance \
--region us-east-1 \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["iam-db-auth-error"]}'
```

To verify if your DB instance is exporting IAM DB authentication logs to CloudWatch Logs, check if the `EnabledCloudwatchLogsExports` parameter is set to `iam-db-auth-error` in the output for the `describe-db-instances` command.

```
aws rds describe-db-instances --region us-east-1 --db-instance-identifier mydbinstance
            ...

             "EnabledCloudwatchLogsExports": [
                "iam-db-auth-error"
            ],
            ...
```

IAM DB authentication CloudWatch metrics

Amazon RDS delivers near-real time metrics about IAM DB authentication to your Amazon CloudWatch account. The following table lists the IAM DB authentication metrics available using CloudWatch:

Metric	Description
`IamDbAuthConnectionRequests`	Total number of connection requests made with IAM DB authentication.

Metric	Description
IamDbAuthConnectionSuccess	Total number of successful IAM DB authentic ation requests.
IamDbAuthConnectionFailure	Total number of failed IAM DB authentication requests.
IamDbAuthConnectionFailureI nvalidToken	Total number of failed IAM DB authentication requests due to invalid token.
IamDbAuthConnectionFailureI nsufficientPermissions	Total number of failed IAM DB authentication requests due to incorrect policies or permissio ns.
IamDbAuthConnectionFailureT hrottling	Total number of failed IAM DB authentic ation requests due to IAM DB authentication throttling.
IamDbAuthConnectionFailureS erverError	Total number of failed IAM DB authentication requests due to an internal server error in the IAM DB authentication feature.

Common issues and solutions

You might encounter the following issues when using IAM DB authention. Use the remediation steps in the table to solve the issues:

Error	Metric(s)	Cause	Solution
[ERROR] Failed to authenticate the connectio n request for user *db_user* because the provided token is malformed	IamDbAuth Connectio nFailure IamDbAuth Connectio nFailureI nvalidToken	The IAM DB authentia tion token in the connection request is either not a valid SigV4a token, or it is not formatted correctly.	Check your token generation strategy in your applicati on. In some cases, make sure you are passing the token with valid formatting. Truncating the token

Error	Metric(s)	Cause	Solution
or otherwise invalid. (Status Code: 400, Error Code: InvalidToken)			(or incorrect string formatting) will make the token invalid.
[ERROR] Failed to authenticate the connection request for user *db_user* because the token age is longer than 15 minutes. (Status Code: 400, Error Code:ExpiredToken)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureInvalidToken	The IAM DB authentication token has expired. Tokens are only valid for 15 minutes.	Check your token caching and/or token re-use logic in your application. You should not re-use tokens that are older than 15 minutes.

Error	Metric(s)	Cause	Solution
[ERROR] Failed to authorize the connection request for user *db_user* because the IAM policy assumed by the caller 'arn:aws:sts::123456789012:assumed-role/ <RoleName>/ <RoleSession>' is not authorized to perform `rds-db:connect` on the DB instance. (Status Code: 403, Error Code:NotAuthorized)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureInsufficientPermissions	This error might be due to the following reasons: • The IAM policy assumed by the application does not authorize the rds-db:connect action. • You are assuming the incorrect role/policy for *db_user* to connect to the database. • You are assuming the correct policy for *db_user*, but you are not connecting to the correct database.	Verify that the IAM role and/or policy you are assuming in your application. Make sure you assume the same policy to generate the token as to connect to the DB.
[ERROR] Failed to authorize the connection request for user *db_user* due to IAM DB authentication throttling. (Status Code: 429, Error Code: ThrottlingException)	IamDbAuthConnectionFailure IamDbAuthConnectionFailureThrottling	You are making too many connection requests to your DB in a short amount of time. IAM DB authentication throttling limit is 200 connections per second.	Reduce the rate of establishing new connections with IAM authentication. Consider implementing connection pooling using RDS Proxy in order to reuse established connections in your application.

Error	Metric(s)	Cause	Solution

`[ERROR] Failed to authorize the connection request for user db_user due to an internal IAM DB authentication error. (Status Code: 500, Error Code: InternalError)`	`IamDbAuthConnectionFailure`
	
	`IamDbAuthConnectionFailureThrottling`
		There was an internal error while authorizing the DB connecito n with IAM DB authentication.	Reach out to https://aws.amazon.com/premiumsupport/ to investigate the issue.

Troubleshooting Amazon RDS identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon RDS and IAM.

Topics

- I'm not authorized to perform an action in Amazon RDS
- I'm not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Amazon RDS resources

I'm not authorized to perform an action in Amazon RDS

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when the `mateojackson` user tries to use the console to view details about a *widget* but does not have `rds:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rds:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *my-example-widget* resource using the `rds:GetWidget` action.

I'm not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials. Ask that person to update your policies to allow you to pass a role to Amazon RDS.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when a user named `marymajor` tries to use the console to perform an action in Amazon RDS. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to allow people outside of my AWS account to access my Amazon RDS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon RDS supports these features, see How Amazon RDS works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.

- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging and monitoring in Amazon RDS

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon RDS and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon RDS resources and responding to potential incidents:

Amazon CloudWatch Alarms

Using Amazon CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms do not invoke actions because they are in a particular state. Rather the state must have changed and been maintained for a specified number of periods.

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon RDS . CloudTrail captures all API calls for Amazon RDS as events, including calls from the console and from code calls to Amazon RDS API operations. Using the information collected by CloudTrail, you can determine the request that was made to Amazon RDS , the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Monitoring Amazon RDS API calls in AWS CloudTrail .

Enhanced Monitoring

Amazon RDS provides metrics in real time for the operating system (OS) that your DB instance runs on. You can view the metrics for your DB instance using the console, or consume the Enhanced Monitoring JSON output from Amazon CloudWatch Logs in a monitoring system of your choice. For more information, see Monitoring OS metrics with Enhanced Monitoring .

Amazon RDS Performance Insights

Performance Insights expands on existing Amazon RDS monitoring features to illustrate your database's performance and help you analyze any issues that affect it. With the Performance

Insights dashboard, you can visualize the database load and filter the load by waits, SQL statements, hosts, or users. For more information, see Monitoring DB load with Performance Insights on Amazon RDS .

Database Logs

You can view, download, and watch database logs using the AWS Management Console, AWS CLI, or RDS API. For more information, see Monitoring Amazon RDS log files .

Amazon RDS Recommendations

Amazon RDS provides automated recommendations for database resources. These recommendations provide best practice guidance by analyzing DB instance configuration, usage, and performance data. For more information, see Recommendations from Amazon RDS .

Amazon RDS Event Notification

Amazon RDS uses the Amazon Simple Notification Service (Amazon SNS) to provide notification when an Amazon RDS event occurs. These notifications can be in any notification form supported by Amazon SNS for an AWS Region, such as an email, a text message, or a call to an HTTP endpoint. For more information, see Working with Amazon RDS event notification .

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

Trusted Advisor has the following Amazon RDS -related checks:

- Amazon RDS Idle DB Instances
- Amazon RDS Security Group Access Risk
- Amazon RDS Backups
- Amazon RDS Multi-AZ

For more information on these checks, see Trusted Advisor best practices (checks).

For more information about monitoring Amazon RDS , see Monitoring metrics in an Amazon RDS instance .

Compliance validation for Amazon RDS

Third-party auditors assess the security and compliance of Amazon RDS as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS services in scope by compliance program. For general information, see AWS compliance programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading reports in AWS Artifact.

Your compliance responsibility when using Amazon RDS is determined by the sensitivity of your data, your organization's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and compliance quick start guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS compliance resources – This collection of workbooks and guides that might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub CSPM – This AWS service provides a comprehensive view of your security state within AWS. Security Hub CSPM uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub CSPM controls reference.

Resilience in Amazon RDS

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS global infrastructure.

In addition to the AWS global infrastructure, Amazon RDS offers features to help support your data resiliency and backup needs.

Backup and restore

Amazon RDS creates and saves automated backups of your DB instance. Amazon RDS creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases.

Amazon RDS creates automated backups of your DB instance during the backup window of your DB instance. Amazon RDS saves the automated backups of your DB instance according to the backup retention period that you specify. If necessary, you can recover your database to any point in time during the backup retention period. You can also back up your DB instance manually, by manually creating a DB snapshot.

You can create a DB instance by restoring from this DB snapshot as a disaster recovery solution if the source DB instance fails.

For more information, see Backing up, restoring, and exporting data.

Replication

Amazon RDS uses the MariaDB, MySQL, Oracle, and PostgreSQL DB engines' built-in replication functionality to create a special type of DB instance called a read replica from a source DB instance. Updates made to the source DB instance are asynchronously copied to the read replica. You can reduce the load on your source DB instance by routing read queries from your applications to the read replica. Using read replicas, you can elastically scale out beyond the capacity constraints of a single DB instance for read-heavy database workloads. You can promote a read replica to a

standalone instance as a disaster recovery solution if the source DB instance fails. For some DB engines, Amazon RDS also supports other replication options.

For more information, see Working with DB instance read replicas.

Failover

Amazon RDS provides high availability and failover support for DB instances using Multi-AZ deployments. Amazon RDS uses several different technologies to provide failover support. Multi-AZ deployments for Oracle, PostgreSQL, MySQL, and MariaDB DB instances use Amazon's failover technology. SQL Server DB instances use SQL Server Database Mirroring (DBM).

For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.

Infrastructure security in Amazon RDS

As a managed service, Amazon Relational Database Service is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon RDS through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

In addition, Amazon RDS offers features to help support infrastructure security.

Security groups

Security groups control the access that traffic has in and out of a DB instance. By default, network access is turned off to a DB instance. You can specify rules in a security group that allow access from an IP address range, port, or security group. After ingress rules are configured, the same rules apply to all DB instances that are associated with that security group.

For more information, see Controlling access with security groups.

Public accessibility

When you launch a DB instance inside a virtual private cloud (VPC) based on the Amazon VPC service, you can turn on or off public accessibility for that DB instance. To designate whether the DB instance that you create has a DNS name that resolves to a public IP address, you use the *Public accessibility* parameter. By using this parameter, you can designate whether there is public access to the DB instance. You can modify a DB instance to turn on or off public accessibility by modifying the *Public accessibility* parameter.

For more information, see Hiding a DB instance in a VPC from the internet.

Note

If your DB instance is in a VPC but isn't publicly accessible, you can also use an AWS Site-to-Site VPN connection or an Direct Connect connection to access it from a private network. For more information, see Internetwork traffic privacy.

Amazon RDS API and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon RDS API endpoints by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink.

AWS PrivateLink enables you to privately access Amazon RDS API operations without an internet gateway, NAT device, VPN connection, or Direct Connect connection. DB instances in your VPC don't need public IP addresses to communicate with Amazon RDS API endpoints to launch, modify, or terminate DB instances. Your DB instances also don't need public IP addresses to use any of the available RDS API operations. Traffic between your VPC and Amazon RDS doesn't leave the Amazon network.

Each interface endpoint is represented by one or more elastic network interfaces in your subnets. For more information on elastic network interfaces, see Elastic network interfaces in the *Amazon EC2 User Guide.*

For more information about VPC endpoints, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*. For more information about RDS API operations, see Amazon RDS API Reference.

You don't need an interface VPC endpoint to connect to a DB instance. For more information, see Scenarios for accessing a DB instance in a VPC.

Considerations for VPC endpoints

Before you set up an interface VPC endpoint for Amazon RDS API endpoints, ensure that you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

All RDS API operations relevant to managing Amazon RDS resources are available from your VPC using AWS PrivateLink.

VPC endpoint policies are supported for RDS API endpoints. By default, full access to RDS API operations is allowed through the endpoint. For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Availability

Amazon RDS API currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (New Zealand)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Taipei)
- Asia Pacific (Thailand)
- Asia Pacific (Tokyo)
- Canada (Central)
- Canada West (Calgary)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Zurich)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Stockholm)
- Europe (Milan)
- Israel (Tel Aviv)
- Mexico (Central)
- Middle East (Bahrain)
- South America (São Paulo)

- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Creating an interface VPC endpoint for Amazon RDS API

You can create a VPC endpoint for the Amazon RDS API using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

Create a VPC endpoint for Amazon RDS API using the service name `com.amazonaws.region.rds`.

Excluding AWS Regions in China, if you enable private DNS for the endpoint, you can make API requests to Amazon RDS with the VPC endpoint using its default DNS name for the AWS Region, for example `rds.us-east-1.amazonaws.com`. For the China (Beijing) and China (Ningxia) AWS Regions, you can make API requests with the VPC endpoint using `rds-api.cn-north-1.amazonaws.com.cn` and `rds-api.cn-northwest-1.amazonaws.com.cn`, respectively.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for Amazon RDS API

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon RDS API. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Amazon RDS API actions

The following is an example of an endpoint policy for Amazon RDS API. When attached to an endpoint, this policy grants access to the listed Amazon RDS API actions for all principals on all resources.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBInstance",
            "rds:ModifyDBInstance",
            "rds:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

Example: VPC endpoint policy that denies all access from a specified AWS account

The following VPC endpoint policy denies AWS account 123456789012 all access to resources using the endpoint. The policy allows all actions from other accounts.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Security best practices for Amazon RDS

Use AWS Identity and Access Management (IAM) accounts to control access to Amazon RDS API operations, especially operations that create, modify, or delete Amazon RDS resources. Such

resources include DB instances , security groups, and parameter groups. Also use IAM to control actions that perform common administrative actions such as backing up and restoring DB instances .

- Create an individual user for each person who manages Amazon RDS resources, including yourself. Don't use AWS root credentials to manage Amazon RDS resources.
- Grant each user the minimum set of permissions required to perform his or her duties.
- Use IAM groups to effectively manage permissions for multiple users.
- Rotate your IAM credentials regularly.
- Configure AWS Secrets Manager to automatically rotate the secrets for Amazon RDS . For more information, see Rotating your AWS Secrets Manager secrets in the *AWS Secrets Manager User Guide*. You can also retrieve the credential from AWS Secrets Manager programmatically. For more information, see Retrieving the secret value in the *AWS Secrets Manager User Guide*.

For more information about Amazon RDS security, see Security in Amazon RDS . For more information about IAM, see AWS Identity and Access Management. For information on IAM best practices, see IAM best practices.

AWS Security Hub CSPM uses security controls to evaluate resource configurations and security standards to help you comply with various compliance frameworks. For more information about using Security Hub CSPM to evaluate RDS resources, see Amazon Relational Database Service controls in the AWS Security Hub User Guide.

You can monitor your usage of RDS as it relates to security best practices by using Security Hub CSPM. For more information, see What is AWS Security Hub CSPM?.

Use the AWS Management Console, the AWS CLI, or the RDS API to change the password for your master user. If you use another tool, such as a SQL client, to change the master user password, it might result in privileges being revoked for the user unintentionally.

Controlling access with security groups

VPC security groups control the access that traffic has in and out of a DB instance . By default, network access is turned off for a DB instance . You can specify rules in a security group that allow access from an IP address range, port, or security group. After ingress rules are configured, the same rules apply to all DB instances that are associated with that security group. You can specify up to 20 rules in a security group.

Overview of VPC security groups

Each VPC security group rule makes it possible for a specific source to access a DB instance in a VPC that is associated with that VPC security group. The source can be a range of addresses (for example, 203.0.113.0/24), or another VPC security group. By specifying a VPC security group as the source, you allow incoming traffic from all instances (typically application servers) that use the source VPC security group. VPC security groups can have rules that govern both inbound and outbound traffic. However, the outbound traffic rules typically don't apply to DB instances . Outbound traffic rules apply only if the DB instance acts as a client. For example, outbound traffic rules apply to an Oracle DB instance with outbound database links. You must use the Amazon EC2 API or the **Security Group** option on the VPC console to create VPC security groups.

When you create rules for your VPC security group that allow access to the instances in your VPC, you must specify a port for each range of addresses that the rule allows access for. For example, if you want to turn on Secure Shell (SSH) access for instances in the VPC, create a rule allowing access to TCP port 22 for the specified range of addresses.

You can configure multiple VPC security groups that allow access to different ports for different instances in your VPC. For example, you can create a VPC security group that allows access to TCP port 80 for web servers in your VPC. You can then create another VPC security group that allows access to TCP port 3306 for RDS for MySQL DB instances in your VPC.

For more information on VPC security groups, see Security groups in the *Amazon Virtual Private Cloud User Guide*.

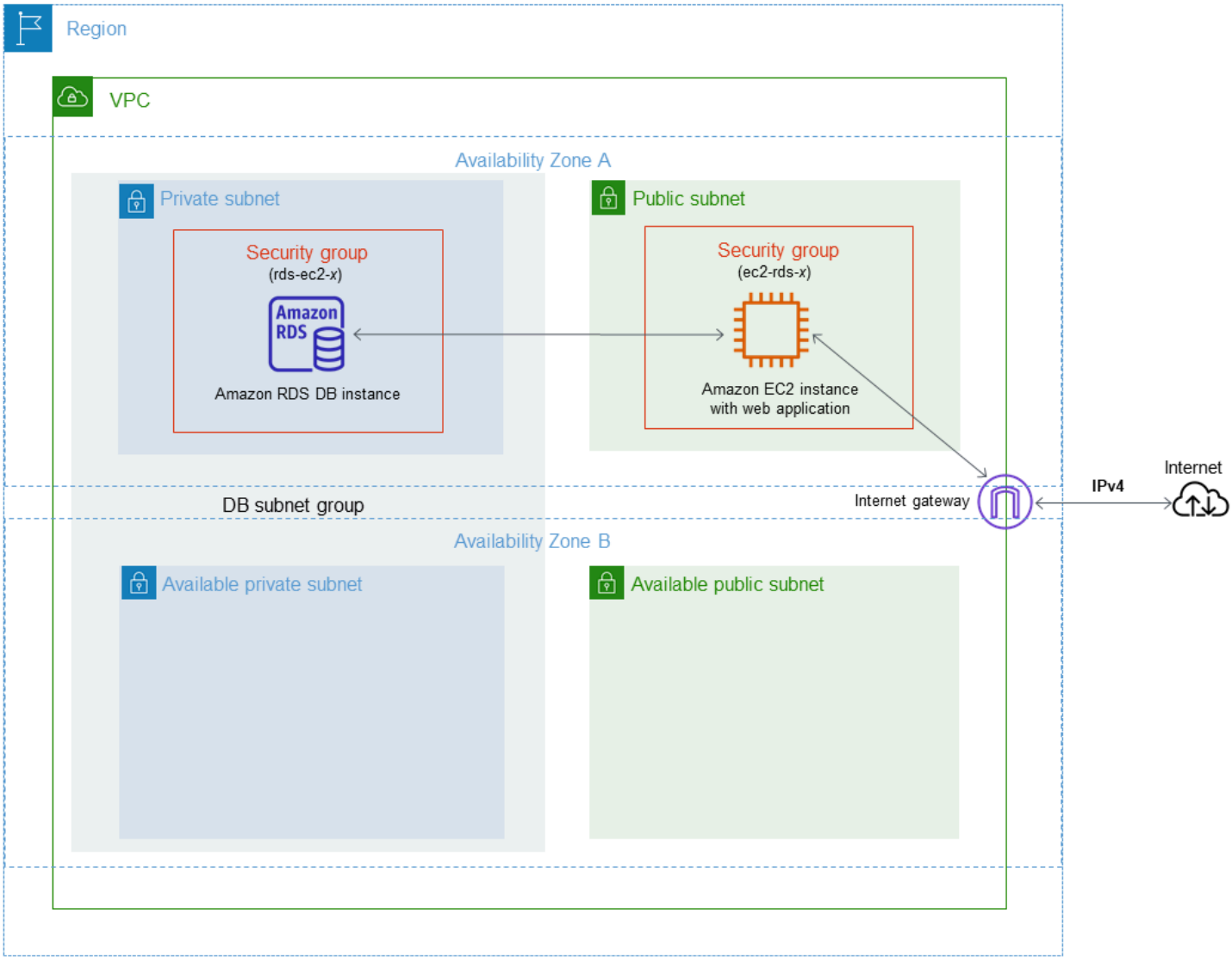
> **Note**
>
> If your DB instance is in a VPC but isn't publicly accessible, you can also use an AWS Site-to-Site VPN connection or an Direct Connect connection to access it from a private network. For more information, see Internetwork traffic privacy .

Security group scenario

A common use of a DB instance in a VPC is to share data with an application server running in an Amazon EC2 instance in the same VPC, which is accessed by a client application outside the VPC. For this scenario, you use the RDS and VPC pages on the AWS Management Console or the RDS and EC2 API operations to create the necessary instances and security groups:

1. Create a VPC security group (for example, `sg-0123ec2example`) and define inbound rules that use the IP addresses of the client application as the source. This security group allows your client application to connect to EC2 instances in a VPC that uses this security group.
2. Create an EC2 instance for the application and add the EC2 instance to the VPC security group (`sg-0123ec2example`) that you created in the previous step.
3. Create a second VPC security group (for example, `sg-6789rdsexample`) and create a new rule by specifying the VPC security group that you created in step 1 (`sg-0123ec2example`) as the source.
4. Create a new DB instance and add the DB instance to the VPC security group (`sg-6789rdsexample`) that you created in the previous step. When you create the DB instance , use the same port number as the one specified for the VPC security group (`sg-6789rdsexample`) rule that you created in step 3.

The following diagram shows this scenario.



For detailed instructions about configuring a VPC for this scenario, see Tutorial: Create a VPC for use with a DB instance (IPv4 only) . For more information about using a VPC, see Amazon VPC and Amazon RDS .

Creating a VPC security group

You can create a VPC security group for a DB instance by using the VPC console. For information about creating a security group, see Provide access to your DB instance in your VPC by creating a security group and Security groups in the *Amazon Virtual Private Cloud User Guide*.

Associating a security group with a DB instance

You can associate a security group with a DB instance by using **Modify** on the RDS console, the `ModifyDBInstance` Amazon RDS API, or the `modify-db-instance` AWS CLI command.

The following CLI example associates a specific VPC security group and removes DB security groups from the DB instance

```
aws rds modify-db-instance --db-instance-identifier dbName --vpc-security-group-ids sg-ID
```

For information about modifying a DB instance, see Modifying an Amazon RDS DB instance . For security group considerations when you restore a DB instance from a DB snapshot, see Security group considerations .

> **Note**
>
> The RDS console displays different security group rule names for your database if the Port value is configured to a non-default value.

For RDS for Oracle DB instances, additional security groups can be associated by populating the security group options setting for the Oracle Enterprise Manager Database Express (OEM), Oracle Management Agent for Enterprise Manager Cloud Control (OEM Agent) and the Oracle Secure Sockets Layer options. In this case, both security groups associated with the DB instance and options settings apply to the DB instance. For more information about these option groups, see Oracle Enterprise Manager ,Oracle Management Agent for Enterprise Manager Cloud Control , and Oracle Secure Sockets Layer .

Master user account privileges

When you create a new DB instance , the default master user that you use gets certain privileges for that DB instance . You can't change the master user name after the DB instance is created.

Important

We strongly recommend that you do not use the master user directly in your applications. Instead, adhere to the best practice of using a database user created with the minimal privileges required for your application.

Note

If you accidentally delete the permissions for the master user, you can restore them by modifying the DB instance and setting a new master user password. For more information about modifying a DB instance , see Modifying an Amazon RDS DB instance .

The following table shows the privileges and database roles the master user gets for each of the database engines.

Database engine	System privilege	Database role
RDS for Db2	The master user is assigned to the `masterdba` group and assigned the `master_user_role` . `SYSMON,DBADM with DATAACCESS AND ACCCESSCTRL , BINDADD,CONNECT, CREATETAB ,CREATE_SECURE_OBJECT , EXPLAIN,IMPLICIT_SCHEMA , LOAD,SQLADM,WLMADM`	`DBA,DBA_RESTRICTED , DEVELOPER ,ROLE_NULLID_PACKAGES , ROLE_PROCEDURES ,ROLE_TABLESPACES` For more information, see Amazon RDS for Db2 default roles .
RDS for MariaDB	`SELECT,INSERT,UPDATE,DELETE, CREATE,DROP,RELOAD, PROCESS,REFERENCES ,INDEX, ALTER,SHOW DATABASES ,CREATE TEMPORARY TABLES ,LOCK TABLES, EXECUTE,REPLICATION CLIENT,CREATE VIEW ,SHOW VIEW,CREATE`	—

Database engine	System privilege	Database role
	`ROUTINE, ALTER ROUTINE,CREATE USER, EVENT,TRIGGER,REPLICATION SLAVE` Starting with RDS for MariaDB version 11.4, the master user also gets the `SHOW CREATE ROUTINE` privilege.	
RDS for MySQL 8.0.36 and higher	`SELECT,INSERT,UPDATE, DELETE,CREATE,DROP, RELOAD,PROCESS, REFERENCES ,INDEX,ALTER, SHOW DATABASES ,CREATE TEMPORARY TABLES,LOCK TABLES,EXECUTE, REPLICATI ON SLAVE ,REPLICATION CLIENT , CREATE VIEW,SHOW VIEW,CREATE ROUTINE,ALTER ROUTINE,CREATE USER,EVENT,TRIGGER, CREATE ROLE,DROP ROLE, APPLICATION_PASSWO RD_ADMIN , ROLE_ADMIN ,SET_USER_ID , XA_RECOVER_ADMIN`	`rds_superuser_role` For more information about `rds_superuser_role` , see Role-based privilege model for RDS for MySQL .
RDS for MySQL versions lower than 8.0.36	`SELECT,INSERT,UPDATE, DELETE,CREATE,DROP, RELOAD,PROCESS, REFERENCES ,INDEX,ALTER, SHOW DATABASES ,CREATE TEMPORARY TABLES,LOCK TABLES,EXECUTE, REPLICATI ON CLIENT ,CREATE VIEW, SHOW VIEW,CREATE ROUTINE,ALTER ROUTINE,CREATE USER,EVENT, TRIGGER,REPLICATION SLAVE`	—

Database engine	System privilege	Database role
RDS for PostgreSQL	CREATE ROLE,CREATE DB, PASSWORD VALID UNTIL INFINITY,CREATE EXTENSION ,ALTER EXTENSION ,DROP EXTENSION ,CREATE TABLESPACE ,ALTER <OBJECT> OWNER,CHECKPOINT , PG_CANCEL_BACKEND( ) , PG_TERMINATE_BACKEND() ,SELECT PG_STAT_REPLICATION ,EXECUTE PG_STAT_STATEMENTS_RESET() ,OWN POSTGRES_FDW_HANDLER() ,OWN POSTGRES_FDW_VALIDATOR() ,OWN POSTGRES_FDW , EXECUTE PG_BUFFERCACHE_PAGES() ,SELECT PG_BUFFERCACHE	RDS_SUPERUSER For more information about RDS_SUPERUSER, see Understanding PostgreSQL roles and permissions .
RDS for Oracle	ADMINISTER DATABASE TRIGGER ,ALTER DATABASE LINK,ALTER PUBLIC DATABASE LINK, AUDIT SYSTEM,CHANGE NOTIFICATION , DROP ANY DIRECTORY ,EXEMPT ACCESS POLICY,EXEMPT IDENTITY POLICY,EXEMPT REDACTION POLICY,FLASHBACK ANY TABLE, GRANT ANY OBJECT PRIVILEGE ,RESTRICTED SESSION ,SELECT ANY TABLE,UNLIMITED TABLESPACE	DBA **Note** The DBA role is exempt from the following privileges: ALTER DATABASE,ALTER SYSTEM, CREATE ANY DIRECTORY ,CREATE EXTERNAL JOB,CREATE PLUGGABLE DATABASE, GRANT ANY PRIVILEGE ,GRANT ANY ROLE,READ ANY FILE GROUP

Database engine	System privilege	Database role
Amazon RDS for Microsoft SQL Server	ADMINISTER BULK OPERATIONS ,ALTER ANY CONNECTION ,ALTER ANY CREDENTIAL , ALTER ANY EVENT SESSION,ALTER ANY LINKED SERVER,ALTER ANY LOGIN,ALTER ANY SERVER AUDIT,ALTER ANY SERVER ROLE, ALTER SERVER STATE,ALTER TRACE, CONNECT SQL,CREATE ANY DATABASE, VIEW ANY DATABASE,VIEW ANY DEFINITION ,VIEW SERVER STATE,ALTER ON ROLE SQLAgentOperatorRole	DB_OWNER (database-level role), PROCESSADMIN (server-level role), SETUPADMIN (server-level role), SQLAgentUserRole (database-level role), SQLAgentReaderRole (database-level role), and SQLAgentOperatorRole (database-level role)

Using service-linked roles for Amazon RDS

Amazon RDS uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Amazon RDS. Service-linked roles are predefined by Amazon RDS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes using Amazon RDS easier because you don't have to manually add the necessary permissions. Amazon RDS defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon RDS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This protects your Amazon RDS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS services that work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon RDS

Amazon RDS uses the service-linked role named AWSServiceRoleForRDS to allow Amazon RDS to call AWS services on behalf of your DB instances.

The AWSServiceRoleForRDS service-linked role trusts the following services to assume the role:

- `rds.amazonaws.com`

This service-linked role has a permissions policy attached to it called `AmazonRDSServiceRolePolicy` that grants it permissions to operate in your account.

For more information about this policy, including the JSON policy document, see AmazonRDSServiceRolePolicy in the *AWS Managed Policy Reference Guide*.

> **Note**
>
> You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. If you encounter the following error message:

Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.
Make sure you have the following permissions enabled:

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"rds.amazonaws.com"
        }
    }
}
```

For more information, see Service-linked role permissions in the *IAM User Guide*.

Creating a service-linked role for Amazon RDS

You don't need to manually create a service-linked role. When you create a DB instance, Amazon RDS creates the service-linked role for you.

Important

If you were using the Amazon RDS service before December 1, 2017, when it began supporting service-linked roles, then Amazon RDS created the AWSServiceRoleForRDS role in your account. To learn more, see A new role appeared in my AWS account.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a DB instance, Amazon RDS creates the service-linked role for you again.

Editing a service-linked role for Amazon RDS

Amazon RDS does not allow you to edit the AWSServiceRoleForRDS service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities

might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon RDS

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must delete all of your DB instances before you can delete the service-linked role.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**. Then choose the name (not the check box) of the AWSServiceRoleForRDS role.
3. On the **Summary** page for the chosen role, choose the **Last Accessed** tab.
4. On the **Last Accessed** tab, review recent activity for the service-linked role.

> **Note**
>
> If you are unsure whether Amazon RDS is using the AWSServiceRoleForRDS role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the AWS Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

If you want to remove the AWSServiceRoleForRDS role, you must first delete *all* of your DB instances .

Deleting all of your instances

Use one of these procedures to delete each of your instances.

To delete an instance (console)

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Databases**.
3. Choose the instance that you want to delete.
4. For **Actions**, choose **Delete**.
5. If you are prompted for **Create final Snapshot?**, choose **Yes** or **No**.
6. If you chose **Yes** in the previous step, for **Final snapshot name** enter the name of your final snapshot.
7. Choose **Delete**.

To delete an instance (CLI)

See `delete-db-instance` in the *AWS CLI Command Reference*.

To delete an instance (API)

See `DeleteDBInstance` in the *Amazon RDS API Reference*.

You can use the IAM console, the IAM CLI, or the IAM API to delete the AWSServiceRoleForRDS service-linked role. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Service-linked role permissions for Amazon RDS Custom

Amazon RDS Custom uses the service-linked role named `AWSServiceRoleForRDSCustom` to allow RDS Custom to call AWS services on behalf of your RDS DB resources.

The AWSServiceRoleForRDSCustom service-linked role trusts the following services to assume the role:

- `custom.rds.amazonaws.com`

This service-linked role has a permissions policy attached to it called `AmazonRDSCustomServiceRolePolicy` that grants it permissions to operate in your account.

Creating, editing, or deleting the service-linked role for RDS Custom works the same as for Amazon RDS. For more information, see AWS managed policy: AmazonRDSCustomServiceRolePolicy.

Note

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. If you encounter the following error message: **Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.**
Make sure you have the following permissions enabled:

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
AmazonRDSCustomServiceRolePolicy",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"custom.rds.amazonaws.com"
        }
    }
}
```

For more information, see Service-linked role permissions in the *IAM User Guide*.

Service-linked role permissions for Amazon RDS Beta

Amazon RDS uses the service-linked role named `AWSServiceRoleForRDSBeta` to allow Amazon RDS to call AWS services on behalf of your RDS DB resources.

The AWSServiceRoleForRDSBeta service-linked role trusts the following services to assume the role:

- `rds.amazonaws.com`

This service-linked role has a permissions policy attached to it called `AmazonRDSBetaServiceRolePolicy` that grants it permissions to operate in your account. For more information, see AWS managed policy: AmazonRDSBetaServiceRolePolicy.

Note

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. If you encounter the following error message: **Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.**
Make sure you have the following permissions enabled:

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
AmazonRDSBetaServiceRolePolicy",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"custom.rds.amazonaws.com"
        }
    }
}
```

For more information, see Service-linked role permissions in the *IAM User Guide*.

Service-linked role for Amazon RDS Preview

Amazon RDS uses the service-linked role named `AWSServiceRoleForRDSPreview` to allow Amazon RDS to call AWS services on behalf of your RDS DB resources.

The AWSServiceRoleForRDSPreview service-linked role trusts the following services to assume the role:

- `rds.amazonaws.com`

This service-linked role has a permissions policy attached to it called `AmazonRDSPreviewServiceRolePolicy` that grants it permissions to operate in your account. For more information, see AWS managed policy: AmazonRDSPreviewServiceRolePolicy.

Note

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. If you encounter the following error message: **Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.**
Make sure you have the following permissions enabled:

```
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/custom.rds.amazonaws.com/
AmazonRDSPreviewServiceRolePolicy",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"custom.rds.amazonaws.com"
        }
    }
}
```

For more information, see Service-linked role permissions in the *IAM User Guide*.

Amazon VPC and Amazon RDS

Amazon Virtual Private Cloud (Amazon VPC) makes it possible for you to launch AWS resources, such as Amazon RDS DB instances, into a virtual private cloud (VPC).

When you use a VPC, you have control over your virtual networking environment. You can choose your own IP address range, create subnets, and configure routing and access control lists. There is no additional cost to run your DB instance in a VPC.

Accounts have a default VPC. All new DB instances are created in the default VPC unless you specify otherwise.

Topics

- Working with a DB instance in a VPC
- Updating the VPC for a DB instance
- Scenarios for accessing a DB instance in a VPC
- Tutorial: Create a VPC for use with a DB instance (IPv4 only)
- Tutorial: Create a VPC for use with a DB instance (dual-stack mode)
- Moving a DB instance not in a VPC into a VPC

Following, you can find a discussion about VPC functionality relevant to Amazon RDS DB instances. For more information about Amazon VPC, see Amazon VPC Getting Started Guide and Amazon VPC User Guide.

Working with a DB instance in a VPC

Your DB instance is in a virtual private cloud (VPC). A VPC is a virtual network that is logically isolated from other virtual networks in the AWS Cloud. Amazon VPC makes it possible for you to launch AWS resources, such as an Amazon RDS DB instance or Amazon EC2 instance, into a VPC. The VPC can either be a default VPC that comes with your account or one that you create. All VPCs are associated with your AWS account.

Your default VPC has three subnets that you can use to isolate resources inside the VPC. The default VPC also has an internet gateway that can be used to provide access to resources inside the VPC from outside the VPC.

For a list of scenarios involving Amazon RDS DB instances in a VPC and outside of a VPC, see Scenarios for accessing a DB instance in a VPC.

Topics

- Working with a DB instance in a VPC
- VPC encryption control
- Working with DB subnet groups
- Shared subnets
- Amazon RDS IP addressing
- Hiding a DB instance in a VPC from the internet
- Creating a DB instance in a VPC

In the following tutorials, you can learn to create a VPC that you can use for a common Amazon RDS scenario:

- Tutorial: Create a VPC for use with a DB instance (IPv4 only)
- Tutorial: Create a VPC for use with a DB instance (dual-stack mode)

Working with a DB instance in a VPC

Here are some tips on working with a DB instance in a VPC:

- Your VPC must have at least two subnets. These subnets must be in two different Availability Zones in the AWS Region where you want to deploy your DB instance. A *subnet* is a segment of a VPC's IP address range that you can specify and that you can use to group DB instances based on your security and operational needs.

  For Multi-AZ deployments, defining a subnet for two or more Availability Zones in an AWS Region allows Amazon RDS to create a new standby in another Availability Zone as needed. Make sure to do this even for Single-AZ deployments, just in case you want to convert them to Multi-AZ deployments at some point.

  > **Note**
  >
  > The DB subnet group for a Local Zone can have only one subnet.

- If you want your DB instance in the VPC to be publicly accessible, make sure to turn on the VPC attributes *DNS hostnames* and *DNS resolution*.

- Your VPC must have a DB subnet group that you create. You create a DB subnet group by specifying the subnets you created. Amazon RDS chooses a subnet and an IP address within that subnet group to associate with your DB instance. The DB instance uses the Availability Zone that contains the subnet.
- Your VPC must have a VPC security group that allows access to the DB instance.

  For more information, see Scenarios for accessing a DB instance in a VPC.
- The CIDR blocks in each of your subnets must be large enough to accommodate spare IP addresses for Amazon RDS to use during maintenance activities, including failover and compute scaling. For example, a range such as 10.0.0.0/24 and 10.0.1.0/24 is typically large enough.
- A VPC can have an *instance tenancy* attribute of either *default* or *dedicated*. All default VPCs have the instance tenancy attribute set to default, and a default VPC can support any DB instance class.

  If you choose to have your DB instance in a dedicated VPC where the instance tenancy attribute is set to dedicated, the DB instance class of your DB instance must be one of the approved Amazon EC2 dedicated instance types. For example, the r5.large EC2 dedicated instance corresponds to the db.r5.large DB instance class. For information about instance tenancy in a VPC, see Dedicated instances in the *Amazon Elastic Compute Cloud User Guide*.

  For more information about the instance types that can be in a dedicated instance, see Amazon EC2 dedicated instances on the Amazon EC2 pricing page.

  > **Note**
  >
  > When you set the instance tenancy attribute to dedicated for a DB instance, it doesn't guarantee that the DB instance will run on a dedicated host.

- When an option group is assigned to a DB instance, it's associated with the DB instance's VPC. This linkage means that you can't use the option group assigned to a DB instance if you attempt to restore the DB instance into a different VPC.
- If you restore a DB instance into a different VPC, make sure to either assign the default option group to the DB instance, assign an option group that is linked to that VPC, or create a new option group and assign it to the DB instance. With persistent or permanent options, such as Oracle TDE, you must create a new option group that includes the persistent or permanent option when restoring a DB instance into a different VPC.

VPC encryption control

VPC encryption controls allow you to enforce encryption-in-transit for all network traffic within your VPCs. Use encryption control to meet regulatory compliance requirements by ensuring that only encryption-capable Nitro-based hardware can be provisioned in designated VPCs. Encryption control also catches compatibility issues at API request time rather than during provisioning. Your existing workloads continue operating and only new incompatible requests are blocked.

Set your VPC encryption controls by setting the VPC control mode to :

- *disabled* (default)
- *monitor*
- *enforced*

To check the current control mode for your VPC, use the AWS Management Console or DescribeVpcs CLI or API command.

If your VPC enforces encryption, you can only provision Nitro-based DB instances that support encryption in transit in that VPC. For more information see, DB instance class types. For information about Nitro instances, see Instances built on the AWS Nitro System in the *Amazon EC2 User Guide*.

> **Note**
>
> If you try to provision incompatible DB instances in an encryption-enforced VPC, Amazon RDS returns a `VpcEncryptionControlViolationException` exception.

Working with DB subnet groups

Subnets are segments of a VPC's IP address range that you designate to group your resources based on security and operational needs. A *DB subnet group* is a collection of subnets (typically private) that you create in a VPC and that you then designate for your DB instances. By using a DB subnet group, you can specify a particular VPC when creating DB instances using the AWS CLI or RDS API. If you use the console, you can choose the VPC and subnet groups you want to use.

Each DB subnet group should have subnets in at least two Availability Zones in a given AWS Region. When creating a DB instance in a VPC, you choose a DB subnet group for it. From the DB

subnet group, Amazon RDS chooses a subnet and an IP address within that subnet to associate with the DB instance. The DB uses the Availability Zone that contains the subnet. Amazon RDS always assigns an IP address from a subnet that has free IP address space.

If the primary DB instance of a Multi-AZ deployment fails, Amazon RDS can promote the corresponding standby and later create a new standby using an IP address of the subnet in one of the other Availability Zones.

The subnets in a DB subnet group are either public or private. The subnets are public or private, depending on the configuration that you set for their network access control lists (network ACLs) and routing tables. For a DB instance to be publicly accessible, all of the subnets in its DB subnet group must be public. If a subnet that's associated with a publicly accessible DB instance changes from public to private, it can affect DB instance availability.

To create a DB subnet group that supports dual-stack mode, make sure that each subnet that you add to the DB subnet group has an Internet Protocol version 6 (IPv6) CIDR block associated with it. For more information, see Amazon RDS IP addressing and Migrating to IPv6 in the *Amazon VPC User Guide.*

> **Note**
>
> The DB subnet group for a Local Zone can have only one subnet.

When Amazon RDS creates a DB instance in a VPC, it assigns a network interface to your DB instance by using an IP address from your DB subnet group. However, we strongly recommend that you use the Domain Name System (DNS) name to connect to your DB instance. We recommend this because the underlying IP address changes during failover.

> **Note**
>
> For each DB instance that you run in a VPC, make sure to reserve at least one address in each subnet in the DB subnet group for use by Amazon RDS for recovery actions.

Shared subnets

You can create a DB instance in a shared VPC.

Some considerations to keep in mind while using shared VPCs:

- You can move a DB instance from a shared VPC subnet to a non-shared VPC subnet and vice-versa.
- Participants in a shared VPC must create a security group in the VPC to allow them to create a DB instance.
- Owners and participants in a shared VPC can access the database by using SQL queries. However, only the creator of a resource can make any API calls on the resource.

Amazon RDS IP addressing

IP addresses enable resources in your VPC to communicate with each other, and with resources over the internet. Amazon RDS supports both IPv4 and IPv6 addressing protocols. By default, Amazon RDS and Amazon VPC use the IPv4 addressing protocol. You can't turn off this behavior. When you create a VPC, make sure to specify an IPv4 CIDR block (a range of private IPv4 addresses). You can optionally assign an IPv6 CIDR block to your VPC and subnets, and assign IPv6 addresses from that block to DB instances in your subnet.

Support for the IPv6 protocol expands the number of supported IP addresses. By using the IPv6 protocol, you ensure that you have sufficient available addresses for the future growth of the internet. New and existing RDS resources can use IPv4 and IPv6 addresses within your VPC. Configuring, securing, and translating network traffic between the two protocols used in different parts of an application can cause operational overhead. You can standardize on the IPv6 protocol for Amazon RDS resources to simplify your network configuration.

Topics

- IPv4 addresses
- IPv6 addresses
- Dual-stack mode

IPv4 addresses

When you create a VPC, you must specify a range of IPv4 addresses for the VPC in the form of a CIDR block, such as `10.0.0.0/16`. A *DB subnet group* defines the range of IP addresses in this CIDR block that a DB instance can use. These IP addresses can be private or public.

A private IPv4 address is an IP address that's not reachable over the internet. You can use private IPv4 addresses for communication between your DB instance and other resources, such as Amazon

EC2 instances, in the same VPC. Each DB instance has a private IP address for communication in the VPC.

A public IP address is an IPv4 address that's reachable from the internet. You can use public addresses for communication between your DB instance and resources on the internet, such as a SQL client. You control whether your DB instance receives a public IP address.

For a tutorial that shows you how to create a VPC with only private IPv4 addresses that you can use for a common Amazon RDS scenario, see Tutorial: Create a VPC for use with a DB instance (IPv4 only).

IPv6 addresses

You can optionally associate an IPv6 CIDR block with your VPC and subnets, and assign IPv6 addresses from that block to the resources in your VPC. Each IPv6 address is globally unique.

The IPv6 CIDR block for your VPC is automatically assigned from Amazon's pool of IPv6 addresses. You can't choose the range yourself.

When connecting to an IPv6 address, make sure that the following conditions are met:

- The client is configured so that client to database traffic over IPv6 is allowed.
- RDS security groups used by the DB instance are configured correctly so that client to database traffic over IPv6 is allowed.
- The client operating system stack allows traffic on the IPv6 address, and operating system drivers and libraries are configured to choose the correct default DB instance endpoint (either IPv4 or IPv6).

For more information about IPv6, see IP Addressing in the *Amazon VPC User Guide*.

Dual-stack mode

A DB instance runs in dual-stack mode when it can communicate over both IPv4 and IPv6 addressing protocols. Resources can then communicate with the DB instance using either IPv4, IPv6, or both protocols. Private dual-stack mode DB instances have IPv6 endpoints that RDS restricts to VPC access only, ensuring your IPv6 endpoints remain private. Public dual-stack mode DB instances provide both IPv4 and IPv6 endpoints that you can access from the internet.

Topics

- Dual-stack mode and DB subnet groups

- Working with dual-stack mode DB instances
- Modifying IPv4-only DB instances to use dual-stack mode
- Region and version availability
- Limitations for dual-stack network DB instances

For a tutorial that shows you how to create a VPC with both IPv4 and IPv6 addresses that you can use for a common Amazon RDS scenario, see Tutorial: Create a VPC for use with a DB instance (dual-stack mode).

Dual-stack mode and DB subnet groups

To use dual-stack mode, make sure that each subnet in the DB subnet group that you associate with the DB instance has an IPv6 CIDR block associated with it. You can create a new DB subnet group or modify an existing DB subnet group to meet this requirement. After a DB instance is in dual-stack mode, clients can connect to it normally. Make sure that client security firewalls and RDS DB instance security groups are accurately configured to allow traffic over IPv6. To connect, clients use the DB instance's endpoint. Client applications can specify which protocol is preferred when connecting to a database. In dual-stack mode, the DB instance detects the client's preferred network protocol, either IPv4 or IPv6, and uses that protocol for the connection.

If a DB subnet group stops supporting dual-stack mode because of subnet deletion or CIDR disassociation, there's a risk of an incompatible network state for DB instances that are associated with the DB subnet group. Also, you can't use the DB subnet group when you create a new dual-stack mode DB instance.

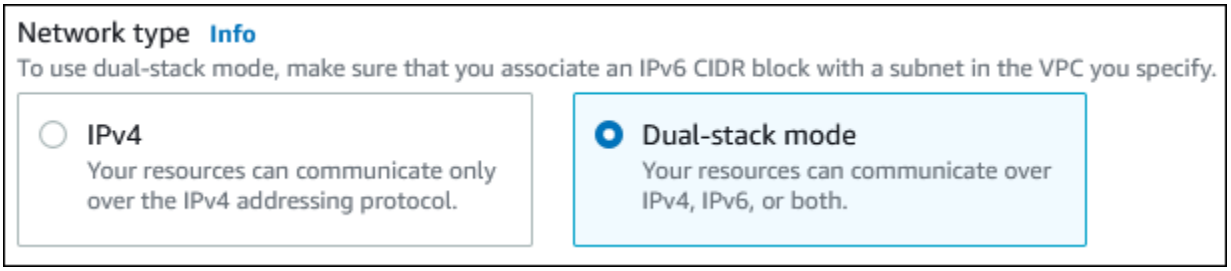
To determine whether a DB subnet group supports dual-stack mode by using the AWS Management Console, view the **Network type** on the details page of the DB subnet group. To determine whether a DB subnet group supports dual-stack mode by using the AWS CLI, run the describe-db-subnet-groups command and view `SupportedNetworkTypes` in the output.

Read replicas are treated as independent DB instances and can have a network type that's different from the primary DB instance. If you change the network type of a read replica's primary DB instance, the read replica isn't affected. When you are restoring a DB instance, you can restore it to any network type that's supported.

Working with dual-stack mode DB instances

When you create or modify a DB instance, you can specify dual-stack mode to allow your resources to communicate with your DB instance over IPv4, IPv6, or both.

When you use the AWS Management Console to create or modify a DB instance, you can specify dual-stack mode in the **Network type** section. The following image shows the **Network type** section in the console.



When you use the AWS CLI to create or modify a DB instance, set the `--network-type` option to `DUAL` to use dual-stack mode. When you use the RDS API to create or modify a DB instance, set the `NetworkType` parameter to `DUAL` to use dual-stack mode. When you are modifying the network type of a DB instance, downtime is possible. If dual-stack mode isn't supported by the specified DB engine version or DB subnet group, the `NetworkTypeNotSupported` error is returned.

For more information about creating a DB instance, see Creating an Amazon RDS DB instance. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

To determine whether a DB instance is in dual-stack mode by using the console, view the **Network type** on the **Connectivity & security** tab for the DB instance.

Modifying IPv4-only DB instances to use dual-stack mode

You can modify an IPv4-only DB instance to use dual-stack mode. To do so, change the network type of the DB instance. The modification might result in downtime.

It is recommended that you change the network type of your Amazon RDS DB instances during a maintenance window. Currently, setting the network type of new instances to dual-stack mode isn't supported. You can set network type manually by using the `modify-db-instance` command.

Before modifying a DB instance to use dual-stack mode, make sure that its DB subnet group supports dual-stack mode. If the DB subnet group associated with the DB instance doesn't support dual-stack mode, specify a different DB subnet group that supports it when you modify the DB instance. Modifying the DB subnet group of a DB instance can cause downtime.

If you modify the DB subnet group of a DB instance before you change the DB instance to use dual-stack mode, make sure that the DB subnet group is valid for the DB instance before and after the change.

For RDS for PostgreSQL, RDS for MySQL, RDS for Oracle, and RDS for MariaDB Single-AZ instances, we recommend that you run the modify-db-instance command with only the `--network-type` parameter set to `DUAL` to change the network to dual-stack mode. Adding other parameters along with the `--network-type` parameter in the same API call could result in downtime. To modify multiple parameters, ensure that the network type modification is successfully completed before sending another `modify-db-instance` request with other parameters.

Network type modifications for RDS for PostgreSQL, RDS for MySQL, RDS for Oracle, and RDS for MariaDB Multi-AZ DB instances cause a brief downtime and trigger a failover if you only use the `--network-type` parameter or if you combine parameters in a modify-db-instance command.

Network type modifications on RDS for SQL Server Single-AZ or Multi-AZ DB instances cause downtime if you only use the `--network-type` parameter or if you combine parameters in a `modify-db-instance` command. Network type modifications cause failover in an SQL Server Multi-AZ instance.

If you can't connect to the DB instance after the change, make sure that the client and database security firewalls and route tables are accurately configured to allow traffic to the database on the selected network (either IPv4 or IPv6). You might also need to modify operating system parameter, libraries, or drivers to connect using an IPv6 address.

When you modify a DB instance to use dual-stack mode, there can't be a pending change from a Single-AZ deployment to a Multi-AZ deployment, or from a Multi-AZ deployment to a Single-AZ deployment.

To modify an IPv4-only DB instance to use dual-stack mode

1. Modify a DB subnet group to support dual-stack mode, or create a DB subnet group that supports dual-stack mode:

   a. Associate an IPv6 CIDR block with your VPC.

      For instructions, see Add an IPv6 CIDR block to your VPC in the *Amazon VPC User Guide*.

   b. Attach the IPv6 CIDR block to all of the subnets in your the DB subnet group.

      For instructions, see Add an IPv6 CIDR block to your subnet in the *Amazon VPC User Guide*.

   c. Confirm that the DB subnet group supports dual-stack mode.

If you are using the AWS Management Console, select the DB subnet group, and make sure that the **Supported network types** value is **Dual, IPv4**.

If you are using the AWS CLI, run the describe-db-subnet-groups command, and make sure that the `SupportedNetworkType` value for the DB instance is `Dual, IPv4`.

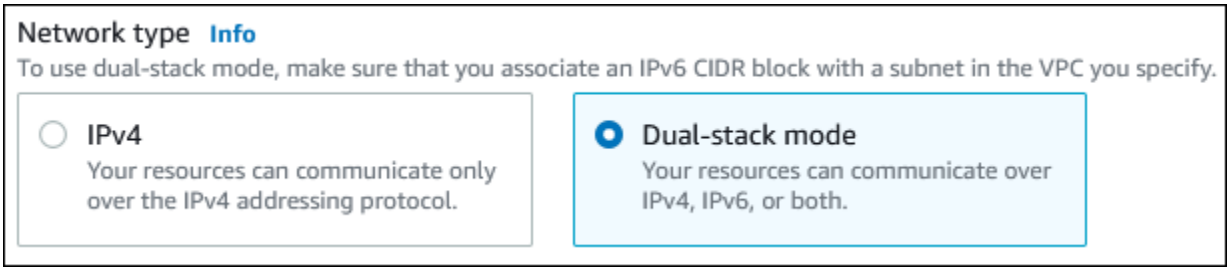
2. Modify the security group associated with the DB instance to allow IPv6 connections to the database, or create a new security group that allows IPv6 connections.

   For instructions, see Security group rules in the *Amazon VPC User Guide*.

3. Modify the DB instance to support dual-stack mode. To do so, set the **Network type** to **Dual-stack mode**.

   If you are using the console, make sure that the following settings are correct:

   - **Network type** – **Dual-stack mode**



   - **DB subnet group** – The DB subnet group that you configured in a previous step
   - **Security group** – The security that you configured in a previous step

   If you are using the AWS CLI, make sure that the following settings are correct:

   - `--network-type` – `dual`
   - `--db-subnet-group-name` – The DB subnet group that you configured in a previous step
   - `--vpc-security-group-ids` – The VPC security group that you configured in a previous step

   For example:

```
aws rds modify-db-instance --db-instance-identifier my-instance --network-type
 "DUAL"
```

4. Confirm that the DB instance supports dual-stack mode.

   If you are using the console, choose the **Connectivity & security** tab for the DB instance. On that tab, make sure that the **Network type** value is **Dual-stack mode**.

   If you are using the AWS CLI, run the describe-db-instances command, and make sure that the `NetworkType` value for the DB instance is `dual`.

   Run the `dig` command on the DB instance endpoint to identify the IPv6 address associated with it.

   ```
   dig db-instance-endpoint AAAA
   ```

   Use the DB instance endpoint, not the IPv6 address, to connect to the DB instance.

Region and version availability

Feature availability and support varies across specific versions of each database engine, and across AWS Regions. For more information on version and Region availability with dual-stack mode, see Supported Regions and DB engines for dual-stack mode in Amazon RDS.

Limitations for dual-stack network DB instances

The following limitations apply to dual-stack network DB instances:

- DB instances can't use the IPv6 protocol exclusively. They can use IPv4 exclusively, or they can use the IPv4 and IPv6 protocol (dual-stack mode).
- Amazon RDS doesn't support native IPv6 subnets.
- For RDS for SQL Server, dual-stack mode DB instances that use Always On AGs availability group listener endpoints only present IPv4 addresses.
- You can't use RDS Proxy with dual-stack mode DB instances.
- You can't use dual-stack mode with RDS on AWS Outposts DB instances.
- You can't use dual-stack mode with DB instances in a Local Zone.

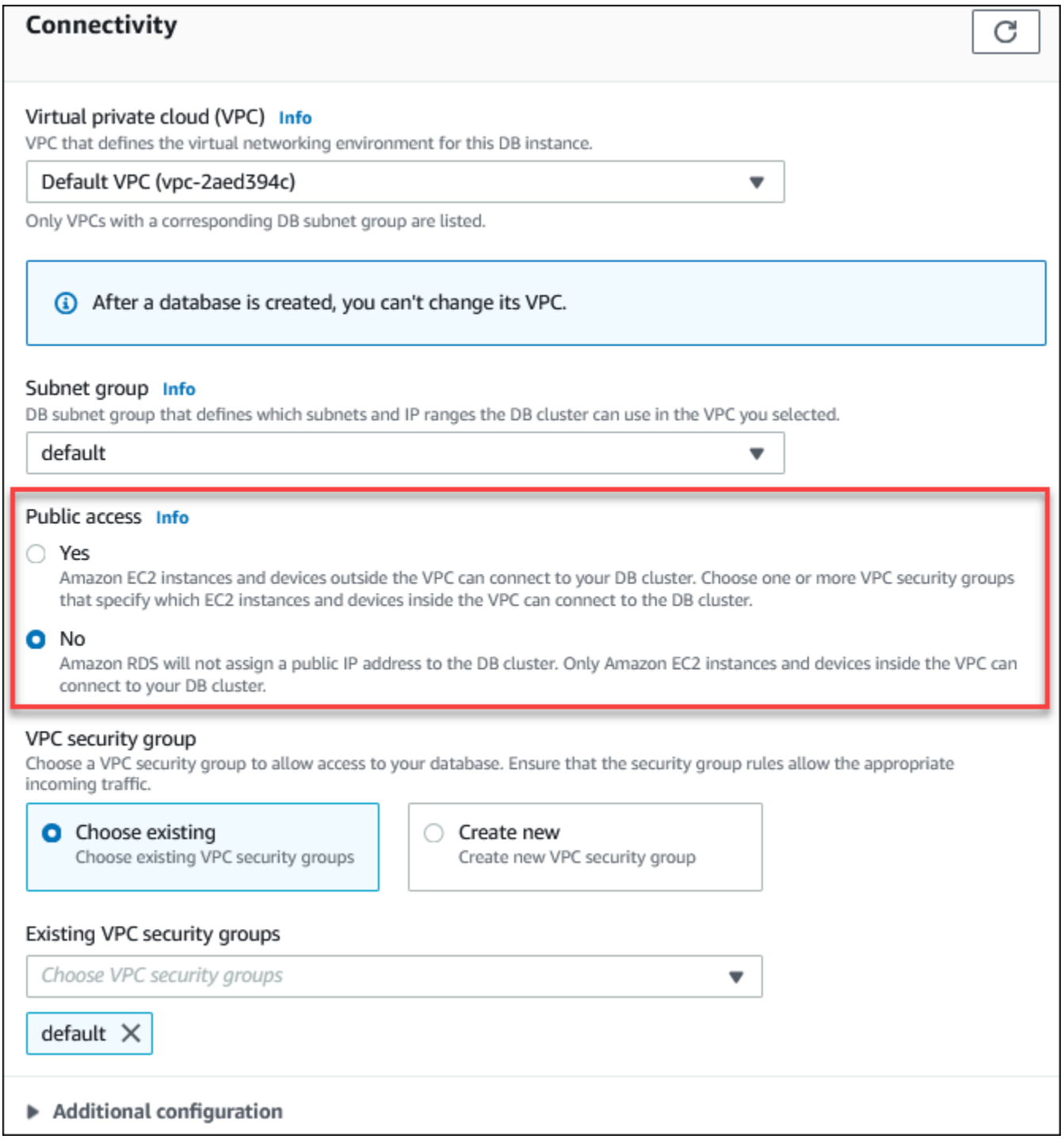
Hiding a DB instance in a VPC from the internet

One common Amazon RDS scenario is to have a VPC in which you have an Amazon EC2 instance with a public-facing web application and a DB instance with a database that isn't publicly

accessible. For example, you can create a VPC that has a public subnet and a private subnet. EC2 instances that function as web servers can be deployed in the public subnet. The DB instances are deployed in the private subnet. In such a deployment, only the web servers have access to the DB instances. For an illustration of this scenario, see A DB instance in a VPC accessed by an Amazon EC2 instance in the same VPC.

When you launch a DB instance inside a VPC, the DB instance has a private IP address for traffic inside the VPC. This private IP address isn't publicly accessible. You can use the **Public access** option to designate whether the DB instance also has a public IP address in addition to the private IP address. If the DB instance is designated as publicly accessible, its DNS endpoint resolves to the private IP address from within the VPC. It resolves to the public IP address from outside of the VPC. Access to the DB instance is ultimately controlled by the security group it uses. That public access is not permitted if the security group assigned to the DB instance doesn't include inbound rules that permit it. In addition, for a DB instance to be publicly accessible, the subnets in its DB subnet group must have an internet gateway. For more information, see Can't connect to Amazon RDS DB instance

You can modify a DB instance to turn on or off public accessibility by modifying the **Public access** option. The following illustration shows the **Public access** option in the **Additional connectivity configuration** section. To set the option, open the **Additional connectivity configuration** section in the **Connectivity** section.



For information about modifying a DB instance to set the **Public access** option, see Modifying an Amazon RDS DB instance.

Creating a DB instance in a VPC

The following procedures help you create a DB instance in a VPC. To use the default VPC, you can begin with step 2, and use the VPC and DB subnet group have already been created for you. If you want to create an additional VPC, you can create a new VPC.

> **Note**
>
> If you want your DB instance in the VPC to be publicly accessible, you must update the DNS information for the VPC by enabling the VPC attributes *DNS hostnames* and *DNS resolution*. For information about updating the DNS information for a VPC instance, see Updating DNS support for your VPC.

Follow these steps to create a DB instance in a VPC:

- Step 1: Create a VPC
- Step 2: Create a DB subnet group
- Step 3: Create a VPC security group
- Step 4: Create a DB instance in the VPC

Step 1: Create a VPC

Create a VPC with subnets in at least two Availability Zones. You use these subnets when you create a DB subnet group. If you have a default VPC, a subnet is automatically created for you in each Availability Zone in the AWS Region.

For more information, see Create a VPC with private and public subnets, or see Create a VPC in the *Amazon VPC User Guide*.

Step 2: Create a DB subnet group

A DB subnet group is a collection of subnets (typically private) that you create for a VPC and that you then designate for your DB instances. A DB subnet group allows you to specify a particular VPC when you create DB instances using the AWS CLI or RDS API. If you use the console, you can just choose the VPC and subnets you want to use. Each DB subnet group must have at least one subnet in at least two Availability Zones in the AWS Region. As a best practice, each DB subnet group should have at least one subnet for every Availability Zone in the AWS Region.

For Multi-AZ deployments, defining a subnet for all Availability Zones in an AWS Region enables Amazon RDS to create a new standby replica in another Availability Zone if necessary. You can follow this best practice even for Single-AZ deployments, because you might convert them to Multi-AZ deployments in the future.

For a DB instance to be publicly accessible, the subnets in the DB subnet group must have an internet gateway. For more information about internet gateways for subnets, see Connect to the internet using an internet gateway in the *Amazon VPC User Guide*.

> **Note**
>
> The DB subnet group for a Local Zone can have only one subnet.

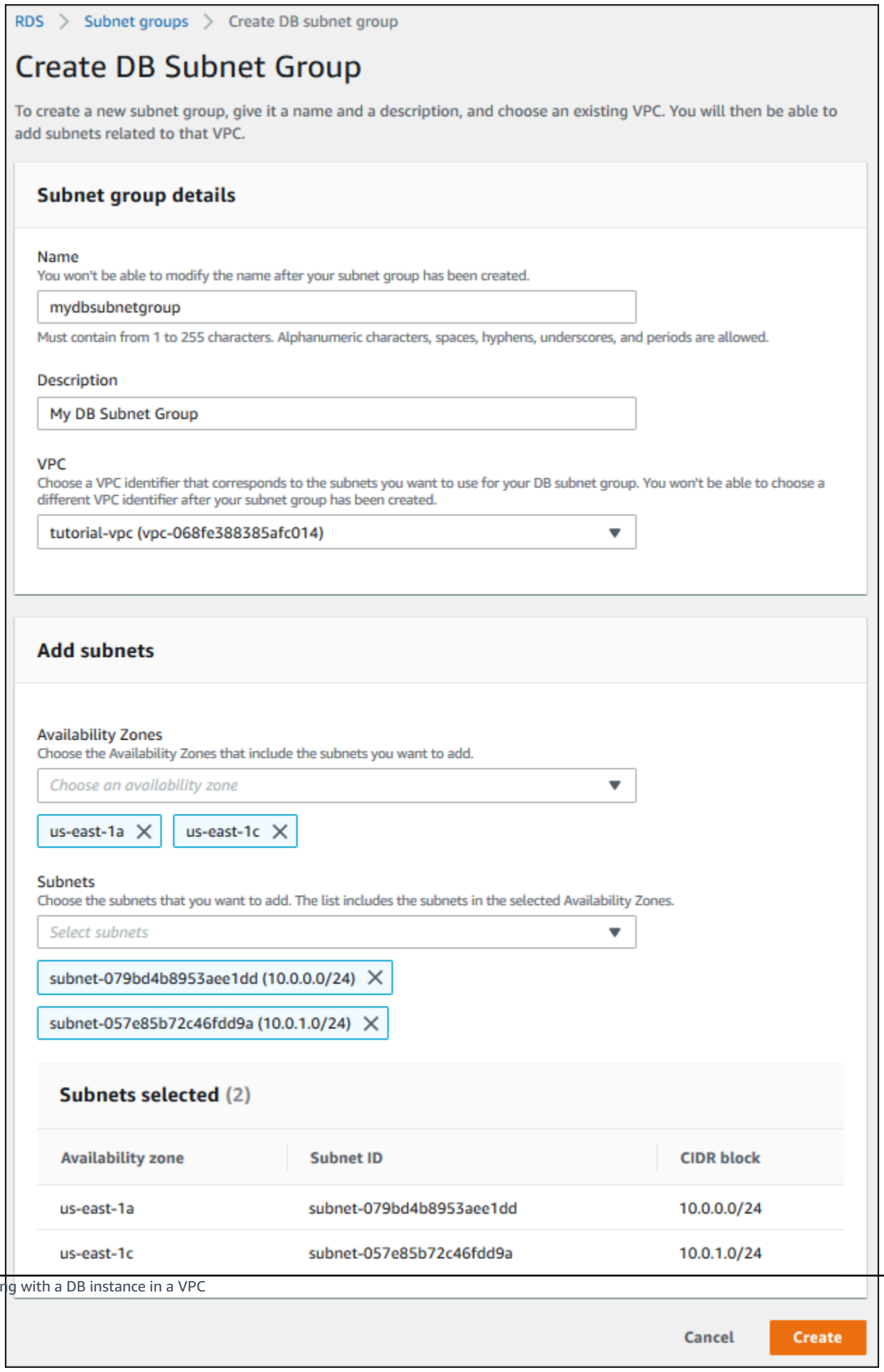
When you create a DB instance in a VPC, you can choose a DB subnet group. Amazon RDS chooses a subnet and an IP address within that subnet to associate with your DB instance. If no DB subnet groups exist, Amazon RDS creates a default subnet group when you create a DB instance. Amazon RDS creates and associates an Elastic Network Interface to your DB instance with that IP address. The DB instance uses the Availability Zone that contains the subnet.

For Multi-AZ deployments, defining a subnet for two or more Availability Zones in an AWS Region allows Amazon RDS to create a new standby in another Availability Zone should the need arise. You need to do this even For Single-AZ deployments, just in case you want to convert them to Multi-AZ deployments at some point.

In this step, you create a DB subnet group and add the subnets that you created for your VPC.

To create a DB subnet group

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Subnet groups**.
3. Choose **Create DB Subnet Group**.
4. For **Name**, type the name of your DB subnet group.
5. For **Description**, type a description for your DB subnet group.
6. For **VPC**, choose the default VPC or the VPC that you created.
7. In the **Add subnets** section, choose the Availability Zones that include the subnets from **Availability Zones**, and then choose the subnets from **Subnets**.


RDS > Subnet groups > Create DB subnet group
Create DB Subnet Group
To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.
Subnet group details
Name
You won't be able to modify the name after your subnet group has been created.
mydbsubnetgroup
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.
Description
My DB Subnet Group
VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.
tutorial-vpc (vpc-068fe388385afc014)
Add subnets
Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
us-east-1a
us-east-1c
Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
subnet-079bd4b8953aee1dd (10.0.0.0/24)
subnet-057e85b72c46fdd9a (10.0.1.0/24)
Subnets selected (2)
Availability zone
Subnet ID
CIDR block
us-east-1a
subnet-079bd4b8953aee1dd
10.0.0.0/24
us-east-1c
subnet-057e85b72c46fdd9a
10.0.1.0/24
Cancel
Create

> **Note**
>
> If you have enabled a Local Zone, you can choose an Availability Zone group on the **Create DB subnet group** page. In this case, choose the **Availability Zone group**, **Availability Zones**, and **Subnets**.

8. Choose **Create**.

   Your new DB subnet group appears in the DB subnet groups list on the RDS console. You can choose the DB subnet group to see details, including all of the subnets associated with the group, in the details pane at the bottom of the window.

Step 3: Create a VPC security group

Before you create your DB instance, you can create a VPC security group to associate with your DB instance. If you don't create a VPC security group, you can use the default security group when you create a DB instance. For instructions on how to create a security group for your DB instance, see Create a VPC security group for a private DB instance, or see Control traffic to resources using security groups in the *Amazon VPC User Guide*.

Step 4: Create a DB instance in the VPC

In this step, you create a DB instance and use the VPC name, the DB subnet group, and the VPC security group you created in the previous steps.

> **Note**
>
> If you want your DB instance in the VPC to be publicly accessible, you must enable the VPC attributes *DNS hostnames* and *DNS resolution*. For more information, see DNS attributes for your VPC in the *Amazon VPC User Guide*.

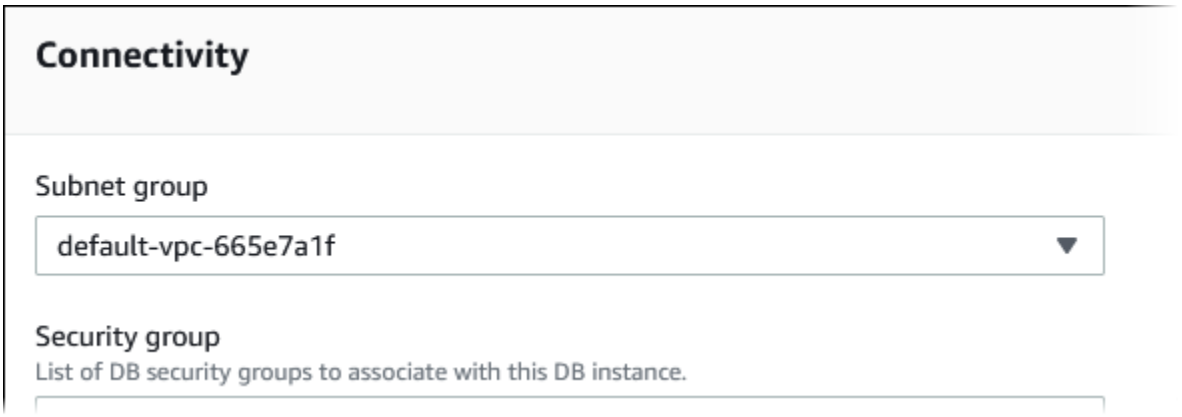
For details on how to create a DB instance, see Creating an Amazon RDS DB instance.

When prompted in the **Connectivity** section, enter the VPC name, the DB subnet group, and the VPC security group.

Updating the VPC for a DB instance

You can use the AWS Management Console to move your DB instance to a different VPC.

For information about modifying a DB instance, see Modifying an Amazon RDS DB instance. In the **Connectivity** section of the modify page, shown following, enter the new DB subnet group for **DB subnet group**. The new subnet group must be a subnet group in a new VPC.



You can't change the VPC for a DB instance if the following conditions apply:

- The DB instance is in multiple Availability Zones. You can convert the DB instance to a single Availability Zone, move it to a new VPC, and then convert it back to a Multi-AZ DB instance. For more information, see Configuring and managing a Multi-AZ deployment for Amazon RDS.
- The DB instance has one or more read replicas. You can remove the read replicas, move the DB instance to a new VPC, and then add the read replicas again. For more information, see Working with DB instance read replicas.
- The DB instance is a read replica. You can promote the read replica, and then move the standalone DB instance to a new VPC. For more information, see Promoting a read replica to be a standalone DB instance.
- The subnet group in the target VPC doesn't have subnets in the DB instance's the Availability Zone. You can add subnets in the DB instance's Availability Zone to the DB subnet group, and then move the DB instance to the new VPC. For more information, see Working with DB subnet groups.

Scenarios for accessing a DB instance in a VPC

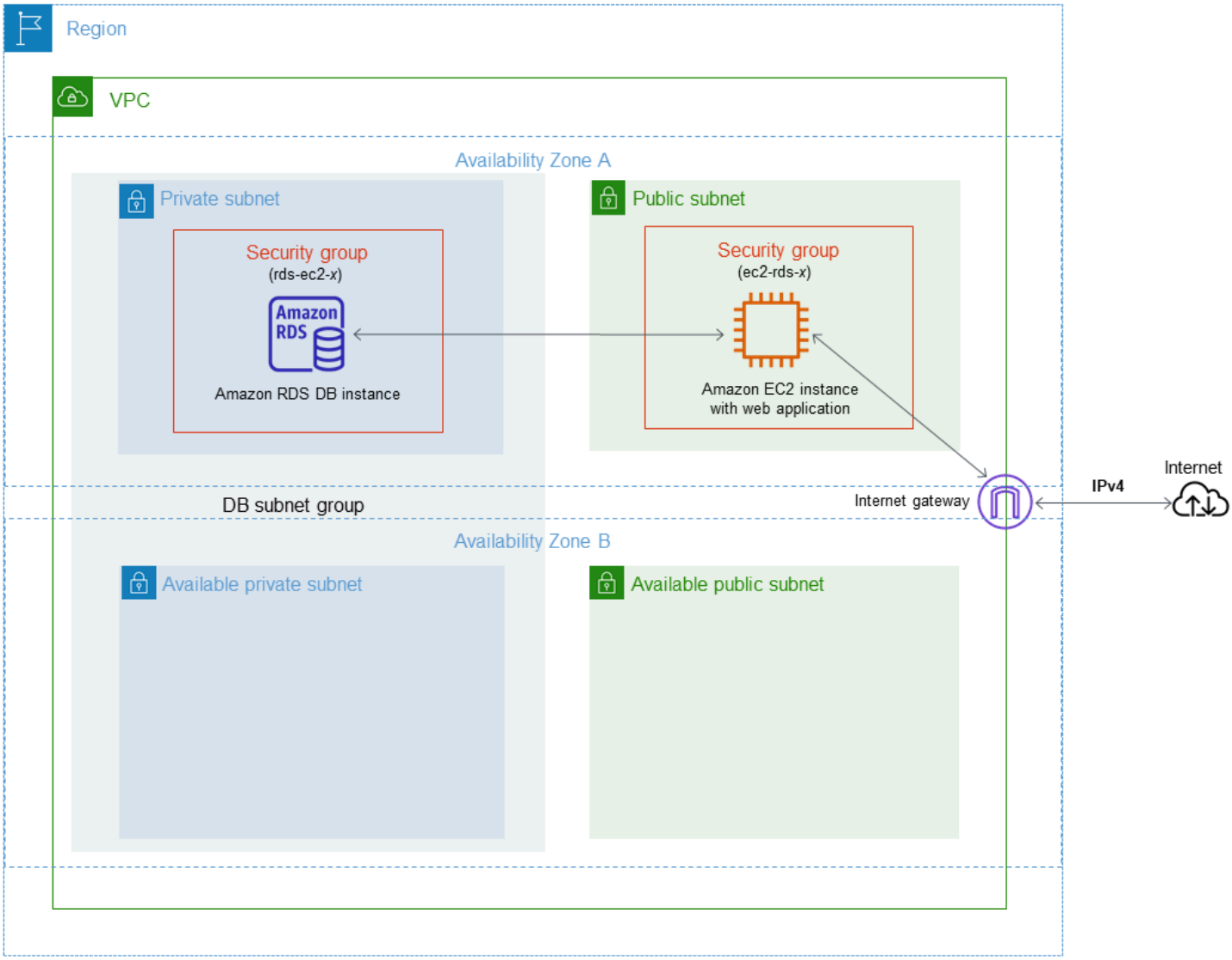
Amazon RDS supports the following scenarios for accessing a DB instance in a VPC:

- An Amazon EC2 instance in the same VPC
- An EC2 instance in a different VPC
- A client application through the internet
- A private network

A DB instance in a VPC accessed by an Amazon EC2 instance in the same VPC

A common use of a DB instance in a VPC is to share data with an application server that is running in an Amazon EC2 instance in the same VPC.

The following diagram shows this scenario.



The simplest way to manage access between EC2 instances and DB instances in the same VPC is to do the following:

- Create a VPC security group for your DB instances to be in. This security group can be used to restrict access to the DB instances. For example, you can create a custom rule for this security group. This might allow TCP access using the port that you assigned to the DB instance when you created it and an IP address you use to access the DB instance for development or other purposes.
- Create a VPC security group for your EC2 instances (web servers and clients) to be in. This security group can, if needed, allow access to the EC2 instance from the internet by using the VPC's routing table. For example, you can set rules on this security group to allow TCP access to the EC2 instance over port 22.
- Create custom rules in the security group for your DB instances that allow connections from the security group you created for your EC2 instances. These rules might allow any member of the security group to access the DB instances.

There is an additional public and private subnet in a separate Availability Zone. An RDS DB subnet group requires a subnet in at least two Availability Zones. The additional subnet makes it easy to switch to a Multi-AZ DB instance deployment in the future.

For a tutorial that shows you how to create a VPC with both public and private subnets for this scenario, see Tutorial: Create a VPC for use with a DB instance (IPv4 only).

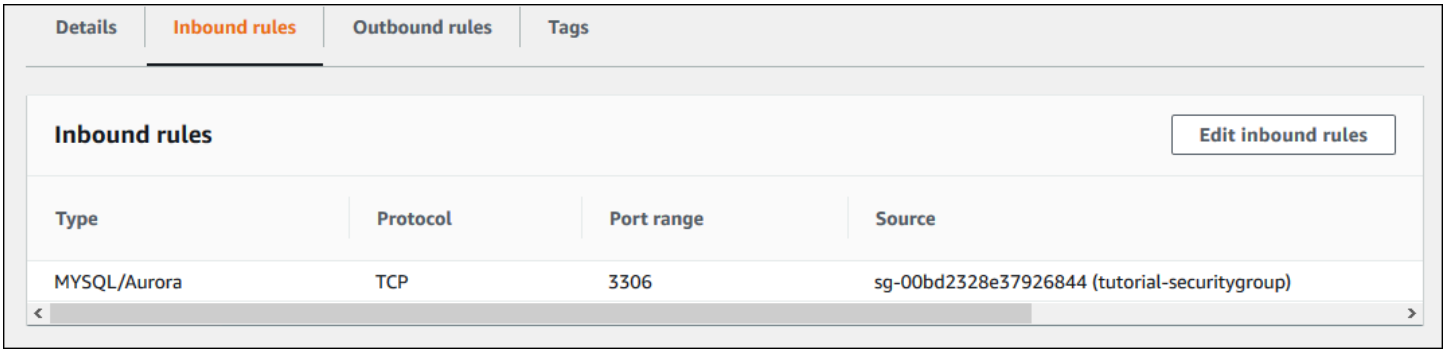
> **Tip**
>
> You can set up network connectivity between an Amazon EC2 instance and a DB instance automatically when you create the DB instance. For more information, see Configure automatic network connectivity with an EC2 instance.

To create a rule in a VPC security group that allows connections from another security group, do the following:

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc.
2. In the navigation pane, choose **Security groups**.

3. Choose or create a security group for which you want to allow access to members of another security group. In the preceding scenario, this is the security group that you use for your DB instances. Choose the **Inbound rules** tab, and then choose **Edit inbound rules**.
4. On the **Edit inbound rules** page, choose **Add rule**.
5. For **Type**, choose the entry that corresponds to the port you used when you created your DB instance, such as **MYSQL/Aurora**.
6. In the **Source** box, start typing the ID of the security group, which lists the matching security groups. Choose the security group with members that you want to have access to the resources protected by this security group. In the scenario preceding, this is the security group that you use for your EC2 instance.
7. If required, repeat the steps for the TCP protocol by creating a rule with **All TCP** as the **Type** and your security group in the **Source** box. If you intend to use the UDP protocol, create a rule with **All UDP** as the **Type** and your security group in **Source**.
8. Choose **Save rules**.

The following screen shows an inbound rule with a security group for its source.

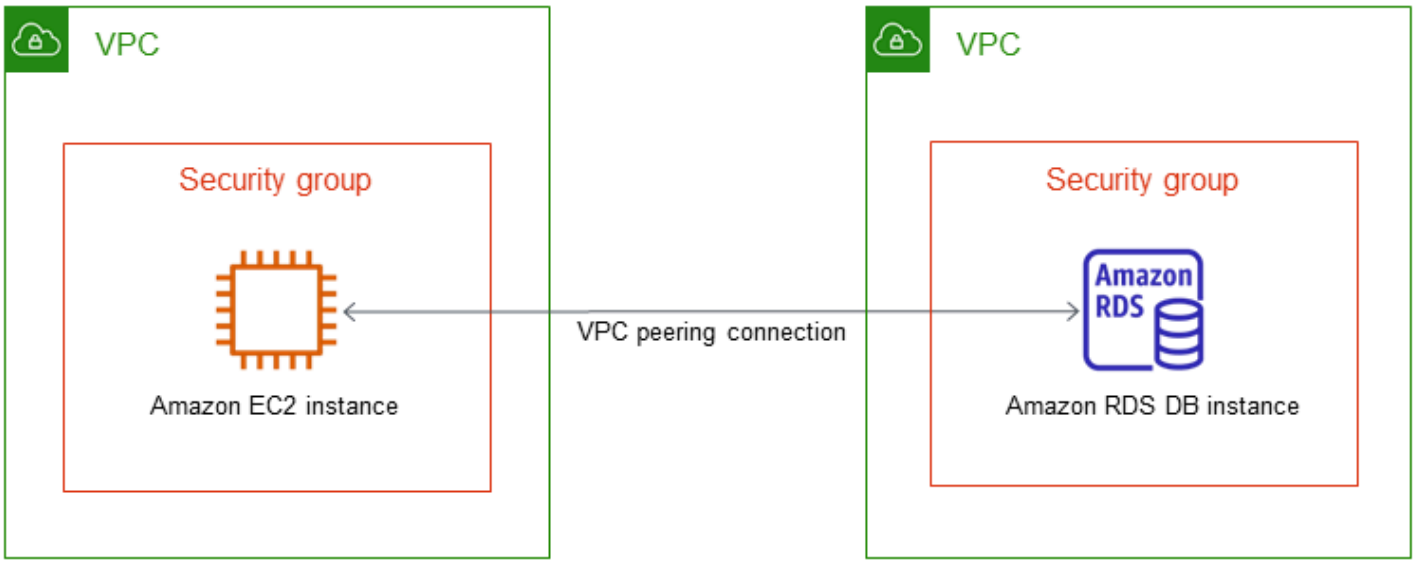


For more information about connecting to the DB instance from your EC2 instance, see Connecting to an Amazon RDS DB instance .

A DB instance in a VPC accessed by an EC2 instance in a different VPC

When your DB instances is in a different VPC from the EC2 instance you are using to access it, you can use VPC peering to access the DB instance.

The following diagram shows this scenario.

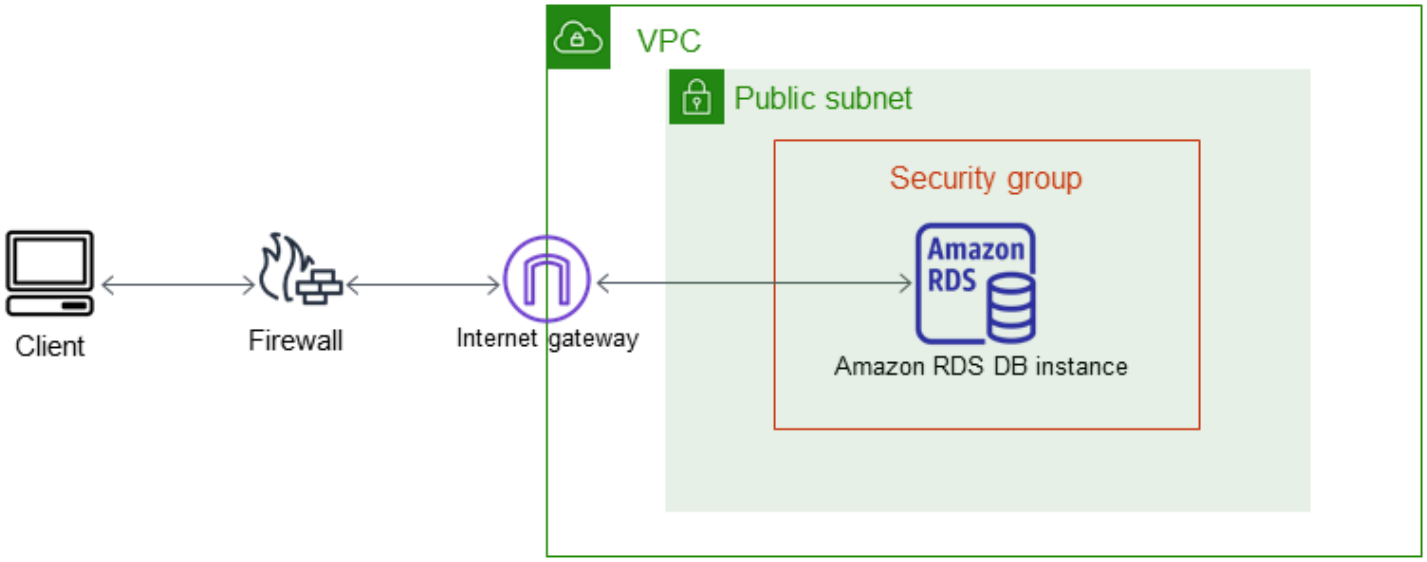


A VPC peering connection is a networking connection between two VPCs that enables you to route traffic between them using private IP addresses. Resources in either VPC can communicate with each other as if they are within the same network. You can create a VPC peering connection between your own VPCs, with a VPC in another AWS account, or with a VPC in a different AWS Region. To learn more about VPC peering, see VPC peering in the *Amazon Virtual Private Cloud User Guide*.

A DB instance in a VPC accessed by a client application through the internet

To access a DB instances in a VPC from a client application through the internet, you configure a VPC with a single public subnet, and an internet gateway to enable communication over the internet.

The following diagram shows this scenario.



We recommend the following configuration:

- A VPC of size /16 (for example CIDR: 10.0.0.0/16). This size provides 65,536 private IP addresses.
- A subnet of size /24 (for example CIDR: 10.0.0.0/24). This size provides 256 private IP addresses.
- An Amazon RDS DB instance that is associated with the VPC and the subnet. Amazon RDS assigns an IP address within the subnet to your DB instance.
- An internet gateway which connects the VPC to the internet and to other AWS products.
- A security group associated with the DB instance. The security group's inbound rules allow your client application to access to your DB instance.

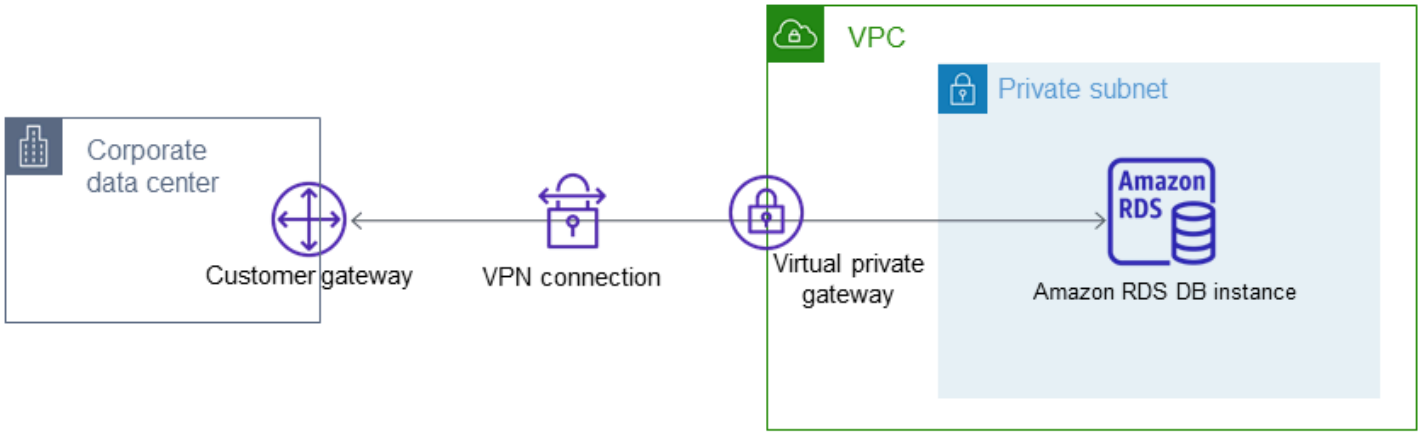
For information about creating a DB instances in a VPC, see Creating a DB instance in a VPC.

A DB instance in a VPC accessed by a private network

If your DB instance isn't publicly accessible, you have the following options for accessing it from a private network:

- An AWS Site-to-Site VPN connection. For more information, see What is AWS Site-to-Site VPN?
- An Direct Connect connection. For more information, see What is Direct Connect?
- An AWS Client VPN connection. For more information, see What is AWS Client VPN?

The following diagram shows a scenario with an AWS Site-to-Site VPN connection.

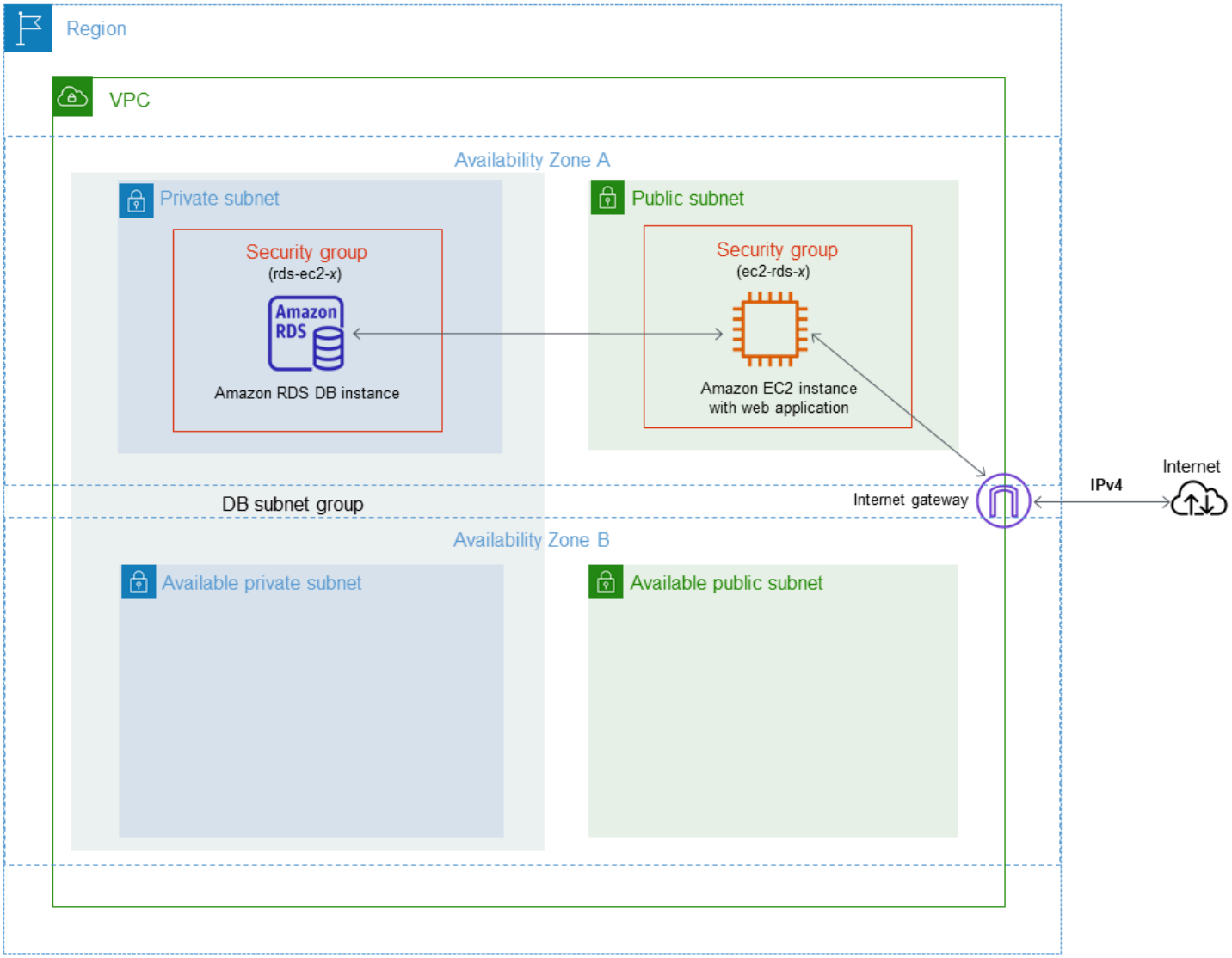


For more information, see Internetwork traffic privacy.

Tutorial: Create a VPC for use with a DB instance (IPv4 only)

A common scenario includes a DB instance in a virtual private cloud (VPC) based on the Amazon VPC service. This VPC shares data with a web server that is running in the same VPC. In this tutorial, you create the VPC for this scenario.

The following diagram shows this scenario. For information about other scenarios, see Scenarios for accessing a DB instance in a VPC.



Your DB instance needs to be available only to your web server, and not to the public internet. Thus, you create a VPC with both public and private subnets. The web server is hosted in the public subnet, so that it can reach the public internet. The DB instance is hosted in a private subnet. The web server can connect to the DB instance because it is hosted within the same VPC. But the DB instance isn't available to the public internet, providing greater security.

This tutorial configures an additional public and private subnet in a separate Availability Zone. These subnets aren't used by the tutorial. An RDS DB subnet group requires a subnet in at least two Availability Zones. The additional subnet makes it easier to switch to a Multi-AZ DB instance deployment in the future.

This tutorial describes configuring a VPC for Amazon RDS DB instances. For a tutorial that shows you how to create a web server for this VPC scenario, see Tutorial: Create a web server and an Amazon RDS DB instance. For more information about Amazon VPC, see Amazon VPC Getting Started Guide and Amazon VPC User Guide.

> **Tip**
>
> You can set up network connectivity between an Amazon EC2 instance and a DB instance automatically when you create the DB instance. The network configuration is similar to the one described in this tutorial. For more information, see Configure automatic network connectivity with an EC2 instance.

Create a VPC with private and public subnets

Use the following procedure to create a VPC with both public and private subnets.

To create a VPC and subnets

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the top-right corner of the AWS Management Console, choose the Region to create your VPC in. This example uses the US West (Oregon) Region.
3. In the upper-left corner, choose **VPC dashboard**. To begin creating a VPC, choose **Create VPC**.
4. For **Resources to create** under **VPC settings**, choose **VPC and more**.
5. For the **VPC settings**, set these values:

   - **Name tag auto-generation** – `tutorial`
   - **IPv4 CIDR block** – `10.0.0.0/16`
   - **IPv6 CIDR block** – **No IPv6 CIDR block**
   - **Tenancy** – **Default**
   - **Number of Availability Zones (AZs)** – **2**
   - **Customize AZs** – Keep the default values.

- **Number of public subnet** – **2**
- **Number of private subnets** – **2**
- **Customize subnets CIDR blocks** – Keep the default values.
- **NAT gateways ($)** – **None**
- **VPC endpoints** – **None**
- **DNS options** – Keep the default values.

> **Note**
>
> Amazon RDS requires at least two subnets in two different Availability Zones to support Multi-AZ DB instance deployments. This tutorial creates a Single-AZ deployment, but the requirement makes it easier to convert to a Multi-AZ DB instance deployment in the future.

6. Choose **Create VPC**.

Create a VPC security group for a public web server

Next, you create a security group for public access. To connect to public EC2 instances in your VPC, you add inbound rules to your VPC security group. These allow traffic to connect from the internet.

To create a VPC security group

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
   - **Security group name: `tutorial-securitygroup`**
   - **Description: `Tutorial Security Group`**
   - **VPC:** Choose the VPC that you created earlier, for example: **vpc-*identifier* (tutorial-vpc)**
4. Add inbound rules to the security group.
   a. Determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH). To determine your public IP address, in a different browser window or tab,

you can use the service at https://checkip.amazonaws.com. An example of an IP address is `203.0.113.25/32`.

In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, find the range of IP addresses used by client computers.

> **Warning**
>
> If you use `0.0.0.0/0` for SSH access, you make it possible for all IP addresses to access your public instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your instances using SSH.

b. In the **Inbound rules** section, choose **Add rule**.

c. Set the following values for your new inbound rule to allow SSH access to your Amazon EC2 instance. If you do this, you can connect to your Amazon EC2 instance to install the web server and other utilities. You also connect to your EC2 instance to upload content for your web server.

- **Type: SSH**
- **Source:** The IP address or range from Step a, for example: **`203.0.113.25/32`**.

d. Choose **Add rule**.

e. Set the following values for your new inbound rule to allow HTTP access to your web server:

- **Type: HTTP**
- **Source: `0.0.0.0/0`**

5. Choose **Create security group** to create the security group.

Note the security group ID because you need it later in this tutorial.

Create a VPC security group for a private DB instance

To keep your DB instance private, create a second security group for private access. To connect to private DB instances in your VPC, you add inbound rules to your VPC security group that allow traffic from your web server only.

To create a VPC security group

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
   - **Security group name: `tutorial-db-securitygroup`**
   - **Description: `Tutorial DB Instance Security Group`**
   - **VPC:** Choose the VPC that you created earlier, for example: **vpc-*identifier* (tutorial-vpc)**
4. Add inbound rules to the security group.
   a. In the **Inbound rules** section, choose **Add rule**.
   b. Set the following values for your new inbound rule to allow MySQL traffic on port 3306 from your Amazon EC2 instance. If you do this, you can connect from your web server to your DB instance. By doing so, you can store and retrieve data from your web application to your database.
      - **Type: `MySQL/Aurora`**
      - **Source:** The identifier of the **tutorial-securitygroup** security group that you created previously in this tutorial, for example: **sg-9edd5cfb**.
5. Choose **Create security group** to create the security group.

Create a DB subnet group

A *DB subnet group* is a collection of subnets that you create in a VPC and that you then designate for your DB instances. A DB subnet group makes it possible for you to specify a particular VPC when creating DB instances.

To create a DB subnet group

1. Identify the private subnets for your database in the VPC.

a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

b. Choose **VPC Dashboard**, and then choose **Subnets**.

c. Note the subnet IDs of the subnets named **tutorial-subnet-private1-us-west-2a** and **tutorial-subnet-private2-us-west-2b**.

   You need the subnet IDs when you create your DB subnet group.

2. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.

   Make sure that you connect to the Amazon RDS console, not to the Amazon VPC console.

3. In the navigation pane, choose **Subnet groups**.

4. Choose **Create DB subnet group**.

5. On the **Create DB subnet group** page, set these values in **Subnet group details**:

   - **Name: `tutorial-db-subnet-group`**
   - **Description: `Tutorial DB Subnet Group`**
   - **VPC: tutorial-vpc (vpc-*`identifier`*)**

6. In the **Add subnets** section, choose the **Availability Zones** and **Subnets**.

   For this tutorial, choose **us-west-2a** and **us-west-2b** for the **Availability Zones**. For **Subnets**, choose the private subnets you identified in the previous step.

7. Choose **Create**.

   Your new DB subnet group appears in the DB subnet groups list on the RDS console. You can choose the DB subnet group to see details in the details pane at the bottom of the window. These details include all of the subnets associated with the group.

> **Note**
>
> If you created this VPC to complete Tutorial: Create a web server and an Amazon RDS DB instance, create the DB instance by following the instructions in Create an Amazon RDS DB instance.

Deleting the VPC

After you create the VPC and other resources for this tutorial, you can delete them if they are no longer needed.

> **Note**
>
> If you added resources in the VPC that you created for this tutorial, you might need to delete these before you can delete the VPC. For example, these resources might include Amazon EC2 instances or Amazon RDS DB instances. For more information, see Delete your VPC in the *Amazon VPC User Guide*.

To delete a VPC and related resources

1. Delete the DB subnet group.
   a. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
   b. In the navigation pane, choose **Subnet groups**.
   c. Select the DB subnet group you want to delete, such as **tutorial-db-subnet-group**.
   d. Choose **Delete**, and then choose **Delete** in the confirmation window.
2. Note the VPC ID.
   a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
   b. Choose **VPC Dashboard**, and then choose **VPCs**.
   c. In the list, identify the VPC that you created, such as **tutorial-vpc**.
   d. Note the **VPC ID** of the VPC that you created. You need the VPC ID in later steps.
3. Delete the security groups.
   a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
   b. Choose **VPC Dashboard**, and then choose **Security Groups**.
   c. Select the security group for the Amazon RDS DB instance, such as **tutorial-db-securitygroup**.
   d. For **Actions**, choose **Delete security groups**, and then choose **Delete** on the confirmation page.

e. On the **Security Groups** page, select the security group for the Amazon EC2 instance, such as **tutorial-securitygroup**.

f. For **Actions**, choose **Delete security groups**, and then choose **Delete** on the confirmation page.

4. Delete the VPC.

a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

b. Choose **VPC Dashboard**, and then choose **VPCs**.

c. Select the VPC you want to delete, such as **tutorial-vpc**.

d. For **Actions**, choose **Delete VPC**.

The confirmation page shows other resources that are associated with the VPC that will also be deleted, including the subnets associated with it.

e. On the confirmation page, enter **`delete`**, and then choose **Delete**.

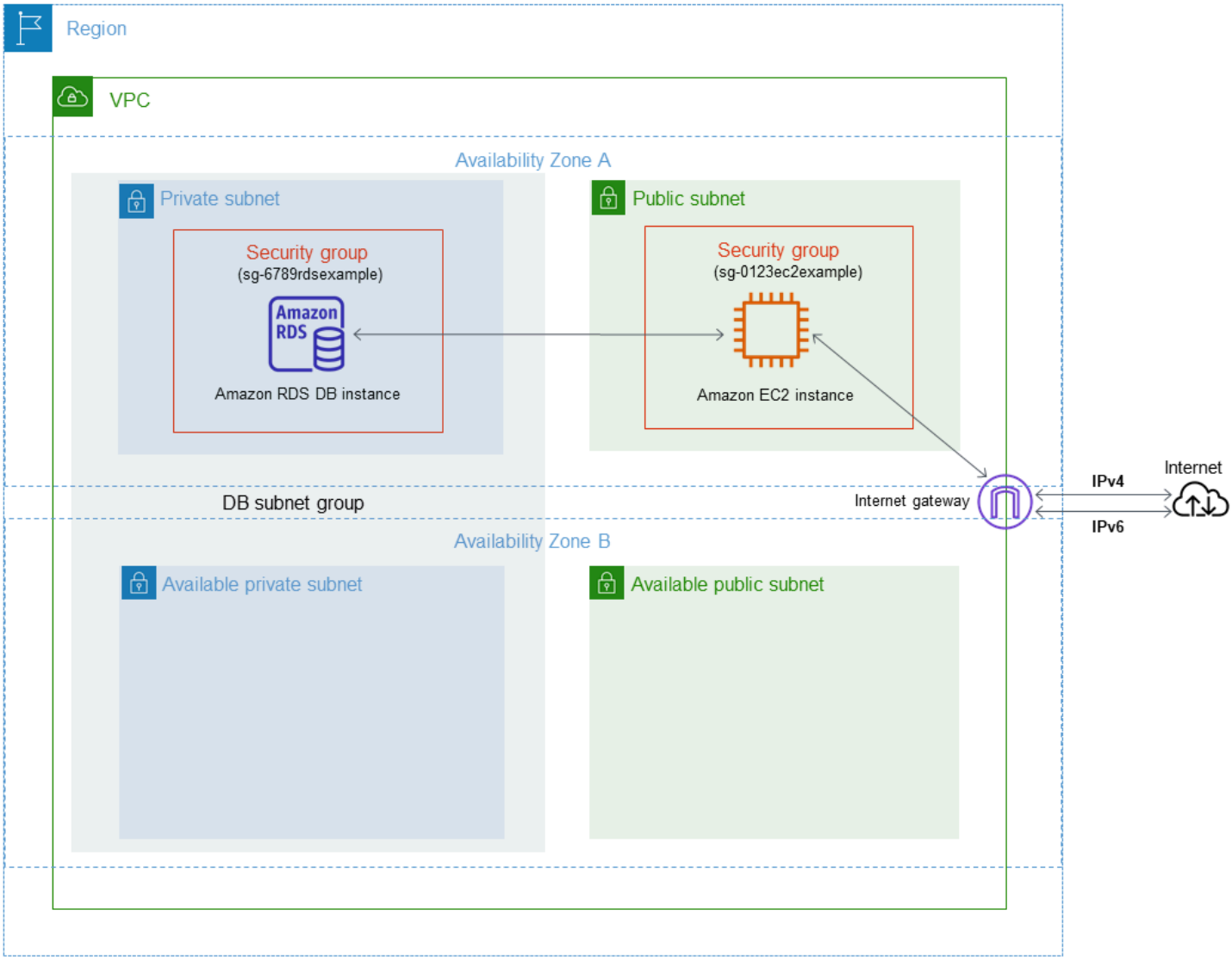
Tutorial: Create a VPC for use with a DB instance (dual-stack mode)

A common scenario includes a DB instance in a virtual private cloud (VPC) based on the Amazon VPC service. This VPC shares data with a public Amazon EC2 instance that is running in the same VPC.

In this tutorial, you create the VPC for this scenario that works with a database running in dual-stack mode. Dual-stack mode to enable connection over the IPv6 addressing protocol. For more information about IP addressing, see Amazon RDS IP addressing.

Dual-stack network instances are supported in most regions. For more information see Region and version availability. To see the limitations of dual-stack mode, see Limitations for dual-stack network DB instances.

The following diagram shows this scenario.



For information about other scenarios, see Scenarios for accessing a DB instance in a VPC.

Your DB instance needs to be available only to your Amazon EC2 instance, and not to the public internet. Thus, you create a VPC with both public and private subnets. The Amazon EC2 instance is hosted in the public subnet, so that it can reach the public internet. The DB instance is hosted in a private subnet. The Amazon EC2 instance can connect to the DB instance because it's hosted within the same VPC. However, the DB instance is not available to the public internet, providing greater security.

This tutorial configures an additional public and private subnet in a separate Availability Zone. These subnets aren't used by the tutorial. An RDS DB subnet group requires a subnet in at least two Availability Zones. The additional subnet makes it easy to switch to a Multi-AZ DB instance deployment in the future.

To create a DB instance that uses dual-stack mode, specify **Dual-stack mode** for the **Network type** setting. You can also modify a DB instance with the same setting. For more information, see Creating an Amazon RDS DB instance and Modifying an Amazon RDS DB instance.

This tutorial describes configuring a VPC for Amazon RDS DB instances. For more information about Amazon VPC, see Amazon VPC User Guide.

Create a VPC with private and public subnets

Use the following procedure to create a VPC with both public and private subnets.

To create a VPC and subnets

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the upper-right corner of the AWS Management Console, choose the Region to create your VPC in. This example uses the US East (Ohio) Region.
3. In the upper-left corner, choose **VPC dashboard**. To begin creating a VPC, choose **Create VPC**.
4. For **Resources to create** under **VPC settings**, choose **VPC and more**.
5. For the remaining **VPC settings**, set these values:
   - **Name tag auto-generation** – **`tutorial-dual-stack`**
   - **IPv4 CIDR block** – **`10.0.0.0/16`**
   - **IPv6 CIDR block** – **Amazon-provided IPv6 CIDR block**
   - **Tenancy** – **Default**
   - **Number of Availability Zones (AZs)** – **2**
   - **Customize AZs** – Keep the default values.
   - **Number of public subnet** – **2**
   - **Number of private subnets** – **2**
   - **Customize subnets CIDR blocks** – Keep the default values.
   - **NAT gateways ($)** – **None**
   - **Egress only internet gateway** – **No**
   - **VPC endpoints** – **None**
   - **DNS options** – Keep the default values.

> **Note**
>
> Amazon RDS requires at least two subnets in two different Availability Zones to support Multi-AZ DB instance deployments. This tutorial creates a Single-AZ deployment, but the requirement makes it easy to convert to a Multi-AZ DB instance deployment in the future.

6. Choose **Create VPC**.

Create a VPC security group for a public Amazon EC2 instance

Next, you create a security group for public access. To connect to public EC2 instances in your VPC, add inbound rules to your VPC security group that allow traffic to connect from the internet.

To create a VPC security group

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
   - **Security group name: `tutorial-dual-stack-securitygroup`**
   - **Description: `Tutorial Dual-Stack Security Group`**
   - **VPC:** Choose the VPC that you created earlier, for example: **vpc-*identifier* (tutorial-dual-stack-vpc)**
4. Add inbound rules to the security group.
   a. Determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH).

      An example of an Internet Protocol version 4 (IPv4) address is `203.0.113.25/32`.
      An example of an Internet Protocol version 6 (IPv6) address range is `2001:db8:1234:1a00::/64`.

      In many cases, you might connect through an internet service provider (ISP) or from behind your firewall without a static IP address. If so, find the range of IP addresses used by client computers.

> **Warning**
>
> If you use `0.0.0.0/0` for IPv4 or `::0` for IPv6, you make it possible for all IP addresses to access your public instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your instances.

b. In the **Inbound rules** section, choose **Add rule**.

c. Set the following values for your new inbound rule to allow Secure Shell (SSH) access to your Amazon EC2 instance. If you do this, you can connect to your EC2 instance to install SQL clients and other applications. Specify an IP address so you can access your EC2 instance:

- **Type: SSH**
- **Source:** The IP address or range from step a. An example of an IPv4 IP address is **`203.0.113.25/32`**. An example of an IPv6 IP address is **`2001:DB8::/32`**.

5. Choose **Create security group** to create the security group.

   Note the security group ID because you need it later in this tutorial.

Create a VPC security group for a private DB instance

To keep your DB instance private, create a second security group for private access. To connect to private DB instances in your VPC, add inbound rules to your VPC security group. These allow traffic from your Amazon EC2 instance only.

To create a VPC security group

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:

- **Security group name: `tutorial-dual-stack-db-securitygroup`**
- **Description: `Tutorial Dual-Stack DB Instance Security Group`**

- **VPC:** Choose the VPC that you created earlier, for example: **vpc-*identifier* (tutorial-dual-stack-vpc)**

4. Add inbound rules to the security group:

   a. In the **Inbound rules** section, choose **Add rule**.

   b. Set the following values for your new inbound rule to allow MySQL traffic on port 3306 from your Amazon EC2 instance. If you do, you can connect from your EC2 instance to your DB instance. Doing this means that you can send data from your EC2 instance to your database.

      - **Type: MySQL/Aurora**
      - **Source:** The identifier of the **tutorial-dual-stack-securitygroup** security group that you created previously in this tutorial, for example **sg-9edd5cfb**.

5. To create the security group, choose **Create security group**.

Create a DB subnet group

A *DB subnet group* is a collection of subnets that you create in a VPC and that you then designate for your DB instances. By using a DB subnet group, you can specify a particular VPC when creating DB instances. To create a DB subnet group that is DUAL compatible, all subnets must be DUAL compatible. To be DUAL compatible, a subnet must have an IPv6 CIDR associated with it.

To create a DB subnet group

1. Identify the private subnets for your database in the VPC.

   a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

   b. Choose **VPC Dashboard**, and then choose **Subnets**.

   c. Note the subnet IDs of the subnets named **tutorial-dual-stack-subnet-private1-us-west-2a** and **tutorial-dual-stack-subnet-private2-us-west-2b**.

      You will need the subnet IDs when you create your DB subnet group.

2. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.

   Make sure that you connect to the Amazon RDS console, not to the Amazon VPC console.

3. In the navigation pane, choose **Subnet groups**.

4. Choose **Create DB subnet group**.

5. On the **Create DB subnet group** page, set these values in **Subnet group details**:
   - **Name: `tutorial-dual-stack-db-subnet-group`**
   - **Description: `Tutorial Dual-Stack DB Subnet Group`**
   - **VPC: tutorial-dual-stack-vpc (vpc-*`identifier`*)**
6. In the **Add subnets** section, choose values for the **Availability Zones** and **Subnets** options.

   For this tutorial, choose **us-east-2a** and **us-east-2b** for the **Availability Zones**. For **Subnets**, choose the private subnets you identified in the previous step.
7. Choose **Create**.

Your new DB subnet group appears in the DB subnet groups list on the RDS console. You can choose the DB subnet group to see its details. These include the supported addressing protocols and all of the subnets associated with the group and the network type supported by the DB subnet group.

Create an Amazon EC2 instance in dual-stack mode

To create an Amazon EC2 instance, follow the instructions in Launch an instance using the new launch instance wizard in the *Amazon EC2 User Guide*.

On the **Configure Instance Details** page, set these values and keep the other values as their defaults:

- **Network** – Choose an existing VPC with both public and private subnets, such as **tutorial-dual-stack-vpc** (vpc-*`identifier`*) created in Create a VPC with private and public subnets.
- **Subnet** – Choose an existing public subnet, such as **subnet-*`identifier`* | tutorial-dual-stack-subnet-public1-us-east-2a | us-east-2a** created in Create a VPC security group for a public Amazon EC2 instance.
- **Auto-assign Public IP** – Choose **Enable**.
- **Auto-assign IPv6 IP** – Choose **Enable**.
- **Firewall (security groups)** – Choose **Select an existing security group**.
- **Common security groups** – Choose an existing security group, such as the `tutorial-securitygroup` created in Create a VPC security group for a public Amazon EC2 instance. Make sure that the security group that you choose includes inbound rules for Secure Shell (SSH) and HTTP access.

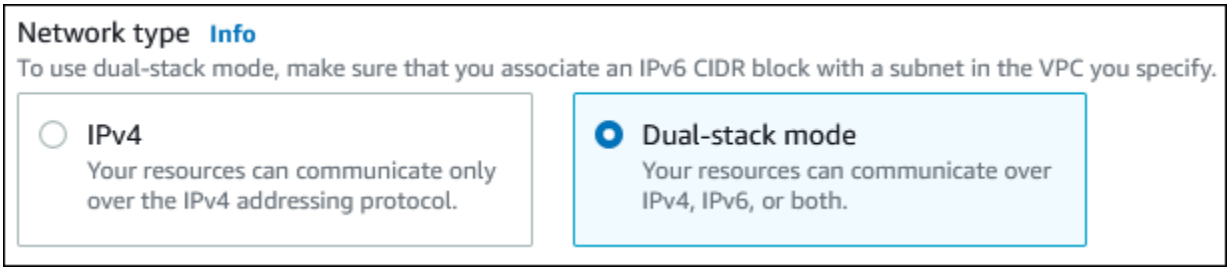
Create a DB instance in dual-stack mode

In this step, you create a DB instance that runs in dual-stack mode.

To create a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the upper-right corner of the console, choose the AWS Region where you want to create the DB instance. This example uses the US East (Ohio) Region.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.
5. On the **Create database** page, make sure that the **Standard create** option is chosen, and then choose the MySQL DB engine type.
6. In the **Connectivity** section, set these values:

   - **Network type** – Choose **Dual-stack mode**.

   - **Virtual private cloud (VPC)** – Choose an existing VPC with both public and private subnets, such as **tutorial-dual-stack-vpc** (vpc-*identifier*) created in Create a VPC with private and public subnets.

     The VPC must have subnets in different Availability Zones.

   - **DB subnet group** – Choose a DB subnet group for the VPC, such as **tutorial-dual-stack-db-subnet-group** created in Create a DB subnet group.
   - **Public access** – Choose **No**.
   - **VPC security group (firewall)** – Select **Choose existing**.
   - **Existing VPC security groups** – Choose an existing VPC security group that is configured for private access, such as **tutorial-dual-stack-db-securitygroup** created in Create a VPC security group for a private DB instance.

Remove other security groups, such as the default security group, by choosing the **X** associated with each.

- **Availability Zone** – Choose **us-west-2a**.

  To avoid cross-AZ traffic, make sure the DB instance and the EC2 instance are in the same Availability Zone.

7. For the remaining sections, specify your DB instance settings. For information about each setting, see Settings for DB instances.

Connect to your Amazon EC2 instance and DB instance

After you create your Amazon EC2 instance and DB instance in dual-stack mode, you can connect to each one using the IPv6 protocol. To connect to an Amazon EC2 instance using the IPv6 protocol, follow the instructions in Connect to your Linux instance in the *Amazon EC2 User Guide*.

To connect to your RDS for MySQL DB instance from the Amazon EC2 instance, follow the instructions in Connect to a MySQL DB instance.

Deleting the VPC

After you create the VPC and other resources for this tutorial, you can delete them if they are no longer needed.

If you added resources in the VPC that you created for this tutorial, you might need to delete these before you can delete the VPC. Examples of resources are Amazon EC2 instances or DB instances. For more information, see Delete your VPC in the *Amazon VPC User Guide*.

To delete a VPC and related resources

1. Delete the DB subnet group:

   a. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.
   b. In the navigation pane, choose **Subnet groups**.
   c. Select the DB subnet group to delete, such as **tutorial-db-subnet-group**.
   d. Choose **Delete**, and then choose **Delete** in the confirmation window.
2. Note the VPC ID:

   a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

b. Choose **VPC Dashboard**, and then choose **VPCs**.

c. In the list, identify the VPC you created, such as **tutorial-dual-stack-vpc**.

d. Note the **VPC ID** value of the VPC that you created. You need this VPC ID in subsequent steps.

3. Delete the security groups:

a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

b. Choose **VPC Dashboard**, and then choose **Security Groups**.

c. Select the security group for the Amazon RDS DB instance, such as **tutorial-dual-stack-db-securitygroup**.

d. For **Actions**, choose **Delete security groups**, and then choose **Delete** on the confirmation page.

e. On the **Security Groups** page, select the security group for the Amazon EC2 instance, such as **tutorial-dual-stack-securitygroup**.

f. For **Actions**, choose **Delete security groups**, and then choose **Delete** on the confirmation page.

4. Delete the NAT gateway:

a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

b. Choose **VPC Dashboard**, and then choose **NAT Gateways**.

c. Select the NAT gateway of the VPC that you created. Use the VPC ID to identify the correct NAT gateway.

d. For **Actions**, choose **Delete NAT gateway**.

e. On the confirmation page, enter **`delete`**, and then choose **Delete**.

5. Delete the VPC:

a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.

b. Choose **VPC Dashboard**, and then choose **VPCs**.

c. Select the VPC that you want to delete, such as **tutorial-dual-stack-vpc**.

d. For **Actions**, choose **Delete VPC**.

The confirmation page shows other resources that are associated with the VPC that will also be deleted, including the subnets associated with it.

e. On the confirmation page, enter **`delete`**, and then choose **Delete**.

6. Release the Elastic IP addresses:

    a. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
    b. Choose **EC2 Dashboard**, and then choose **Elastic IPs**.
    c. Select the Elastic IP address that you want to release.
    d. For **Actions**, choose **Release Elastic IP addresses**.
    e. On the confirmation page, choose **Release**.

Moving a DB instance not in a VPC into a VPC

Some legacy DB instances on the EC2-Classic platform are not in a VPC. If your DB instance is not in a VPC, you can use the AWS Management Console to easily move your DB instance into a VPC. Before you can move a DB instance not in a VPC, into a VPC, you must create the VPC.

EC2-Classic was retired on August 15, 2022. If you haven't migrated from EC2-Classic to a VPC, we recommend that you migrate as soon as possible. For more information, see Migrate from EC2-Classic to a VPC in the *Amazon EC2 User Guide* and the blog EC2-Classic Networking is Retiring – Here's How to Prepare.

Important

If you are a new Amazon RDS customer, if you have never created a DB instance before, or if you are creating a DB instance in an AWS Region you have not used before, in almost all cases you are on the *EC2-VPC* platform and have a default VPC. For information about working with DB instances in a VPC, see Working with a DB instance in a VPC.

Follow these steps to create a VPC for your DB instance.

- Step 1: Create a VPC
- Step 2: Create a DB subnet group
- Step 3: Create a VPC security group

After you create the VPC, follow these steps to move your DB instance into the VPC.

- Updating the VPC for a DB instance

We highly recommend that you create a backup of your DB instance immediately before the migration. Doing so ensures that you can restore the data if the migration fails. For more information, see Backing up, restoring, and exporting data.

The following are some limitations to moving your DB instance into the VPC.

- **Previous generation DB instance classes** – Previous generation DB instance classes might not be supported on the VPC platform. When moving a DB instance to a VPC, choose a db.m3 or db.r3 DB instance class. After you move the DB instance to a VPC, you can scale the DB instance to use a later DB instance class. For a full list of VPC supported instance classes, see Amazon RDS instance types.
- **Multi-AZ** – Moving a Multi-AZ DB instance not in a VPC into a VPC is not currently supported. To move your DB instance to a VPC, first modify the DB instance so that it is a single-AZ deployment. Change the **Multi-AZ deployment** setting to **No**. After you move the DB instance to a VPC, modify it again to make it a Multi-AZ deployment. For more information, see Modifying an Amazon RDS DB instance.
- **Read replicas** – Moving a DB instance with read replicas not in a VPC into a VPC is not currently supported. To move your DB instance to a VPC, first delete all of its read replicas. After you move the DB instance to a VPC, recreate the read replicas. For more information, see Working with DB instance read replicas.
- **Option groups** – If you move your DB instance to a VPC, and the DB instance is using a custom option group, change the option group that is associated with your DB instance. Option groups are platform-specific, and moving to a VPC is a change in platform. To use a custom option group in this case, assign the default VPC option group to the DB instance, assign an option group that is used by other DB instances in the VPC you are moving to, or create a new option group and assign it to the DB instance. For more information, see Working with option groups.

Alternatives for moving a DB instance not in a VPC into a VPC with minimal downtime

Using the following alternatives, you can move a DB instance not in a VPC into a VPC with minimal downtime. These alternatives cause minimum disruption to the source DB instance and allow it to serve user traffic during the migration. However, the time required to migrate to a VPC will vary based on the database size and the live workload characteristics.

- **AWS Database Migration Service (AWS DMS)** – AWS DMS enables the live migration of data while keeping the source DB instance fully operational, but it replicates only a limited set of DDL statements. AWS DMS doesn't propagate items such as indexes, users, privileges, stored procedures, and other database changes not directly related to table data. In addition, AWS DMS doesn't automatically use RDS snapshots for the initial DB instance creation, which can increase migration time. For more information, see AWS Database Migration Service.

- **DB snapshot restore or point-in-time recovery** – You can move a DB instance to a VPC by restoring a snapshot of the DB instance or by restoring a DB instance to a point in time. For more information, see Restoring to a DB instance and Restoring a DB instance to a specified time for Amazon RDS.

Quotas and constraints for Amazon RDS

Following, you can find a description of the resource quotas and naming constraints for Amazon RDS.

Topics

- Quotas in Amazon RDS
- Naming constraints in Amazon RDS
- Maximum number of database connections
- File size limits in Amazon RDS

Quotas in Amazon RDS

Each AWS account has quotas, for each AWS Region, on the number of Amazon RDS resources that can be created. After a quota for a resource has been reached, additional calls to create that resource fail with an exception.

The following table lists the resources and their quotas per AWS Region.

Name	Default	Adjustable	Description
Authorizations per DB security group	Each supported Region: 20	No	Number of security group authorizations per DB security group
Custom endpoints per DB cluster	Each supported Region: 5	Yes	The maximum number of custom endpoints that you can create per Aurora DB cluster in this account in the current Region. This value reflects the highest number of custom endpoints in a DB cluster in the account. Other DB

Name	Default	Adjustable	Description
			clusters in the account might have a lower number of custom endpoints.
Custom engine versions	Each supported Region: 40	Yes	The maximum number of custom engine versions allowed in this account in the current Region
DB cluster parameter groups	Each supported Region: 50	Yes	The maximum number of DB cluster parameter groups
DB clusters	Each supported Region: 40	Yes	The maximum number of Aurora clusters allowed in this account in the current Region
DB instances	ap-south-1: 20 Each of the other supported Regions: 40	Yes	The maximum number of DB instances allowed in this account in the current Region
DB shard groups	Each supported Region: 5	Yes	The maximum number of DB shard groups for Aurora Limitless Database in this account in the current Region
DB subnet groups	ap-south-1: 20 Each of the other supported Regions: 50	Yes	The maximum number of DB subnet groups

Name	Default	Adjustable	Description
Data API HTTP request body size	Each supported Region: 4 Megabytes	No	The maximum size allowed for the HTTP request body.
Data API maximum concurrent cluster-secret pairs	Each supported Region: 30	No	The maximum number of unique pairs of Aurora Serverless v1 DB clusters and secrets in concurrent Data API requests for this account in the current AWS Region.
Data API maximum concurrent requests	Each supported Region: 500	No	The maximum number of Data API requests to an Aurora Serverless v1 DB cluster that use the same secret and can be processed at the same time. Additional requests are queued and processed as in-process requests complete.
Data API maximum result set size	Each supported Region: 1 Megabytes	No	The maximum size of the database result set that can be returned by the Data API.
Data API maximum size of JSON response string	Each supported Region: 10 Megabytes	No	The maximum size of the simplified JSON response string returned by the RDS Data API.

Name	Default	Adjustable	Description
Data API requests per second	Each supported Region: 1,000 per second	No	The maximum number of requests to the Data API per second allowed for this account in the current AWS Region. This quota only applies to Amazon Aurora Serverless v1 clusters.
Event subscriptions	Each supported Region: 20	Yes	The maximum number of event subscriptions
IAM roles per DB cluster	Each supported Region: 5	Yes	The maximum number of IAM roles associated with a DB cluster
IAM roles per DB instance	Each supported Region: 5	Yes	The maximum number of IAM roles associated with a DB instance
Integrations	Each supported Region: 100	No	The maximum number of integrations allowed in this account in the current AWS Region
Manual DB cluster snapshots	Each supported Region: 100	Yes	The maximum number of manual DB cluster snapshots
Manual DB instance snapshots	Each supported Region: 100	Yes	The maximum number of manual DB instance snapshots
Option groups	Each supported Region: 20	Yes	The maximum number of option groups

Name	Default	Adjustable	Description
Parameter groups	ap-south-1: 20 Each of the other supported Regions: 50	Yes	The maximum number of parameter groups
Proxies	Each supported Region: 20	Yes	The maximum number of proxies allowed in this account in the current AWS Region
Read replicas per primary	Each supported Region: 15	Yes	The maximum number of read replicas per primary DB instance. This quota cant be adjusted for Amazon Aurora.
Reserved DB instances	ap-south-1: 20 Each of the other supported Regions: 40	Yes	The maximum number of reserved DB instances allowed in this account in the current AWS Region
Security groups	ap-south-1: 20 Each of the other supported Regions: 25	Yes	The maximum number of DB security groups
Subnets per DB subnet group	Each supported Region: 20	No	The maximum number of subnets per DB subnet group

Name	Default	Adjustable	Description
Total storage for all DB instances	Each supported Region: 100,000	Yes	The maximum total storage (in GB) on EBS volumes for all Amazon RDS DB instances added together. This quota does not apply to Amazon Aurora, which has a maximum cluster volume of 128 TiB for each DB cluster.

Note

By default, you can have up to a total of 40 DB instances. RDS DB instances, Aurora DB instances, Amazon Neptune instances, and Amazon DocumentDB instances apply to this quota.
The following limitations apply to the Amazon RDS DB instances:

- 10 for each SQL Server edition (Enterprise, Standard, Web, and Express) under the "license-included" model
- 10 for Oracle under the "license-included" model
- 40 for Db2 under the "bring-your-own-license" (BYOL) licensing model
- 40 for MySQL, MariaDB, or PostgreSQL
- 40 for Oracle under the "bring-your-own-license" (BYOL) licensing model

If your application requires more DB instances, you can request additional DB instances by opening the Service Quotas console. In the navigation pane, choose **AWS services**. Choose **Amazon Relational Database Service (Amazon RDS)**, choose a quota, and follow the directions to request a quota increase. For more information, see Requesting a quota increase in the *Service Quotas User Guide*.
For RDS for Oracle, you can create up to 15 read replicas per source DB instance in each Region, but we recommend limiting replicas to 5 to minimize replication lag.

Backups managed by AWS Backup are considered manual DB snapshots, but don't count toward the manual snapshot quota. For information about AWS Backup, see the *AWS Backup Developer Guide*.

Note that the default quota for cross-Region automated backups is 20 for an AWS account, and the default quota for the number of concurrent snapshot copy requests is 20 for each Region for an AWS account.

If you use any RDS API operations and exceed the default quota for the number of calls per second, the Amazon RDS API issues an error like the following one.

ClientError: An error occurred (ThrottlingException) when calling the *API_name* operation: Rate exceeded.

Here, reduce the number of calls per second. The quota is meant to cover most use cases. If higher quotas are needed, you can request a quota increase by using one of the following options:

- From the console, open the Service Quotas console.
- From the AWS CLI, use the request-service-quota-increase AWS CLI command.

For more information, see the *Service Quotas User Guide*.

Naming constraints in Amazon RDS

The naming constraints in Amazon RDS are as follows:

- DB instance identifier:
  - Must contain 1–63 alphanumeric characters or hyphens.
  - First character must be a letter.
  - Can't end with a hyphen or contain two consecutive hyphens.
  - Must be unique for all DB instances per AWS account, per AWS Region.
- Initial database name:
  - Database name constraints differ for each database engine. For more information, see the available settings when creating each DB instance.
  - SQL Server – Create your databases after creating your DB instance.

- Master username – Master username constraints differ for each database engine. For more information, see the available settings when creating the DB instance.
- Master password:
  - The password for the database master user can include any printable ASCII character except /, ', ", @, or a space.

    For Oracle, & is an additional character limitation.
  - The password can contain the following number of printable ASCII characters depending on the DB engine:
    - Db2: 8–255
    - MariaDB and MySQL: 8–41
    - Oracle: 8–30
    - SQL Server and PostgreSQL: 8–128
- DB parameter group:
  - Must contain 1–255 alphanumeric characters.
  - First character must be a letter.
  - Hyphens are allowed, but the name cannot end with a hyphen or contain two consecutive hyphens.
- DB subnet group:
  - Must contain 1–255 characters.
  - Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.

Maximum number of database connections

The maximum number of simultaneous database connections varies by the DB engine type and the memory allocation for the DB instance class. The maximum number of connections is generally set in the parameter group associated with the DB instance. The exception is Microsoft SQL Server, where it is set in the server properties for the DB instance in SQL Server Management Studio (SSMS).

Database connections consume memory. Setting one of these parameters too high can cause a low memory condition that might cause a DB instance to be placed in the **incompatible-parameters** status. For more information, see Diagnosing and resolving incompatible parameters status for a memory limit.

If your applications frequently open and close connections, or keep a large number of long-lived connections open, we recommend that you use Amazon RDS Proxy. RDS Proxy is a fully managed, highly available database proxy that uses connection pooling to share database connections securely and efficiently. To learn more about RDS Proxy, see Amazon RDS Proxy.

Note

For Oracle, you set the maximum number of user processes and user and system sessions. For Db2, you can't set maximum connections. The limit is 64000.

The following table shows information about the maximum database connections for different DB engines.

DB engine	Parameter	Allowed values	Default value	Description
MariaDB	`max_connections`	1–100000	• For MariaDB 10.5 and higher versions, the default is: LEAST({DBInstanceClassMemory/25165760},12000) The formula is effectively equivalent to MB/25. If the default value calculation results in a value greater than 12,000, Amazon RDS sets the limit to 12,000. • For MariaDB version 10.4:	Number of simultaneous client connections allowed

DB engine	Parameter	Allowed values	Default value	Description
			{DBInstanceClassMemory/12582880} The formula is effectively equivalent to MB/12.	
MySQL	`max_connections`	1–100000	{DBInstanceClassMemory/12582880} The formula is effectively equivalent to MB/12.	Number of simultaneous client connections allowed
Oracle	`processes`	80–20000	LEAST({DBInstanceClassMemory/9868951}, 20000)	User processes
Oracle	`sessions`	100–65535	Not applicable	User and system sessions
PostgreSQL	`max_connections`	6–262143	LEAST({DBInstanceClassMemory/9531392}, 5000)	Maximum number of concurrent connections
SQL Server	`user connections`	0–32767	0 (unlimited)	Maximum number of concurrent connections. For more information, see Configure the user connections (server configuration option).

`DBInstanceClassMemory` is in bytes. For details about how this value is calculated, see Specifying DB parameters. Because of memory reserved for the operating system and RDS

management processes, this memory size is smaller than the value in gibibytes (GiB) shown in Hardware specifications for DB instance classes.

For example, some DB instance classes have 8 GiB of memory, which is 8,589,934,592 bytes. For a MySQL DB instance running on a DB instance class with 8 GiB of memory, such as db.m7g.large, the equation that uses the total memory would be `8589934592/12582880=683`. However, the variable `DBInstanceClassMemory` automatically subtracts the amounts reserved to the operating system and the RDS processes that manage the DB instance. The remainder of the subtraction is then divided by 12,582,880. This calculation results in approximately 630 for the value of `max_connections` instead of 683. This value depends on the DB instance class and DB engine.

When a MariaDB or MySQL DB instance is running on a small DB instance class, such as db.t3.micro or db.t3.small, the total memory available is low. For these DB instance classes, RDS reserves a significant portion of the available memory, which affects the value `max_connections`. For example, the default maximum number of connections for a MySQL DB instance running on a db.t3.micro DB instance class is approximately 60. You can determine the `max_connections` value for your DB MariaDB or MySQL DB instance by connecting to it and running the following SQL command:

```
SHOW GLOBAL VARIABLES LIKE 'max_connections';
```

File size limits in Amazon RDS

File size limits apply to certain Amazon RDS DB instances. For more information, see the following engine-specific limits:

- MariaDB file size limits in Amazon RDS
- MySQL file size limits in Amazon RDS
- Oracle file size limits in Amazon RDS

Troubleshooting for Amazon RDS

Use the following sections to help troubleshoot problems you have with DB instances in Amazon RDS and Amazon Aurora.

Topics

- Can't connect to Amazon RDS DB instance
- Amazon RDS security issues
- Troubleshooting incompatible-network state
- Resetting the DB instance owner password
- Amazon RDS DB instance outage or reboot
- Amazon RDS DB parameter changes not taking effect
- Amazon RDS DB instance running out of storage
- Amazon RDS insufficient DB instances available
- Freeable memory issues in Amazon RDS
- MySQL and MariaDB issues
- Can't set backup retention period to 0

For information about debugging problems using the Amazon RDS API, see Troubleshooting applications on Amazon RDS.

Can't connect to Amazon RDS DB instance

When you can't connect to a DB instance, the following are common causes:

- **Inbound rules** – The access rules enforced by your local firewall and the IP addresses authorized to access your DB instance might not match. The problem is most likely the inbound rules in your security group.

  By default, DB instances don't allow access. Access is granted through a security group associated with the VPC that allows traffic into and out of the DB instance. If necessary, add inbound and outbound rules for your particular situation to the security group. You can specify an IP address, a range of IP addresses, or another VPC security group.

> **Note**
>
> When adding a new inbound rule, you can choose **My IP** for **Source** to allow access to the DB instance from the IP address detected in your browser.

For more information about setting up security groups, see Provide access to your DB instance in your VPC by creating a security group.

> **Note**
>
> Client connections from IP addresses within the range 169.254.0.0/16 aren't permitted. This is the Automatic Private IP Addressing Range (APIPA), which is used for local-link addressing.

- **Public accessibility** – To connect to your DB instance from outside of the VPC, such as by using a client application, the instance must have a public IP address assigned to it.

  To make the instance publicly accessible, modify it and choose **Yes** under **Public accessibility**. For more information, see Hiding a DB instance in a VPC from the internet.

- **Port** – The port that you specified when you created the DB instance can't be used to send or receive communications because of your local firewall restrictions. To determine if your network allows the specified port to be used for inbound and outbound communication, check with your network administrator.

- **Availability** – For a newly created DB instance, the DB instance has a status of `creating` until the DB instance is ready to use. When the state changes to `available`, you can connect to the DB instance. Depending on the size of your DB instance, it can take up to 20 minutes before an instance is available.

- **Internet gateway** – For a DB instance to be publicly accessible, the subnets in its DB subnet group must have an internet gateway.

  To configure an internet gateway for a subnet

  1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
  2. In the navigation pane, choose **Databases**, and then choose the name of the DB instance.

3. In the **Connectivity & security** tab, write down the values of the VPC ID under **VPC** and the subnet ID under **Subnets**.
4. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
5. In the navigation pane, choose **Internet Gateways**. Verify that there is an internet gateway attached to your VPC. Otherwise, choose **Create Internet Gateway** to create an internet gateway. Select the internet gateway, and then choose **Attach to VPC** and follow the directions to attach it to your VPC.
6. In the navigation pane, choose **Subnets**, and then select your subnet.
7. On the **Route Table** tab, verify that there is a route with `0.0.0.0/0` as the destination and the internet gateway for your VPC as the target.

   If you're connecting to your instance using its IPv6 address, verify that there is a route for all IPv6 traffic (`::/0`) that points to the internet gateway. Otherwise, do the following:

   a. Choose the ID of the route table (rtb-*xxxxxxxx*) to navigate to the route table.

   b. On the **Routes** tab, choose **Edit routes**. Choose **Add route**, use `0.0.0.0/0` as the destination and the internet gateway as the target.

      For IPv6, choose **Add route**, use `::/0` as the destination and the internet gateway as the target.

   c. Choose **Save routes**.

   Also, if you are trying to connect to IPv6 endpoint, make sure that client IPv6 address range is authorized to connect to the DB instance.

For more information, see Working with a DB instance in a VPC.

For engine-specific connection issues, see the following topics:

- Troubleshooting connections to your SQL Server DB instance
- Troubleshooting connections to your Oracle DB instance
- Troubleshooting connections to your RDS for PostgreSQL instance
- Maximum MySQL and MariaDB connections

Testing a connection to a DB instance

You can test your connection to a DB instance using common Linux or Microsoft Windows tools.

From a Linux or Unix terminal, you can test the connection by entering the following. Replace *DB-instance-endpoint* with the endpoint and *port* with the port of your DB instance.

```
nc -zv DB-instance-endpoint port
```

For example, the following shows a sample command and the return value.

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299

  Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/
vvr-data] succeeded!
```

Windows users can use Telnet to test the connection to a DB instance. Telnet actions aren't supported other than for testing the connection. If a connection is successful, the action returns no message. If a connection isn't successful, you receive an error message such as the following.

```
C:\>telnet sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com 819

  Connecting To sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com...Could not
 open
  connection to the host, on port 819: Connect failed
```

If Telnet actions return success, your security group is properly configured.

> **Note**
>
> Amazon RDS doesn't accept internet control message protocol (ICMP) traffic, including ping.

Troubleshooting connection authentication

In some cases, you can connect to your DB instance but you get authentication errors. In these cases, you might want to reset the master user password for the DB instance. You can do this by modifying the RDS instance.

For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

Amazon RDS security issues

To avoid security issues, never use your AWS account root user email address and password for a user account. Best practice is to use your root user to create users and assign those to DB user accounts. You can also use your root user to create other user accounts, if necessary.

For information about creating users, see Creating an IAM user in your AWS account. For information about creating users in AWS IAM Identity Center, see Manage identities in IAM Identity Center.

Error message "failed to retrieve account attributes, certain console functions may be impaired."

You can get this error for several reasons. It might be because your account is missing permissions, or your account hasn't been properly set up. If your account is new, you might not have waited for the account to be ready. If this is an existing account, you might lack permissions in your access policies to perform certain actions such as creating a DB instance. To fix the issue, your administrator needs to provide the necessary roles to your account. For more information, see the IAM documentation.

Troubleshooting incompatible-network state

The incompatible-network state means that the database might still be accessible at the database level but you can't modify or reboot it.

Causes

The incompatible-network state of your DB instance could be a result of one of the following actions:

- Modifying the DB instance class.
- Modifying the DB instance to use Multi-AZ DB cluster deployment.
- Replacing a host because of a maintenance event.
- Launching a replacement DB instance.
- Restoring from a snapshot backup.

- Starting a DB instance that was stopped.

Resolution

Use start-db-instance command

To fix a database that is in an incompatible-network state, follow these instructions:

1. Open the https://console.aws.amazon.com/rds/ and choose **Databases** from the navigation pane.
2. Choose the DB instance that is in the incompatible-network state and note the DB instance identifier, VPC ID, and subnet IDs from the **Connectivity & Security** tab.
3. Use the AWS CLI to run the `start-db-instance` command. Specify the `--db-instance-identifier` value.

> **Note**
>
> Running this command when your database is in incompatible mode might cause some downtime.
> The `start-db-instance` command does not resolve this issue for RDS for SQL Server DB instances.

Your database status changes to **Available** if the command executes successfully.

If your database restarts, the DB instance might execute the last operation run on the instance before it was moved to incompatible-network state. This might move the instance back to the incompatible-network state.

If the `start-db-instance` command is unsuccessful or the instance moves back to incompatible-network state, open the **Databases** page in the RDS console and select the database. Navigate to the **Logs & events** section. The **Recent events** section displays further resolution steps to follow. The messages are classified as follows:

- **INTERNAL RESOURCE CHECK**: There might be issues with your internal resources.
- **DNS CHECK**: Check DNS resolution and hostnames for the VPC in the VPC console.
- **ENI CHECK**: The elastic network interface (ENI) for your database might not exist.

- **GATEWAY CHECK**: The internet gateway for your publicly available database is not attached to the VPC.
- **IP CHECK**: There are no free IP addresses in your subnets.
- **SECURITY GROUP CHECK**: There are no security groups associated with your database or the security groups are invalid.
- **SUBNET CHECK**: There are no valid subnets in your DB subnet group or there are issues with your subnet.
- **VPC CHECK**: The VPC associated with your database is invalid.

Perform point-in-time recovery

It is best practice to have a backup (snapshot or logical), in case your database enters incompatible-network state. See Introduction to backups. If you turned on automated backups, then temporarily stop any writes to the database and perform a point-in-time recovery.

> **Note**
>
> After an instance enters the incompatible-network state, the DB instance might not be accessible to perform a logical backup.

If you didn't turn on automated backups, create a new DB instance. Then migrate the data using AWS Database Migration Service (AWS DMS), or by using a backup and restore tool.

If this does not resolve the issue, contact Support for further assistance.

Resetting the DB instance owner password

If you get locked out of your DB instance, you can log in as the master user. Then you can reset the credentials for other administrative users or roles. If you can't log in as the master user, the AWS account owner can reset the master user password. For details of which administrative accounts or roles you might need to reset, see Master user account privileges.

You can change the DB instance password by using the Amazon RDS console, the AWS CLI command modify-db-instance, or by using the ModifyDBInstance API operation. For more information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

Amazon RDS DB instance outage or reboot

A DB instance outage can occur when a DB instance is rebooted. It can also occur when the DB instance is put into a state that prevents access to it, and when the database is restarted. A reboot can occur when you manually reboot your DB instance. A reboot can also occur when you change a DB instance setting that requires a reboot before it can take effect.

A DB instance reboot occurs when you change a setting that requires a reboot, or when you manually cause a reboot. A reboot can occur immediately if you change a setting and request that the change take effect immediately. Or it can occur during the DB instance's maintenance window.

A DB instance reboot occurs immediately when one of the following occurs:

- You change the backup retention period for a DB instance from 0 to a nonzero value or from a nonzero value to 0. You then set **Apply Immediately** to `true`.
- You change the DB instance class, and **Apply Immediately** is set to `true`.
- You change the storage type from **Magnetic (Standard)** to **General Purpose (SSD**) or **Provisioned IOPS (SSD)**, or from **Provisioned IOPS (SSD)** or **General Purpose (SSD)** to **Magnetic (Standard)**.

A DB instance reboot occurs during the maintenance window when one of the following occurs:

- You change the backup retention period for a DB instance from 0 to a nonzero value or from a nonzero value to 0, and **Apply Immediately** is set to `false`.
- You change the DB instance class, and **Apply Immediately** is set to `false`.

When you change a static parameter in a DB parameter group, the change doesn't take effect until the DB instance associated with the parameter group is rebooted. The change requires a manual reboot. The DB instance isn't automatically rebooted during the maintenance window.

To see a table that shows DB instance actions and the effect that setting the **Apply Immediately** value has, see Modifying an Amazon RDS DB instance.

Amazon RDS DB parameter changes not taking effect

In some cases, you might change a parameter in a DB parameter group but don't see the changes take effect. If so, you likely need to reboot the DB instance associated with the DB parameter

group. When you change a dynamic parameter, the change takes effect immediately. When you change a static parameter, the change doesn't take effect until you reboot the DB instance associated with the parameter group.

You can reboot a DB instance using the RDS console. Or you can explicitly call the `RebootDBInstance` API operation. You can reboot without failover if the DB instance is in a Multi-AZ deployment. The requirement to reboot the associated DB instance after a static parameter change helps mitigate the risk of a parameter misconfiguration affecting an API call. An example of this is calling `ModifyDBInstance` to change the DB instance class. For more information, see Modifying parameters in a DB parameter group in Amazon RDS.

Amazon RDS DB instance running out of storage

If your DB instance runs out of storage space, it might no longer be available. We highly recommend that you constantly monitor the `FreeStorageSpace` metric published in CloudWatch to make sure that your DB instance has enough free storage space.

If your database instance runs out of storage, its status changes to `storage-full`. For example, a call to the `DescribeDBInstances` API operation for a DB instance that has used up its storage outputs the following.

```
aws rds describe-db-instances --db-instance-identifier mydbinstance

DBINSTANCE  mydbinstance  2009-12-22T23:06:11.915Z  db.m5.large  mysql8.0  50  sa
storage-full  mydbinstance.clla4j4jgyph.us-east-1.rds.amazonaws.com  3306
us-east-1b  3
 SECGROUP  default  active
 PARAMGRP  default.mysql8.0  in-sync
```

To recover from this scenario, add more storage space to your instance using the `ModifyDBInstance` API operation or the following AWS CLI command.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --allocated-storage 60 \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --allocated-storage 60 ^
    --apply-immediately
```

```
DBINSTANCE  mydbinstance  2009-12-22T23:06:11.915Z  db.m5.large  mysql8.0  50  sa
storage-full  mydbinstance.clla4j4jgyph.us-east-1.rds.amazonaws.com  3306
us-east-1b  3  60
 SECGROUP  default  active
 PARAMGRP  default.mysql8.0  in-sync
```

Now, when you describe your DB instance, you see that your DB instance has `modifying` status, which indicates the storage is being scaled.

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

```
DBINSTANCE  mydbinstance  2009-12-22T23:06:11.915Z  db.m5.large  mysql8.0  50  sa
modifying  mydbinstance.clla4j4jgyph.us-east-1.rds.amazonaws.com
3306  us-east-1b  3  60
 SECGROUP  default  active
 PARAMGRP  default.mysql8.0  in-sync
```

After storage scaling is complete, your DB instance status changes to `available`.

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

```
DBINSTANCE  mydbinstance  2009-12-22T23:06:11.915Z  db.m5.large  mysql8.0  60  sa
available  mydbinstance.clla4j4jgyph.us-east-1.rds.amazonaws.com  3306
us-east-1b  3
 SECGROUP  default  active
 PARAMGRP  default.mysql8.0  in-sync
```

You can receive notifications when your storage space is exhausted using the `DescribeEvents` operation. For example, in this scenario, if you make a `DescribeEvents` call after these operations you see the following output.

```
aws rds describe-events --source-type db-instance --source-identifier mydbinstance
```

```
2009-12-22T23:44:14.374Z  mydbinstance  Allocated storage has been exhausted db-
instance
2009-12-23T00:14:02.737Z  mydbinstance  Applying modification to allocated storage db-
instance
2009-12-23T00:31:54.764Z  mydbinstance  Finished applying modification to allocated
 storage
```

Amazon RDS insufficient DB instances available

The `InsufficientDBInstanceCapacity` error can be returned when you try to create, start, or modify a DB instance. It can also be returned when you try to restore a DB instance from a DB snapshot. When this error is returned, a common cause is that the specific DB instance class isn't available in the requested Availability Zone. You can try one of the following to solve the problem:

- Retry the request with a different DB instance class.
- Retry the request with a different Availability Zone.
- Retry the request without specifying an explicit Availability Zone.

For information about troubleshooting instance capacity issues for Amazon EC2, see Insufficient instance capacity in the *Amazon EC2 User Guide*.

For information about modifying a DB instance, see Modifying an Amazon RDS DB instance.

Freeable memory issues in Amazon RDS

Freeable memory is the total random access memory (RAM) on a DB instance that can be made available to the database engine. It's the sum of the free operating-system (OS) memory and the available buffer and page cache memory. The database engine uses most of the memory on the host, but OS processes also use some RAM. Memory currently allocated to the database engine or used by OS processes isn't included in freeable memory. When the database engine is running out of memory, the DB instance can use the temporary space that is normally used for buffering and caching. As previously mentioned, this temporary space is included in freeable memory.

You use the `FreeableMemory` metric in Amazon CloudWatch to monitor the freeable memory. For more information, see Monitoring tools for Amazon RDS.

If your DB instance consistently runs low on freeable memory or uses swap space, consider scaling up to a larger DB instance class. For more information, see DB instance classes.

You can also change the memory settings. For example, on RDS for MySQL, you might adjust the size of the `innodb_buffer_pool_size` parameter. This parameter is set by default to 75 percent of physical memory. For more MySQL troubleshooting tips, see How can I troubleshoot low freeable memory in an Amazon RDS for MySQL database?

MySQL and MariaDB issues

You can diagnose and correct issues with MySQL and MariaDB DB instances.

Topics

- Maximum MySQL and MariaDB connections
- Diagnosing and resolving incompatible parameters status for a memory limit
- Diagnosing and resolving lag between read replicas
- Diagnosing and resolving a MySQLor MariaDB read replication failure
- Creating triggers with binary logging enabled requires SUPER privilege
- Diagnosing and resolving point-in-time restore failures
- Replication stopped error
- Read replica create fails or replication breaks with fatal error 1236
- Read replica replication fails to initialize metadata structure

Maximum MySQL and MariaDB connections

The maximum number of connections allowed to an RDS for MySQL or RDS for MariaDB DB instance is based on the amount of memory available for its DB instance class. A DB instance class with more memory available results in a larger number of connections available. For more information on DB instance classes, see DB instance classes.

The connection limit for a DB instance is set by default to the maximum for the DB instance class. You can limit the number of concurrent connections to any value up to the maximum number of connections allowed. Use the `max_connections` parameter in the parameter group for the DB instance. For more information, see Maximum number of database connections and Parameter groups for Amazon RDS.

You can retrieve the maximum number of connections allowed for a MySQL or MariaDB DB instance by running the following query.

```
SELECT @@max_connections;
```

You can retrieve the number of active connections to a MySQL or MariaDB DB instance by running the following query.

```
SHOW STATUS WHERE `variable_name` = 'Threads_connected';
```

Diagnosing and resolving incompatible parameters status for a memory limit

A MariaDB or MySQL DB instance can be placed in **incompatible-parameters** status for a memory limit when the following conditions are met:

- The DB instance is restarted at least three times in one hour or at least five times in one day when the DB instance status is **Available**.
- An attempt to restart the DB instance fails because a maintenance action or monitoring process couldn't restart the DB instance.
- The potential memory usage of the DB instance exceeds 1.2 times the memory allocated to its DB instance class.

When a DB instance is restarted for the third time in one hour or for the fifth time in one day, it performs a check for memory usage. The check makes a calculation of the potential memory usage of the DB instance. The value returned by the calculation is the sum of the following values:

- **Value 1** – The sum of the following parameters:
  - `innodb_additional_mem_pool_size`
  - `innodb_buffer_pool_size`

    You can modify the value for `innodb_buffer_pool_size`. However, the value won't always match what you input. This mismatch occurs for several reasons. First, if the DB instance is a micro DB instance, then we override the default value and set it to 256 MB. For more information, see Overriding innodb_buffer_pool_size.

    Second, we make sure that 500 MB of memory is reserved on the DB instance for the host manager, the engine, the operating system, and the kernel.

Last, we optimize innodb_buffer_pool_size by dividing it into units. The host manager rounds down to the closest multiple of those units. The units are calculated by multiplying innodb_buffer_pool_chunk_size by innodb_buffer_pool_instances. For more information, see Configuring InnoDB Buffer Pool Size in the MySQL documentation.

The default for innodb_buffer_pool_instances is 8, unless innodb_buffer_pool_size is less than 1 GB. If innodb_buffer_pool_size is less than 1 GB, then the default for innodb_buffer_pool_instances is 1. The default for innodb_buffer_pool_chunk_size is 128 MB.

  - innodb_log_buffer_size
  - key_buffer_size
  - query_cache_size (MySQL version 5.7 only)
  - tmp_table_size
- **Value 2** – The max_connections parameter multiplied by the sum of the following parameters:
  - binlog_cache_size
  - join_buffer_size
  - read_buffer_size
  - read_rnd_buffer_size
  - sort_buffer_size
  - thread_stack
- **Value 3** – If the performance_schema parameter is enabled, then multiply the max_connections parameter by 429498.

  If the performance_schema parameter is disabled, then this value is zero.

So, the value returned by the calculation is the following:

```
Value 1 + Value 2 + Value 3
```

When this value exceeds 1.2 times the memory allocated to the DB instance class used by the DB instance, the DB instance is placed in **incompatible-parameters** status. For information about the memory allocated to DB instance classes, see Hardware specifications for DB instance classes.

The calculation multiplies the value of the max_connections parameter by the sum of several parameters. If the max_connections parameter is set to a large value, it might cause the check to

return an inordinately high value for the potential memory usage of the DB instance. In this case, consider lowering the value of the `max_connections` parameter.

To resolve the problem, complete the following steps:

1. Adjust the memory parameters in the DB parameter group associated with the DB instance. Do so such that the potential memory usage is lower than 1.2 times the memory allocated to its DB instance class.

   For information about setting parameters, see Modifying parameters in a DB parameter group in Amazon RDS.
2. Restart the DB instance.

   For information about setting parameters, see Starting an Amazon RDS DB instance that was previously stopped.

Diagnosing and resolving lag between read replicas

After you create a MySQL or MariaDB read replica and the replica is available, Amazon RDS first replicates the changes made to the source DB instance from the time the read replica create operation started. During this phase, the replication lag time for the read replica is greater than 0. You can monitor this lag time in Amazon CloudWatch by viewing the Amazon RDS `ReplicaLag` metric.

The `ReplicaLag` metric reports the value of the `Seconds_Behind_Master` field of the MariaDB or MySQL `SHOW REPLICA STATUS` command. For more information, see SHOW REPLICA STATUS Statement in the MySQL documentation.

When the `ReplicaLag` metric reaches 0, the replica has caught up to the source DB instance. If the `ReplicaLag` metric returns -1, replication might not be active. To troubleshoot a replication error, see Diagnosing and resolving a MySQLor MariaDB read replication failure. A `ReplicaLag` value of -1 can also mean that the `Seconds_Behind_Master` value can't be determined or is `NULL`.

> **Note**
>
> Previous versions of MariaDB used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using a MariaDB version lower than 10.5, then use `SHOW SLAVE STATUS`.

The `ReplicaLag` metric returns -1 during a network outage or when a patch is applied during the maintenance window. In this case, wait for network connectivity to be restored or for the maintenance window to end before you check the `ReplicaLag` metric again.

The MySQL and MariaDB read replication technology is asynchronous. Thus, you can expect occasional increases for the `BinLogDiskUsage` metric on the source DB instance and for the `ReplicaLag` metric on the read replica. For example, consider a situation where a high volume of write operations to the source DB instance occur in parallel. At the same time, write operations to the read replica are serialized using a single I/O thread. Such a situation can lead to a lag between the source instance and read replica.

For more information about read replicas and MySQL, see Replication implementation details in the MySQL documentation. For more information about read replicas and MariaDB, see Replication overview in the MariaDB documentation.

You can reduce the lag between updates to a source DB instance and the subsequent updates to the read replica by doing the following:

- Set the DB instance class of the read replica to have a storage size comparable to that of the source DB instance.
- Make sure that parameter settings in the DB parameter groups used by the source DB instance and the read replica are compatible. For more information and an example, see the discussion of the `max_allowed_packet` parameter in the next section.
- Disable the query cache. For tables that are modified often, using the query cache can increase replica lag because the cache is locked and refreshed often. If this is the case, you might see less replica lag if you disable the query cache. You can disable the query cache by setting the `query_cache_type parameter` to 0 in the DB parameter group for the DB instance. For more information on the query cache, see Query cache configuration.
- Warm the buffer pool on the read replica for InnoDB for MySQL or MariaDB. For example, suppose that you have a small set of tables that are being updated often and you're using the InnoDB or XtraDB table schema. In this case, dump those tables on the read replica. Doing this causes the database engine to scan through the rows of those tables from the disk and then cache them in the buffer pool. This approach can reduce replica lag. The following shows an example.

  For Linux, macOS, or Unix:

```
PROMPT> mysqldump \
```

```
    -h <endpoint> \
    --port=<port> \
    -u=<username> \
    -p <password> \
    database_name table1 table2 > /dev/null
```

For Windows:

```
PROMPT> mysqldump ^
    -h <endpoint> ^
    --port=<port> ^
    -u=<username> ^
    -p <password> ^
    database_name table1 table2 > /dev/null
```

Diagnosing and resolving a MySQLor MariaDB read replication failure

Amazon RDS monitors the replication status of your read replicas. RDS updates the **Replication State** field of the read replica instance to `Error` if replication stops for any reason. You can review the details of the associated error thrown by the MySQL or MariaDB engines by viewing the **Replication Error** field. Events that indicate the status of the read replica are also generated, including RDS-EVENT-0045, RDS-EVENT-0046, and RDS-EVENT-0057. For more information about events and subscribing to events, see Working with Amazon RDS event notification. If a MySQL error message is returned, check the error in the MySQL error message documentation. If a MariaDB error message is returned, check the error in the MariaDB error message documentation.

Common situations that can cause replication errors include the following:

- The value for the `max_allowed_packet` parameter for a read replica is less than the `max_allowed_packet` parameter for the source DB instance.

  The `max_allowed_packet` parameter is a custom parameter that you can set in a DB parameter group. The `max_allowed_packet` parameter is used to specify the maximum size of data manipulation language (DML) that can be run on the database. In some cases, the `max_allowed_packet` value for the source DB instance might be larger than the `max_allowed_packet` value for the read replica. If so, the replication process can throw an error and stop replication. The most common error is `packet bigger than 'max_allowed_packet' bytes`. You can fix the error by having the source and read replica use DB parameter groups with the same `max_allowed_packet` parameter values.

- Writing to tables on a read replica. If you're creating indexes on a read replica, you need to have the `read_only` parameter set to *0* to create the indexes. If you're writing to tables on the read replica, it can break replication.
- Using a nontransactional storage engine such as MyISAM. Read replicas require a transactional storage engine. Replication is only supported for the following storage engines: InnoDB for MySQL or MariaDB.

  You can convert a MyISAM table to InnoDB with the following command:

  ```
  alter table <schema>.<table_name> engine=innodb;
  ```
- Using unsafe nondeterministic queries such as `SYSDATE()`. For more information, see Determination of safe and unsafe statements in binary logging in the MySQL documentation.

The following steps can help resolve your replication error:

- If you encounter a logical error and you can safely skip the error, follow the steps described in Skipping the current replication error for RDS for MySQL. Your MySQL or MariaDB DB instance must be running a version that includes the `mysql_rds_skip_repl_error` procedure. For more information, see mysql.rds_skip_repl_error.
- If you encounter a binary log (binlog) position issue, you can change the replica replay position with the mysql.rds_next_source_log (RDS for MySQL major versions 8.4 and higher) or mysql.rds_next_master_log (RDS for MariaDB and RDS for MySQL major versions 8.0 and lower) command.
- You might encounter a temporary performance issue because of high DML load. If so, you can set the `innodb_flush_log_at_trx_commit` parameter to 2 in the DB parameter group on the read replica. Doing this can help the read replica catch up, though it temporarily reduces atomicity, consistency, isolation, and durability (ACID).
- You can delete the read replica and create an instance using the same DB instance identifier. If you do this, the endpoint remains the same as that of your old read replica.

If a replication error is fixed, the **Replication State** changes to **replicating**. For more information, see Troubleshooting a MySQL read replica problem.

Creating triggers with binary logging enabled requires SUPER privilege

When trying to create triggers in an RDS for MySQL or RDS for MariaDB DB instance, you might receive the following error.

```
"You do not have the SUPER privilege and binary logging is enabled"
```

To use triggers when binary logging is enabled requires the SUPER privilege, which is restricted for RDS for MySQL and RDS for MariaDB DB instances. You can create triggers when binary logging is enabled without the SUPER privilege by setting the `log_bin_trust_function_creators` parameter to true. To set the `log_bin_trust_function_creators` to true, create a new DB parameter group or modify an existing DB parameter group.

You can create a new DB parameter group so you can create triggers in your RDS for MySQL or RDS for MariaDB DB instance with binary logging enabled. To do so, use the following CLI commands. To modify an existing parameter group, start with step 2.

To create a new parameter group to allow triggers with binary logging enabled using the CLI

1. Create a new parameter group.

   For Linux, macOS, or Unix:

   ```
   aws rds create-db-parameter-group \
       --db-parameter-group-name allow-triggers \
       --db-parameter-group-family mysql8.0 \
       --description "parameter group allowing triggers"
   ```

   For Windows:

   ```
   aws rds create-db-parameter-group ^
       --db-parameter-group-name allow-triggers ^
       --db-parameter-group-family mysql8.0 ^
       --description "parameter group allowing triggers"
   ```

2. Modify the DB parameter group to allow triggers.

   For Linux, macOS, or Unix:

   ```
   aws rds modify-db-parameter-group \
   ```

```
    --db-parameter-group-name allow-triggers \
    --parameters "ParameterName=log_bin_trust_function_creators,
 ParameterValue=true, ApplyMethod=pending-reboot"
```

For Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name allow-triggers ^
    --parameters "ParameterName=log_bin_trust_function_creators,
 ParameterValue=true, ApplyMethod=pending-reboot"
```

3. Modify your DB instance to use the new DB parameter group.

   For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-parameter-group-name allow-triggers \
    --apply-immediately
```

   For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-parameter-group-name allow-triggers ^
    --apply-immediately
```

4. For the changes to take effect, manually reboot the DB instance.

```
aws rds reboot-db-instance --db-instance-identifier mydbinstance
```

Diagnosing and resolving point-in-time restore failures

Restoring a DB instance that includes temporary tables

When attempting a point-in-time restore (PITR) of your MySQL or MariaDB DB instance, you might encounter the following error.

```
Database instance could not be restored because there has been incompatible database
 activity for restore
```

```
functionality. Common examples of incompatible activity include using temporary tables,
 in-memory tables,
or using MyISAM tables. In this case, use of Temporary table was detected.
```

PITR relies on both backup snapshots and binary logs (binlogs) from MySQL or MariaDB to restore your DB instance to a particular time. Temporary table information can be unreliable in binlogs and can cause a PITR failure. If you use temporary tables in your MySQL or MariaDB DB instance, you can decrease the possibility of a PITR failure by performing more frequent backups. A PITR failure is most probable in the time between a temporary table's creation and the next backup snapshot.

Restoring a DB instance that includes in-memory tables

You might encounter a problem when restoring a database that has in-memory tables. In-memory tables are purged during a restart. As a result, your in-memory tables might be empty after a reboot. We recommend that when you use in-memory tables, you architect your solution to handle empty tables in the event of a restart. If you're using in-memory tables with replicated DB instances, you might need to recreate the read replicas after a restart. This might be necessary if a read replica reboots and can't restore data from an empty in-memory table.

For more information about backups and PITR, see Introduction to backups and Restoring a DB instance to a specified time for Amazon RDS.

Replication stopped error

When you call the `mysql.rds_skip_repl_error` command, you might receive an error message stating that replication is down or disabled.

This error message appears because replication is stopped and can't be restarted.

If you need to skip a large number of errors, the replication lag can increase beyond the default retention period for binary log files. In this case, you might encounter a fatal error because of binary log files being purged before they have been replayed on the replica. This purge causes replication to stop, and you can no longer call the `mysql.rds_skip_repl_error` command to skip replication errors.

You can mitigate this issue by increasing the number of hours that binary log files are retained on your replication source. After you have increased the binlog retention time, you can restart replication and call the `mysql.rds_skip_repl_error` command as needed.

To set the binlog retention time, use the mysql.rds_set_configuration procedure. Specify a configuration parameter of 'binlog retention hours' along with the number of hours to retain

binlog files on the DB cluster, up to 720 (30 days). The following example sets the retention period for binlog files to 48 hours.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

Read replica create fails or replication breaks with fatal error 1236

After changing default parameter values for a MySQL or MariaDB DB instance, you might encounter one of the following problems:

- You can't create a read replica for the DB instance.
- Replication fails with `fatal error 1236`.

Some default parameter values for MySQL and MariaDB DB instances help to make sure that the database is ACID compliant and read replicas are crash-safe. They do this by making sure that each commit is fully synchronized by writing the transaction to the binary log before it's committed. Changing these parameters from their default values to improve performance can cause replication to fail when a transaction hasn't been written to the binary log.

To resolve this issue, set the following parameter values:

- `sync_binlog = 1`
- `innodb_support_xa = 1`
- `innodb_flush_log_at_trx_commit = 1`

Read replica replication fails to initialize metadata structure

When you attempted to start replication, you received the following error message:

```
Read Replica Replication Error - SQLError: 13124, reason: Replica failed to initialize
 applier metadata structure from the repository
```

This error occurs when there is a problem with the metadata structure of the replica. To fix the metadata structure, you must create a new replica.

To prevent this from happening in the future, perform one of the following actions:

- If possible, disable multi-threading on your replicas. Starting with MySQL 8.0.27, multi-threading is enabled by default.
- If you need to use multi-threading on your replicas, then we recommend that you use GTID-based replication. For more information, see Using GTID-based replication.

Can't set backup retention period to 0

There are several reasons why you might need to set the backup retention period to 0. For example, you can disable automatic backups immediately by setting the retention period to 0.

In some cases, you might set the value to 0 and receive a message saying that the retention period must be between 1 and 35. In these cases, check to make sure that you haven't set up a read replica for the instance. Read replicas require backups for managing read replica logs, and therefore you can't set a retention period of 0.

Amazon RDS API reference

In addition to the AWS Management Console and the AWS Command Line Interface (AWS CLI), Amazon RDS also provides an API. You can use the API to automate tasks for managing your DB instances and other objects in Amazon RDS.

- For an alphabetical list of API operations, see Actions.
- For an alphabetical list of data types, see Data types.
- For a list of common query parameters, see Common parameters.
- For descriptions of the error codes, see Common errors.

For more information about the AWS CLI, see AWS Command Line Interface reference for Amazon RDS.

Topics

- Using the Query API
- Troubleshooting applications on Amazon RDS

Using the Query API

The following sections briefly discuss the parameters and request authentication used with the Query API.

For general information about how the Query API works, see Query requests in the *Amazon EC2 API Reference*.

Query parameters

HTTP Query-based requests are HTTP requests that use the HTTP verb GET or POST and a Query parameter named `Action`.

Each Query request must include some common parameters to handle authentication and selection of an action.

Some operations take lists of parameters. These lists are specified using the `param.`*n* notation. Values of *n* are integers starting from 1.

For information about Amazon RDS Regions and endpoints, go to Amazon Relational Database Service (RDS) in the Regions and Endpoints section of the *Amazon Web Services General Reference*.

Query request authentication

You can only send Query requests over HTTPS, and you must include a signature in every Query request. You must use either AWS signature version 4 or signature version 2. For more information, see Signature Version 4 signing process and Signature version 2 signing process.

Troubleshooting applications on Amazon RDS

Amazon RDS provides specific and descriptive errors to help you troubleshoot problems while interacting with the Amazon RDS API.

Topics

- Retrieving errors
- Troubleshooting tips

For information about troubleshooting for Amazon RDS DB instances, see Troubleshooting for Amazon RDS.

Retrieving errors

Typically, you want your application to check whether a request generated an error before you spend any time processing results. The easiest way to find out if an error occurred is to look for an `Error` node in the response from the Amazon RDS API.

XPath syntax provides a simple way to search for the presence of an `Error` node. It also provides a relatively easy way to retrieve the error code and message. The following code snippet uses Perl and the XML::XPath module to determine if an error occurred during a request. If an error occurred, the code prints the first error code and message in the response.

```
use XML::XPath;
    my $xp = XML::XPath->new(xml =>$response);
    if ( $xp->find("//Error") )
    {print "There was an error processing your request:\n", " Error code: ",
    $xp->findvalue("//Error[1]/Code"), "\n", " ",
    $xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

Troubleshooting tips

We recommend the following processes to diagnose and resolve problems with the Amazon RDS API:

- Verify that Amazon RDS is operating normally in the AWS Region that you're targeting by checking http://status.aws.amazon.com.
- Check the structure of your request.

  Each Amazon RDS operation has a reference page in the *Amazon RDS API Reference*. Double-check that you are using parameters correctly. For ideas about what might be wrong, look at the sample requests or user scenarios to see if those examples do similar operations.
- Check AWS re:Post.

  Amazon RDS has a development community where you can search for solutions to problems others have experienced along the way. To view the topics, go to AWS re:Post.

Document history

Current API version: 2014-10-31

The following table describes important changes in each release of the *Amazon RDS User Guide* after May 2018. For notification about updates to this documentation, you can subscribe to an RSS feed.

> **Note**
>
> You can filter new Amazon RDS features on the What's New with Database? page. For **Products**, choose **Amazon RDS**. Then search using keywords such as **`RDS Proxy`** or **`Oracle 2023`**.

Change	Description	Date
Amazon RDS Extended Support version 5.7.44-RDS.20260212 for RDS for MySQL	The RDS Extended Support version 5.7.44-RDS.20260212 is now available for RDS for MySQL. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	February 26, 2026
Amazon RDS supports MariaDB versions 11.8.6, 11.4.10, 10.11.16, and 10.6.25	RDS for MariaDB now supports MariaDB versions 11.8.6, 11.4.10, 10.11.16, and 10.6.25. For more information, see MariaDB on Amazon RDS versions.	February 10, 2026
Amazon RDS supports MySQL 8.4.8 and 8.0.45	You can now create Amazon RDS DB instances running MySQL 8.4.8 and 8.0.45. For more information, see	February 3, 2026

	MySQL on Amazon RDS versions.	
Amazon RDS Extended Support version 5.7.44-RDS.20251212 for RDS for MySQL	The RDS Extended Support version 5.7.44-RDS.20251212 is now available for RDS for MySQL. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	December 12, 2025
RDS for SQL Server Developer Edition.	RDS for SQL Server now supports SQL Server Developer Edition. For more information, see Working with SQL Server Developer Edition on RDS for SQL Server.	December 2, 2025
Amazon RDS supports MySQL 9.5 in the Database Preview environment	MySQL 9.5 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 9.5 in the Database Preview environment.	November 21, 2025
Amazon RDS supports MySQL 8.4.7 and 8.0.44	You can now create Amazon RDS DB instances running MySQL 8.4.7 and 8.0.44. For more information, see MySQL on Amazon RDS versions.	November 18, 2025

Amazon RDS supports MariaDB versions 11.4.9, 10.11.15, and 10.6.24	RDS for MariaDB now supports MariaDB versions 11.4.9, 10.11.15, and 10.6.24. For more information, see MariaDB on Amazon RDS versions.	November 18, 2025
Amazon RDS supports additional storage volumes for Oracle and SQL Server	You can add up to three additional storage volumes to an RDS for Oracle or RDS for SQL Server DB instance. Depending on your workload requirements, choose between gp3 or io2 storage for each volume. Scale your total storage up to 256 TiB per DB instance by combining primary and additional storage volumes. For more information, see Additional storage volumes.	November 14, 2025
RDS for Oracle supports db.r7i preconfigured instance classes	RDS for Oracle now supports db.r7i preconfigured instances for Enterprise Edition and Standard Edition 2 with BYOL. These instance classes provide additional memory for Oracle database workloads. For example, db.r7i.xlarge.tpc2.mem2x provides twice as much memory as a standard db.r7i.xlarge instance and enables multithreading. For more information, see DB instance class support.	October 2, 2025

Amazon RDS for Db2 supports backing up a single Db2 database	You can now use the `rdsadmin.backup_database` stored procedure to back up a single Db2 database on an RDS for Db2 DB instance to an Amazon S3 bucket. For more information, see rdsadmin.backup_database.	September 30, 2025
Amazon RDS supports reserved DB instances for Amazon RDS for Db2	You can now purchase reserved DB instances for RDS for Db2. For more information, see Reserved DB instances for Amazon RDS.	September 24, 2025
Amazon RDS for Db2 supports Amazon S3 streaming to restore databases	RDS for Db2 now supports restoring a database directly from multiple database backup files in an Amazon S3 bucket. For more information, see rdsadmin.restore_database.	September 22, 2025
Amazon RDS for Db2 supports Amazon S3 streaming to restore databases	RDS for Db2 now supports restoring a database directly from multiple database backup files in an Amazon S3 bucket. For more information, see rdsadmin.restore_database.	September 22, 2025

Amazon RDS supports MySQL 9.4 in the Database Preview environment	MySQL 9.4 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 9.4 in the Database Preview environment.	September 19, 2025
Amazon RDS Extended Support version 5.7.44-RDS.20250818 for RDS for MySQL	The RDS Extended Support version 5.7.44-RDS.20250818 is now available for RDS for MySQL. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	September 15, 2025
RDS for Oracle supports db.m7i and db.r7i for SE2 LI	RDS for Oracle Standard Edition 2 (SE2) using the License Included (LI) model supports the db.m7i and db.r7i instance classes in the Asia Pacific (Thailand) and Mexico (Central) Region. For more information, see Oracle DB instance classes.	August 28, 2025
Amazon RDS is available in the Asia Pacific (New Zealand) Region	Amazon RDS is now available in the Asia Pacific (New Zealand) Region. For more information, see Regions and Availability Zones.	August 28, 2025

RDS for Oracle and RDS Custom for Oracle support bare metal instance classes	RDS for Oracle and RDS Custom for Oracle now support bare metal instance classes for Enterprise Edition with BYOL. Bare metal instances give your applications direct access to the processor and memory resources of the underlying server. For more information, see DB instance classes and DB instance class support for RDS Custom for Oracle.	August 27, 2025
Amazon RDS supports MariaDB 11.8	You can now create Amazon RDS DB instances running MariaDB version 11.8. For more information, see MariaDB on Amazon RDS versions.	August 25, 2025
Amazon RDS for Oracle supports ECDSA cipher suites and ECC certificates	Amazon RDS for Oracle now supports Elliptic Curve Digital Signature Algorithm (ECDSA) cipher suites for the SSL and OEM Agent options. These cipher suites are compatible with ECC certificates. For more information, see Certificate compatibility with cipher suites for the SSL option and Certificate compatibility with cipher suites for the OEM Agent option.	August 25, 2025

Amazon RDS for Db2 supports cross-Region read-only replicas	You can now create RDS for Db2 cross-Region read-only replicas. For more informati on, see Working with replicas for Amazon RDS for Db2.	August 22, 2025
Amazon RDS supports MariaDB 11.4.8, 10.11.14, and 10.6.23	You can now create Amazon RDS DB instances running MariaDB version 11.4.8, 10.11.14, and 10.6.2. For more information, see MariaDB on Amazon RDS versions.	August 14, 2025
Amazon RDS Custom service-l inked role policy updated	The AmazonRDSCustomSer viceRolePolicy managed policy has been updated to comply with EBS CopySnaps hot authorization behavior. The update removes `ec2:CopySnapshot` from one statement and adds two new statement s for source and destinati on snapshot permissions. For more information, see Updates to AWS managed policies for Amazon RDS.	August 7, 2025

Amazon RDS supports MySQL 8.4.6 and 8.0.43	You can now create Amazon RDS DB instances running MySQL 8.4.6 and 8.0.43. MySQL 8.4.6 includes the addition of the rds_security_audit database plugin, an internal RDS MySQL plugin that collects security-related metrics. For more information, see MySQL on Amazon RDS versions.	August 1, 2025
RDS for Oracle supports the db.m6in and db.r6in DB instance classes	You can use the db.m6in and db.r6in DB instance classes with RDS for Oracle with the Bring Your Own License (BYOL) model. For more information, see RDS for Oracle DB instance classes.	July 29, 2025
Zero-ETL integrations from RDS for PostgreSQL and RDS for Oracle are now available	You can now create Zero-ETL integrations with RDS for PostgreSQL and RDS for Oracle as the source. For more information, see Amazon RDS zero-ETL integrations.	July 23, 2025
Amazon RDS for Db2 supports group authorization using Kerberos	You can now perform group authorization using Kerberos with your self-managed Active Directory. For more information, see Setting up Kerberos authentication for RDS for Db2 DB instances.	July 21, 2025

Zero-ETL integrations with Amazon SageMaker AI is generally available	Zero-ETL integrations make transactional data availabl e in an Amazon SageMaker lakehouse within seconds of it being written to an RDS for MySQL DB instance. For more information, see Working with Amazon RDS zero-ETL integrations.	June 30, 2025
Amazon RDS supports MySQL 9.3 in the Database Preview environment	MySQL 9.3 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 9.3 in the Database Preview environment.	June 16, 2025
Amazon RDS for Db2 supports cross-Region replicas in standby mode	You can now create RDS for Db2 cross-Region replicas in standby mode for disaster recovery. For more informati on, see Working with replicas for RDS for Db2.	June 11, 2025
Amazon RDS is available in the Asia Pacific (Taipei) Region	Amazon RDS is now available in the Asia Pacific (Taipei) Region. For more information, see Regions and Availability Zones.	June 5, 2025
Amazon RDS supports MariaDB 10.11.13 and 11.4.7	You can now create Amazon RDS DB instances running MariaDB version 10.11.13 and 11.4.7. For more information, see MariaDB on Amazon RDS versions.	June 4, 2025

End-of-life information for Performance Insights	AWS has announced the end-of-life date for Performance Insights: November 30, 2025. After this date, Amazon RDS will no longer support the Performance Insights console experience, flexible retention periods (1-24 months), and their associated pricing. We recommend upgrading your DB instances to Database Insights before this date. For more information, see Monitoring DB load with Performance Insights on Amazon RDS.	May 30, 2025
Amazon RDS Extended Support version 5.7.44-RDS.20250508 for RDS for MySQL	The RDS Extended Support version 5.7.44-RDS.20250508 is now available for RDS for MySQL. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	May 27, 2025
RDS Custom for Oracle supports the db.m7i and db.r7i instance classes	You can now use the db.m7i and db.r7i instance classes for RDS Custom for Oracle. For more information, see DB instance class support for RDS Custom for Oracle.	May 22, 2025

Viewing support dates for open source engine major versions	You can now view information about the start and end dates for open source engine major versions in RDS standard support and RDS Extended Support by using the AWS CLI and the RDS API. For more information, see Viewing support dates for engine versions in Amazon RDS Extended Support.	May 20, 2025
Amazon RDS supports MariaDB 10.5.29 and 10.6.22	You can now create Amazon RDS DB instances running MariaDB version 10.5.29 and 10.6.22. For more informati on, see MariaDB on Amazon RDS versions.	May 20, 2025
Amazon RDS supports MySQL 8.0.42	You can now create Amazon RDS DB instances running MySQL 8.0.42. For more information, see MySQL on Amazon RDS versions.	May 12, 2025
Amazon RDS supports MySQL 8.4.5	You can now create Amazon RDS DB instances running MySQL 8.4.5. For more information, see MySQL on Amazon RDS versions.	May 1, 2025

Amazon RDS supports backup replication for Asia Pacific (Hong Kong), Asia Pacific (Melbourne), Asia Pacific (Malaysia), Canada West (Calgary), and Europe (Zurich)	Backup replication is now available for databases in the Asia Pacific (Hong Kong), Asia Pacific (Melbourne), Asia Pacific (Malaysia), Canada West (Calgary), and Europe (Zurich) Regions. For more information about available Regions, see Replicating automated backups to another AWS Region.	April 30, 2025
Amazon RDS supports managed master user passwords in the Oracle multitenant architecture	You can use AWS Secrets Manager to manage master user passwords for RDS for Oracle container databases (CDBs) and pluggable databases (PDBs). The single-tenant and multi-tenant configurations of the CDB architecture both support managed passwords. For more information, see User accounts and privileges in a CDB.	April 24, 2025
Amazon RDS for Oracle supports m6id and r6id instance classes for BYOL	You can now create RDS for Oracle DB instances using the m6id and r6id instance classes on the BYOL model. Enterprise and Standard Edition 2 are both supported. For more information, see Supported edition, instance class, and licensing combinations in RDS for Oracle.	April 9, 2025

Amazon RDS supports MariaDB 11.8 in the Database Preview environment	MariaDB 11.8 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MariaDB version 11.8 in the Database Preview environment.	April 2, 2025
Update to IAM service-linked role permissions	The `AmazonRDSCustomServiceRolePolicy` policy now grants additional permissions to manage EC2 key-pair and integrate with Amazon SQS. For more information, see Amazon RDS updates to AWS managed policies.	March 25, 2025
Amazon RDS supports MySQL 9.2 in the Database Preview environment	MySQL 9.2 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 9.2 in the Database Preview environment.	March 24, 2025

Update to existing policy	Amazon RDS added new permissions to the managed policy `AmazonRDSCustomInstanceProfileRolePolicy` to allow the usage of RDS Custom managed secrets on an RDS Custom instance. For more information, see AWS managed policy: AmazonRDSCustomInstanceProfileRolePolicy.	March 20, 2025
Amazon RDS Extended Support version 5.7.44-RDS.20250213 for RDS for MySQL	The RDS Extended Support version 5.7.44-RDS.20250213 is now available for RDS for MySQL. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	March 12, 2025
Update to IAM service-linked role permissions	The `AmazonRDSCustomServiceRolePolicy` policy now grants additional permissions to list and restore Secrets Manager secrets. For more information, see Amazon RDS updates to AWS managed policies.	March 6, 2025
Amazon RDS for Db2 supports the db.m7i and db.r7i instance classes	You can now use the db.m7i and db.r7i instance classes for RDS for Db2. For more information, see DB instance class types.	February 27, 2025

Amazon RDS for Oracle supports Spatial Patch Bundles (SPBs)	RDS for Oracle now publishes Spatial Patch Bundles (SPB) engine versions for Oracle Database 19c. An SPB engine version contains Release Update (RU) patches plus patches specific to Oracle Spatial. Typically, an SPB is released 2-3 weeks after its corresponding RU. You can upgrade your DB instance from a standard RU to an SPB on the same DB engine version or higher. For more information, see Oracle minor version upgrades.	February 25, 2025
Amazon RDS supports MariaDB 11.4.5, 10.11.11, 10.6.21, and 10.5.28	You can now create Amazon RDS DB instances running MariaDB version 11.4.5, 10.11.11, 10.6.21, and 10.5.28. For more informati on, see MariaDB on Amazon RDS versions.	February 24, 2025
Amazon RDS supports CloudWatch Database Insights	You can now use Database Insights to monitor Amazon RDS databases. For more information, see Monitorin g Amazon RDS databases with CloudWatch Database Insights.	February 20, 2025

Amazon RDS supports MySQL 8.4.4 and MySQL 8.0.41	You can now create Amazon RDS DB instances running MySQL 8.4.4 and 8.0.41. For more information, see MySQL on Amazon RDS versions.	February 19, 2025
Amazon RDS Extended Support version 5.7.44-RDS.20250103 for RDS for MySQL	The RDS Extended Support version 5.7.44-RDS.20250103 is now available for RDS for MySQL. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	February 13, 2025
Amazon RDS is available in the Mexico (Central) Region	Amazon RDS is now available in the Mexico (Central) Region. For more information, see Regions and Availability Zones.	January 14, 2025
Amazon RDS supports MariaDB 11.7 in the Database Preview environment	MariaDB 11.7 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MariaDB version 11.7 in the Database Preview environment.	January 9, 2025
Amazon RDS is available in the Asia Pacific (Thailand) Region	Amazon RDS is now available in the Asia Pacific (Thailand) Region. For more information, see Regions and Availability Zones.	January 7, 2025

Amazon RDS supports MariaDB 11.4.4, 10.11.10, 10.6.20, and 10.5.27	You can now create Amazon RDS DB instances running MariaDB version 11.4.4, 10.11.10, 10.6.20, and 10.5.27. For more informati on, see MariaDB on Amazon RDS versions.	December 20, 2024
Amazon RDS for Db2 supports multiple databases	You can add up to 50 Db2 databases to an RDS for Db2 DB instance. For more information, see Multiple databases on an Amazon RDS for Db2 DB instance.	December 20, 2024
Amazon RDS supports MySQL 9.1 in the Database Preview environment	MySQL 9.1 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 9.1 in the Database Preview environment.	December 19, 2024
Amazon RDS supports the db.m8g and db.r8g instance classes	You can now use the db.m8g and db.r8g instance classes for RDS for MySQL, RDS for PostgreSQL, and RDS for MariaDB. For more informati on, see DB instance class types.	November 21, 2024
Amazon RDS supports MySQL 8.4	You can now create Amazon RDS DB instances running MySQL version 8.4. For more information, see MySQL on Amazon RDS versions.	November 21, 2024

Amazon Relational Database Service supports auto-migrating EC2 databases	You can use the RDS console to migrate an EC2 database to Amazon RDS. Amazon RDS uses AWS Database Migration Service (AWS DMS) to migrate your source EC2 database. AWS DMS allows you to migrate relational databases into your AWS Cloud. For more information, see Auto migrating EC2 databases to Amazon Relational Database Service using AWS Database Migration Service.	November 20, 2024
Amazon RDS supports MariaDB version 11.8.5	RDS for MariaDB now supports MariaDB version 11.8.5. For more information, see MariaDB on Amazon RDS versions.	November 20, 2024
Amazon RDS blue/green deployments support scaling storage	You can now use Amazon RDS blue/green deployments to adjust storage settings in the green environment in order to optimize resource allocation. For more information, see Specifying changes when creating a blue/green deployment.	November 20, 2024

Amazon RDS blue/green deployments add storage initialization	Amazon RDS now supports storage initialization in green environments for blue/green deployments, improving volume performance from first use without affecting availability. For more information, see Lazy loading and storage initialization for blue/green deployments.	November 20, 2024
Amazon RDS for Oracle supports the db.m7i and db.r7i instance classes for BYOL	You can use the db.m7i and db.r7i instance classes for RDS for Oracle on the BYOL licensing model. All editions of Oracle Database are supported. For more information, see DB instance class types and Supported RDS for Oracle DB instance classes.	November 18, 2024
Amazon RDS Proxy supports RDS for PostgreSQL 17	You can now create proxies using RDS Proxy for RDS for PostgreSQL 17 DB instances. For more information, see Using Amazon RDS Proxy.	November 15, 2024
Amazon RDS supports MySQL 8.0.40	You can now create Amazon RDS DB instances running MySQL version 8.0.40. For more information, see MySQL on Amazon RDS versions.	November 13, 2024

Amazon RDS supports backup replication for Africa (Cape Town) and Asia Pacific (Hyderabad)	Backup replication is now available for databases in the Africa (Cape Town) and Asia Pacific (Hyderabad) Regions. For more information about available Regions, see Replicating automated backups to another AWS Region.	November 1, 2024
Amazon RDS supports the db.m7i and db.r7i instance classes	You can now use the db.m7i and db.r7i instance classes for RDS for MySQL, RDS for PostgreSQL, and RDS for MariaDB. For more information, see DB instance class types.	October 29, 2024
Amazon RDS for Oracle supports Oracle APEX version 24.1.v1	You can use Oracle APEX 24.1.v1 with Oracle Database 19c and higher. For more information, see Oracle Application Express.	October 22, 2024
Multi-AZ DB clusters support IAM database authentication	You can now authenticate to your Multi-AZ DB cluster using AWS Identity and Access Management (IAM) database authentication. For more information, see Settings for creating Multi-AZ DB clusters.	October 17, 2024
Amazon RDS supports MariaDB 11.4	You can now create Amazon RDS DB instances running MariaDB version 11.4. For more information, see MariaDB on Amazon RDS versions.	October 15, 2024

Amazon RDS supports Console-to-Code	You can now use Console-to-Code to generate code from actions that you perform in the RDS console. The generated code can help you write code to automate your use of other AWS services. For more information, see Use Console-to-Code to generate code for your Amazon RDS console actions.	October 3, 2024
Amazon RDS for Oracle supports Oracle Management Agent 13.5.0.0.v2	RDS supports Oracle Management Agent 13.5.0.0.v2, which requires Oracle Management Service (OMS) 13.5.0.23. For more information, see Requirements for Management Agent.	September 25, 2024
Zero-ETL integrations with Amazon Redshift generally available	Zero-ETL integrations make transactional data availabl e in Amazon Redshift within seconds of it being written to an RDS for MySQL DB instance. The feature is now generally available. For more information, see Working with Amazon RDS zero-ETL integrations with Amazon Redshift.	September 12, 2024

Amazon RDS supports MariaDB 10.11.9, 10.6.19, and 10.5.26	You can now create Amazon RDS DB instances running MariaDB version 10.11.9, 10.6.19, and 10.5.26. For more information, see Requirements for Management Agent.	September 4, 2024
Amazon RDS for Oracle supports the OEM, OEMAGENT, and OLS options for the CDB architecture	You can now use Oracle Enterprise Manager and Oracle Label Security with RDS for Oracle CDB instances . For more information, see Oracle Enterprise Manager and Oracle Label Security.	September 4, 2024
RDS Custom for SQL Server is available in additional Regions	RDS Custom for SQL Server is now available in the US West (N. California) Region, Asia Pacific (Osaka) Region, and Europe (Paris) Region. For more information, see Supported Regions and DB engines for RDS Custom .	August 29, 2024
Amazon RDS Extended Support version 5.7.44-RDS.20240808 for RDS for MySQL	The RDS Extended Support version 5.7.44-RDS.20240808 is now available for RDS for MySQL. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	August 29, 2024

Amazon RDS is available in the Asia Pacific (Malaysia) Region	Amazon RDS is now available in the Asia Pacific (Malaysia) Region. For more information, see Regions and Availability Zones.	August 22, 2024
Amazon RDS supports MySQL 8.0.39	You can now create Amazon RDS DB instances running MySQL version 8.0.39. For more information, see MySQL on Amazon RDS versions.	August 12, 2024
Update to existing policy	Amazon RDS removed `sns:Publish` permission from the `AmazonRDSPreviewServiceRolePolicy` of the `AWSServiceRoleForRDSPreview` service-linked role. For more information, see Amazon RDS updates to AWS managed policies.	August 7, 2024
Update to existing policy	Amazon RDS removed `sns:Publish` permission from the `AmazonRDSBetaServiceRolePolicy` of the `AWSServiceRoleForRDSBeta` service-linked role. For more information, see Amazon RDS updates to AWS managed policies.	August 7, 2024

Amazon RDS supports MySQL 8.4 in the Database Preview environment	MySQL 8.4 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 8.4 in the Database Preview environment.	August 1, 2024
Update to IAM service-linked role permissions	The `AmazonRDSCustomServiceRolePolicy` policy now grants additional permissions to communicate with Amazon RDS services in another AWS Region and copy EC2 images. For more information, see Amazon RDS updates to AWS managed policies.	July 18, 2024
Amazon RDS supports MariaDB 11.4 in the Database Preview environment	MariaDB 11.4 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MariaDB version 11.4 in the Database Preview environment.	July 18, 2024
AWS ODBC Driver for MySQL generally available	The Amazon Web Services (AWS) ODBC Driver for MySQL is a client driver designed for the high availability of RDS for MySQL. For more information, see Connecting to RDS for MySQL with the Amazon Web Services (AWS) ODBC Driver for MySQL.	July 18, 2024

Update to existing policy	Amazon RDS removed `sns:Publish` permission from the `AmazonRDSServiceRolePolicy` of the `AWSServiceRoleForRDS` service-linked role. For more information, see AWS managed policy: Amazon RDSServiceRolePolicy.	July 2, 2024
AWS Marketplace private offer for Db2	AWS Marketplace now supports private offers for a Db2 license through AWS Marketplace for Amazon RDS for Db2. For more information, see Obtaining a private offer.	July 1, 2024
Export Multi-AZ DB cluster snapshot data to Amazon S3	You can now export Multi-AZ DB cluster snapshot data to Amazon S3. For more information, see Exporting DB snapshot data to Amazon S3.	June 27, 2024
Amazon RDS for Oracle supports preconfigured r6i memory optimized instance classes	The db.r6i Oracle DB instance classes are optimized for workloads that require additional memory, storage, and I/O per vCPU. For example, db.r6i.8xlarge.tpc2.mem4x has multithreading turned on and provides 4 times as much memory as db.r6i.8xlarge. For more information, see RDS for Oracle instance classes.	June 21, 2024

Amazon RDS Extended Support version 5.7.44-RDS.20240529 for RDS for MySQL	The RDS Extended Support version 5.7.44-RDS.20240529 is now available for RDS for MySQL. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	June 20, 2024
Amazon RDS supports MySQL 8.0.37	You can now create Amazon RDS DB instances running MySQL version 8.0.37. For more information, see MySQL on Amazon RDS versions.	June 18, 2024
Amazon RDS supports MariaDB 10.11.8, 10.6.18, 10.5.25, and 10.4.34	You can now create Amazon RDS DB instances running MariaDB version 10.11.8, 10.6.18, 10.5.25, and 10.4.34. For more information, see MariaDB on Amazon RDS versions.	June 14, 2024
Amazon RDS is ending support for the db.m4, db.r4, and db.t2 DB instance classes	For the RDS for MariaDB, RDS for MySQL, and RDS for PostgreSQL DB engines, you can no longer create DB instances that use the db.m4, db.r4, and db.t2 instance classes. RDS is automatically upgrading existing DB instances that use these classes to a newer generation. For more information, see DB instance class types.	June 4, 2024

Multi-AZ DB clusters are available in additional AWS Regions	You can create Multi-AZ DB clusters in more AWS Regions. For a table that shows all supported Regions, see Supported Regions and DB engines for Multi-AZ DB clusters in Amazon RDS.	May 29, 2024
AWS Python Driver generally available	The Amazon Web Services (AWS) Python Driver is designed as an advanced Python wrapper. This wrapper is complementary to and extends the functionality of the open-source Psycopg driver. For more informati on, see Connecting to DB instances with the AWS drivers.	May 23, 2024
RDS Proxy is available in more Regions	RDS Proxy is now available in the Asia Pacific (Hyderaba d), Asia Pacific (Melbourne), Middle East (UAE), Israel (Tel Aviv), Canada West (Calgary) , and Europe (Zurich) regions. For more information about RDS Proxy, see Using Amazon RDS Proxy.	May 21, 2024
Db2 license through AWS Marketplace	With Db2 license through AWS Marketplace, you can now pay an hourly rate to subscribe to Db2 licenses for Amazon RDS for Db2. For more information, Amazon RDS for Db2 licensing options.	May 21, 2024

Amazon RDS supports fine-grained access for Performance Insights	You can now allow or deny access to individual dimensions in Performance Insights. This fine-grained access can be used for `GetResourceMetrics`, `DescribeDimensionKeys`, and `GetDimensionKeyDetails` actions. For more information, see Granting fine-grained access for Performance Insights.	May 21, 2024
Amazon RDS Extended Support versions for RDS for MySQL	You can view all releases of RDS Extended Support for RDS for MySQL versions. For more information, see Amazon RDS Extended Support versions for RDS for MySQL.	May 16, 2024
Amazon RDS supports MySQL 8.3 in the Database Preview environment	MySQL 8.3 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 8.3 in the Database Preview environment.	April 30, 2024
Amazon RDS for Db2 supports time zones	RDS for Db2 now supports setting local time zones for new RDS for Db2 DB instances. For more information, see Local time zones for Amazon RDS for Db2 DB instances.	April 25, 2024

Update to IAM service-linked role permissions	The `AmazonRDSCustomServiceRolePolicy` policy now grants additional permissions to associate a service role as an instance profile to a RDS Custom instance. For more information, see Amazon RDS updates to AWS managed policies.	April 19, 2024
Amazon RDS for Oracle supports Oracle Data Guard switchover in all AWS Regions	You can now use Oracle Data Guard switchover in all supported Regions. For more information, see Overview of Oracle Data Guard switchover.	April 16, 2024
RDS Custom for Oracle supports Oracle Standard Edition 2	You can now create DB instances using Standard Edition 2 on Oracle Database 12c Release 1 (12.1), 12c Release 2 (12.2), 18c, and 19c. You can create both CDBs and non-CDBs. For more information, see Edition and licensing support for RDS Custom for Oracle.	April 11, 2024
Amazon RDS for Oracle supports Oracle APEX version 23.2.v1	You can use Oracle APEX 23.2.v1 with Oracle Database 19c and higher. For more information, see Oracle Application Express.	April 11, 2024

Update to RDS Custom service-linked role permissions	The `AmazonRDSCustomServiceRolePolicy` now grants additional permissions to allow RDS Custom for SQL Server to get EC2 instance type information and modify DB host instance type. For more information, Updates to AWS managed policies.	April 8, 2024
Amazon RDS Custom for Oracle supports the db.x2iezn DB instance class	You can now use the db.x2iezn instance class for RDS Custom for Oracle DB instances. For more information, see DB instance class support for RDS Custom for Oracle.	March 26, 2024
Amazon RDS supports the db.c6gd instance classes for Multi-AZ DB clusters	You can now use the db.c6gd instance classes for Multi-AZ DB cluster deployments. For more information, see Instance class availability for Multi-AZ DB clusters.	March 21, 2024

Amazon RDS Extended Support	Creating or restoring an RDS for MySQL 5.7 or RDS for PostgreSQL 11 database now automatically enrolls that database into Amazon RDS Extended Support so your existing applications continue to work as they are. You can opt out of RDS Extended Support to avoid charges after the RDS end of standard support date for your database engine. For more information, see Using Amazon RDS Extended Support.	March 21, 2024
RDS for Db2 integration with AWS License Manager	RDS for Db2 is now integrated with AWS License Manager. If you use the bring your own license (BYOL) model, the AWS License Manager integration aids in monitoring your Db2 license usage within your organization. For more information, see Integrating with AWS License Manager.	March 20, 2024

CA certificate rotation for Multi-AZ DB clusters	You can now rotate the CA certificates for your Multi-AZ DB clusters. Consider using one of the new CA certificates rds-ca-rsa2048-g1, rds-ca-rsa4096-g1, or rds-ca-ecc384-g1. For more information, see Rotating your SSL/TLS certificate.	March 6, 2024
Amazon RDS supports io2 Block Express storage	You can now create RDS DB instances that use the io2 Block Express storage type. For more information, see io2 Block Express storage.	March 6, 2024
RDS Custom for SQL Server supports the db.r5b and db.x2iedn DB instance classes	You can now use the db.r5b and db.x2iedn instance classes for RDS Custom for SQL Server DB instances. For more information, see DB instance class support for RDS Custom for SQL Server.	March 4, 2024
RDS Custom for Oracle is available in the Middle East (UAE) Region	You can create RDS Custom for Oracle DB instances in the Middle East (UAE) Region. For a table that shows all supported AWS Regions, see Supported Regions and DB engines for RDS Custom for Oracle.	March 4, 2024

New AWS managed policy	Amazon RDS added a new managed policy named `AmazonRDSCustomInstanceProfileRolePolicy` to allow RDS Custom to perform automation actions and database management tasks through an EC2 instance profile. For more information, see Amazon RDS updates to AWS managed policies.	February 27, 2024
Amazon RDS supports MariaDB 10.11.7, 10.6.17, 10.5.24, and 10.4.33	You can now create Amazon RDS DB instances running MariaDB version 10.11.7, 10.6.17, 10.5.24, and 10.4.33. For more information, see MariaDB on Amazon RDS versions.	February 26, 2024
Amazon RDS Multi-AZ DB clusters support the Amazon EBS gp3 storage volume	Multi-AZ DB clusters now support gp3 SSD-based EBS volumes. For more information, see gp3 storage.	February 26, 2024
Amazon RDS support for AWS Secrets Manager in the Israel (Tel Aviv) Region	Amazon RDS supports Secrets Manager in the Israel (Tel Aviv) Region. For more information, see Password management with Amazon RDS and AWS Secrets Manager.	February 21, 2024

Amazon RDS for Db2 supports audit logging	RDS for Db2 now supports database-level audit logging. When you enable audit logging for an RDS for Db2 database, Amazon RDS records the database activity and stores the audit logs in Amazon S3. For more information, see Db2 audit logging.	February 15, 2024
Amazon RDS Extended Support	Amazon RDS now automatically enables Amazon RDS Extended Support when RDS for MySQL and RDS for PostgreSQL major engine versions in your DB instances and Multi-AZ DB clusters reach the RDS end of standard support date. For more information, see Using Amazon RDS Extended Support.	February 15, 2024
Amazon RDS supports MySQL 8.0.36	You can now create Amazon RDS DB instances running MySQL version 8.0.36. For more information, see MySQL on Amazon RDS versions.	February 12, 2024

Amazon RDS supports EBCDIC collation for RDS for Db2	You can now create Db2 databases that use EBCDIC collation sequences to sort content in the databases. For more information, see EBCDIC collation for Db2 databases on Amazon RDS.	January 29, 2024
Update to default CA Certificate	The default CA certificate is set to `rds-ca-rsa2048-g1`. For more information, see Using SSL/TLS to encrypt a connection to a DB instance.	January 26, 2024
Amazon RDS for PostgreSQL supports two new crates for PL/Rust, croaring-rs and num-bigint	You can use two new crates in Amazon RDS for PostgreSQL. For more information, see Using crates with PL/Rust.	January 24, 2024
Amazon RDS for PostgreSQL supports TLS version 1.3	You can use Transport Layer Security (TLS) version 1.3 in RDS for PostgreSQL. For more information, see Using SSL with a PostgreSQL DB instance.	January 24, 2024
RDS Custom for SQL Server supports Microsoft SQL Server 2022	You can now create RDS Custom for SQL Server DB instances that use SQL Server 2022. For more information, see Working with RDS Custom for SQL Server.	January 22, 2024

Update to AWS managed policy permissions	The `AmazonRDSServiceRolePolicy` of the `AWSServiceRoleForRDS` service-linked role has new statement IDs. For more information, see Amazon RDS updates to AWS managed policies.	January 19, 2024
RDS Custom for Oracle supports the Europe (Paris) Region	You can create RDS Custom for Oracle DB instances in the Europe (Paris) Region. For more information, see Supported Regions and DB engines for RDS Custom for Oracle.	January 18, 2024
Amazon RDS for MySQL supports multi-source replication	You can now use multi-source replication on RDS for MySQL DB instances. For more information, see Configuring multi-source replication on RDS for MySQL.	January 16, 2024
Amazon RDS supports MySQL 8.2 in the Database Preview environment	MySQL 8.2 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 8.2 in the Database Preview environment.	January 11, 2024
RDS Proxy is available in the Europe (Spain) Region	RDS Proxy is now available in the Europe (Spain) region. For more information about RDS Proxy, see Using Amazon RDS Proxy.	January 8, 2024

Amazon RDS is available in the Canada West (Calgary) Region	Amazon RDS is now available in the Canada West (Calgary) Region. For more information, see Regions and Availability Zones.	December 20, 2023
Amazon RDS for Db2 supports 5,000 local users	You can now add up to 5,000 local users to an authorization list. For more information, see rdsadmin.add_user.	December 20, 2023
Amazon RDS supports viewing and responding to recommendations	Amazon RDS recommend ations now includes threshold based proactive and machine learning based reactive recommendations for RDS for PostgreSQL. For more information, see Recommend ations from Amazon RDS.	December 19, 2023
Amazon RDS supports MariaDB 10.11.6, 10.6.16, 10.5.23, and 10.4.32	You can now create Amazon RDS DB instances running MariaDB version 10.11.6, 10.6.16, 10.5.23, and 10.4.32. For more information, see MariaDB on Amazon RDS versions.	December 12, 2023

Amazon RDS introduces zero-ETL integrations with Amazon Redshift (preview)	Zero-ETL integrations provide a fully managed solution for making transactional data available in Amazon Redshift within seconds of it being written to an RDS for MySQL DB instance. For more information, see Working with Amazon RDS zero-ETL integrations with Amazon Redshift (preview).	November 28, 2023
Amazon RDS supports IBM Db2 database engines	You can now run IBM Db2 database engines in Amazon RDS. For more information, see Amazon RDS for Db2.	November 27, 2023
RDS for PostgreSQL supports major version upgrades to PostgreSQL 16.1 and minor version upgrades to 15.5, 14.10, 13.13, 12.17, and 11.22	With RDS for PostgreSQL, you can now upgrade the DB engine to major version 16.1 and minor version upgrades to 15.5, 14.10, 13.13, 12.17, and 11.22. For more information, see Upgrading the PostgreSQL DB engine for Amazon RDS.	November 17, 2023
RDS Custom for Oracle supports option groups	You can create or modify an option group and associate it with an RDS Custom for Oracle DB instance. The `Timezone` option is now supported. For more information, see Working with option groups in RDS Custom for Oracle.	November 17, 2023

Amazon RDS for MySQL supports the Group Replication plugin	You can now set up an active-active cluster with RDS for MySQL version 8.0.35 or higher DB instances by using the Group Replication plugin developed and maintained by the MySQL community. For more information, see Configuring active-active clusters for RDS for MySQL.	November 17, 2023
Amazon RDS Proxy supports RDS for PostgreSQL 16.1	You can now create proxies using RDS Proxy for RDS for PostgreSQL 16.1 DB instances. For more information, see Using Amazon RDS Proxy.	November 17, 2023
RDS Custom for SQL Server supports Microsoft SQL Server 2019 Developer edition	You can create RDS Custom for SQL Server DB instances that use SQL Server 2019 Developer edition. For more information, see Bring Your Own Media with RDS Custom for SQL Server.	November 16, 2023

Minor version upgrades of Multi-AZ DB clusters with minimal downtime	When you perform a minor version upgrade of a Multi-AZ DB cluster, Amazon RDS now upgrades the reader DB instances before the writer instance, thereby significantly reducing downtime. You can further reduce downtime to one second or less by using RDS Proxy. For more information, see Upgrading the engine version of a Multi-AZ DB cluster.	November 16, 2023
RDS for SQL Server supports Microsoft SQL Server 2022	You can now create RDS DB instances that use SQL Server 2022. For more information, see Microsoft SQL Server versions on Amazon RDS.	November 15, 2023
RDS for MySQL supports upgrading snapshots from version 5.7 to 8.0	You can now upgrade the engine version of an RDS for MySQL snapshot from version 5.7 to version 8.0. You can do so by using the AWS Management Console, or the `ModifyDBSnapshot` operation of the RDS API or AWS CLI. For more information, see Upgrading a MySQL DB snapshot engine version.	November 15, 2023

RDS Custom for SQL Server supports point in time recovery of 1,0000 databases	You can now make up to 1,000 databases eligible for full backup and point in time recovery on your RDS Custom for SQL Server DB instance. For more information, see Restoring an RDS Custom for SQL Server instance to a point in time.	November 15, 2023
RDS Custom for SQL Server supports a using a Service Master Key	RDS Custom for SQL Server now supports using a Service Master Key (SMK). An SMK allows you to encrypt objects such as credentials, and use SQL Server features like TDE and column-encryption. For more information, see Using a Service Master Key with RDS Custom for SQL Server.	November 13, 2023
Amazon RDS supports MySQL 8.1 in the Database Preview environment	MySQL 8.1 is now available in the Database Preview environment in the US East (Ohio) AWS Region. For more information, see MySQL version 8.1 in the Database Preview environment.	November 10, 2023
RDS supports MySQL 8.0.35 and MySQL 5.7.44	You can now create Amazon RDS DB instances running MySQL version 8.0.35 and 5.7.44. For more information, see MySQL on Amazon RDS versions.	November 9, 2023

RDS Proxy supports Multi-AZ DB clusters	RDS Proxy now supports connecting to Multi-AZ DB clusters. For more informati on, see Working with Amazon RDS Proxy endpoints.	November 9, 2023
RDS Custom for Oracle is available in the AWS GovCloud (US) Regions	Amazon RDS is now available in the AWS GovCloud (US) Regions. For more informati on, see Supported Regions and DB engines for RDS Custom for Oracle.	November 9, 2023
Amazon RDS Optimized Writes supports the db.m5 DB instance class	Amazon RDS Optimized Writes now supports the db.m5 DB instance class. For more information, see Improving write performance with Amazon RDS Optimized Writes for MariaDB and Improving write performance with Amazon RDS Optimized Writes for MySQL.	November 9, 2023

Amazon RDS for Oracle supports the multi-tenant configuration of the CDB architecture	With the RDS for Oracle multi-tenant feature, RDS delivers a fully managed Oracle multitenant architecture and experience for your Oracle databases. You can use RDS APIs to create multiple PDBs, called *tenant databases*, in a CDB. RDS offers the multi-tenant configuration of the CDB architecture as an alternative to the legacy single-tenant configuration. For more information, see Multi-tenant configuration of the CDB architecture.	November 8, 2023
Amazon RDS exports Performance Insights metrics to Amazon CloudWatch	Performance Insights lets you export the preconfigured or custom metrics dashboards to Amazon CloudWatch. The exported metrics dashboards are available to view in the CloudWatch console. You can also export a selected Performance Insights metric widget and view the metrics data in the CloudWatch console. For more information, see Exporting Performance Insights metrics to CloudWatch.	November 8, 2023

Amazon RDS Custom for Oracle allows you to upgrade the operating system on a DB instance	You can now upgrade the database or operating system (OS) for an RDS Custom for Oracle DB instance using the CLI command `modify-db-instance` . For more information, see Upgrading a DB instance for Amazon RDS Custom for Oracle.	November 7, 2023
RDS Proxy supports Extended Protocol for RDS for PostgreSQL	You can now execute extended query protocols on an RDS for PostgreSQL DB instance. For more information, see Using Amazon RDS Proxy.	November 6, 2023
RDS for PostgreSQL support for RDS Blue/Green Deployments	You can now create a blue/green deployment from an RDS for PostgreSQL DB instance. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.	October 26, 2023
Update to AWS managed policies	The `AmazonRDSPerformanceInsightsReadOnly` and `AmazonRDSPerformanceInsightsFullAccess` managed policies now includes `Sid` (statement ID) as an identifier in the policy statement. For more information, see Amazon RDS updates to AWS managed policies.	October 23, 2023

RDS Custom for Oracle supports the Europe (Milan) Region	For more information, see Supported Regions and DB engines for RDS Custom for Oracle.	October 23, 2023
Enable RDS Optimized Writes on existing databases	You can now enable RDS Optimized Writes on an existing DB instance even if it was created with an engine version, DB instance class, or file system config uration that doesn't support the feature. For more information, see Enabling RDS Optimized Writes on an existing database for RDS for MySQL, and Enabling RDS Optimized Writes on an existing database for RDS for MariaDB.	October 19, 2023
Amazon RDS supports using a dedicated log volume (DLV).	You can now use a dedicated log volume (DLV) with RDS for MariaDB, RDS for MySQL, and RDS for PostgreSQL. DLVs are ideal for databases with large allocated storage, high I/O per second (IOPS) requirements, or latency-s ensitive workloads. For more information, see Using a dedicated log volume (DLV).	October 17, 2023

Amazon RDS for PostgreSQL, MySQL, and MariaDB support new DB instance classes	You can create Amazon RDS DB instances running PostgreSQL, MySQL, and MariaDB that use the db.m6.in, db.m6idn, db.r6.in, and db.r6.idn DB instance classes. For more information, see Supported DB engines for all available DB instance classes.	October 12, 2023
Amazon RDS for PostgreSQL supports pgactive	The pgactive extension is available in Amazon RDS for PostgreSQL. For more information, see Using PostgreSQL extensions with Amazon RDS for PostgreSQL.	October 9, 2023
RDS Custom for Oracle is available in the Asia Pacific (Jakarta) Region	You can create RDS Custom for Oracle DB instances in the Asia Pacific (Jakarta) Region. For more information, see Supported Regions and DB engines for RDS Custom for Oracle.	October 5, 2023
RDS Custom for SQL Server supports new server-level collations	RDS Custom for SQL Server now supports a wide range of server collations, both in traditional and UTF-8 encoding, for the SQL_Latin, Japanese, German, and Arabic locales. For more information, see Collation and character support for RDS Custom for SQL Server DB instances.	September 26, 2023

Update to AWS managed policy permissions	The `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role has new permissions that allow RDS Custom to create, modify, and delete EventBridge Managed Rules. For more information, see Amazon RDS updates to AWS managed policies.	September 20, 2023
Amazon RDS publishes Performance Insights counter metrics to Amazon CloudWatch	The **DB_PERF_INSIGHTS** metric math function in the CloudWatch console allows you to query Amazon RDS for Performance Insights counter metrics. For more information, see Creating CloudWatch alarms to monitor Amazon RDS.	September 20, 2023
Performance Insights supports digest-level statistics for SQL Server	When you use Performance Insights, you can view SQL statistics both at the statement and digest level for Amazon RDS for SQL Server. For more information, see Analyzing running queries in SQL Server.	September 18, 2023

Amazon RDS for PostgreSQL, MySQL, and MariaDB support the db.m6.id and db.r6.id DB instance class types	You can now create Amazon RDS DB instances running PostgreSQL, MySQL, and MariaDB that use the memory-optimized db.m6.id and db.r6.id DB instance class types. These types offer local NVMe-based SSD storage. For more information, see Supported DB engines for all available DB instance classes.	September 11, 2023
Major version upgrade support for RDS for PostgreSQL Multi-AZ DB clusters	You can now perform major version upgrades of your RDS for PostgreSQL Multi-AZ DB clusters. For more informati on, see Upgrading the engine version of a Multi-AZ DB cluster.	September 7, 2023
Amazon RDS supports MariaDB 10.11.5, 10.6.15, 10.5.22, and 10.4.31	You can now create Amazon RDS DB instances running MariaDB version 10.11.5, 10.6.15, 10.5.22, and 10.4.31. For more information, see MariaDB on Amazon RDS versions.	September 7, 2023

Amazon RDS Extended Support	Amazon RDS announces the upcoming ability to continue running RDS for MySQL and RDS for PostgreSQL major engine versions in your DB instances past the RDS end of standard support date. For more information, see Using Amazon RDS Extended Support.	September 1, 2023
RDS Custom supports starting and stopping an RDS Custom for SQL Server DB instance	RDS Custom now supports starting and stopping an RDS Custom for SQL Server DB instance. For more information, see Starting and stopping an RDS Custom for SQL Server DB instance.	August 31, 2023
Amazon RDS Optimized Writes supports the db.r5 DB instance class	Amazon RDS Optimized Writes now supports the db.r5 DB instance class. For more information, see Improving write performance with Amazon RDS Optimized Writes for MariaDB and Improving write performance with Amazon RDS Optimized Writes for MySQL.	August 31, 2023

Amazon RDS for Oracle supports timezone file autoupgrade for CDBs	With the `TIMEZONE_FILE_AUTOUPGRADE` option, you can upgrade the current time zone file to the latest version on your RDS for Oracle container database (CDB). For more informati on, see Oracle time zone file autoupgrade.	August 29, 2023
Amazon RDS Optimized Writes supports the db.m6g and db.m6i DB instance classes	Amazon RDS Optimized Writes now supports the db.m6g and db.m6i DB instance classes. For more information, see Improving write performance with Amazon RDS Optimized Writes for MariaDB and Improving write performance with Amazon RDS Optimized Writes for MySQL.	August 28, 2023
Amazon RDS supports MariaDB 10.11	You can now create Amazon RDS DB instances running MariaDB version 10.11. For more information, see MariaDB on Amazon RDS versions.	August 21, 2023

Update to AWS managed policy permissions	The `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role has new permissions that allow RDS Custom to create network interfaces. For more information, see Amazon RDS updates to AWS managed policies.	August 18, 2023
Update to AWS managed policy permissions	The `AmazonRDSFullAccess` managed policy has new permissions that allows you to generate, view, and delete the performance analysis report for a time period. For more information, see Amazon RDS updates to AWS managed policies.	August 17, 2023
Update to AWS managed policy permissions	The addition of new permissions to `AmazonRDSPerformanceInsightsReadOnly` managed policy and addition of new managed policy `AmazonRDSPerformanceInsightsFullAccess` allows you generate a DB load analysis report for a time period. For more information, see Amazon RDS updates to AWS managed policies.	August 16, 2023

Amazon RDS supports performance analysis for a period of time	Performance Insights allows you to create and view performance analysis reports for a specific period of time. The report provides the insights identified and recommendations to resolve the performance issues. For more information, see Analyzing DB load for a period of time.	August 16, 2023
Amazon RDS Custom for Oracle supports the db.r5b and db.x2iedn DB instance classes	You can now use the db.r5b and db.x2iedn instance classes for RDS Custom for Oracle DB instances. For more information, see DB instance class support for RDS Custom for Oracle.	August 16, 2023
Amazon RDS Custom for Oracle supports the db.m6i, db.r6i, and db.t3 DB instance classes	You can now use the db.m6i, db.r6i, and db.t3 instance classes for RDS Custom for Oracle DB instances. For more information, see DB instance class support for RDS Custom for Oracle.	August 15, 2023
Amazon RDS for PostgreSQL now supports PostgreSQL version 16 Beta 3 in the database preview environment	PostgreSQL version 16 Beta 3 is now available in the database preview envir onment in the US East (Ohio) AWS Region. For more information, see Working with the database preview environment.	August 11, 2023

Amazon RDS supports MySQL 8.0.34 and 5.7.43	You can now create Amazon RDS DB instances running MySQL version 8.0.34 and 5.7.43. For more information, see MySQL on Amazon RDS versions.	August 9, 2023
RDS for SQL Server supports OS metrics view for the standby replica	You can now view OS metrics for standby replica for RDS for SQL Server. For more information, see Viewing OS metrics in the RDS console.	August 3, 2023
RDS for Oracle supports Oracle Data Guard for CDBs	RDS for Oracle supports Data Guard read replicas for Oracle Database 19c and 21c container databases (CDBs). You can create, manage, and promote read replicas in a CDB, just as you can in a non-CDB, using the existing RDS APIs. For more information, see Multitenant read replicas.	August 1, 2023
Amazon RDS is available in the Israel (Tel Aviv) Region	Amazon RDS is now available in the Israel (Tel Aviv) Region. For more information, see Regions and Availability Zones.	August 1, 2023
Amazon RDS supports Oracle APEX version 23.1.v1	You can use Oracle APEX 23.1.v1 with Oracle Database 19c and higher. For more information, see Oracle Application Express.	July 26, 2023

Amazon RDS Custom for Oracle supports a nondefault Oracle SID	When you create an RDS Custom for Oracle DB instance using Oracle Database 19c, you can specify a nondefault Oracle system identifier (Oracle SID). This value is also the name of the CDB. For more informati on, see Multitenant architect ure considerations .	July 21, 2023
RDS for SQL Server supports Self Managed Active Directory	You can now use Self Managed Active Directory to directly join your RDS for SQL Server DB instances to your Microsoft Active Directory (AD) domains. Self-mana ged AD domains can be on-premises or in the cloud. For more information, see Working with Self Managed Active Directory.	July 7, 2023
PostgreSQL logical replicati on support for Multi-AZ DB clusters	You can now use PostgreSQ L logical replication with your Multi-AZ DB cluster to replicate and synchronize individual tables rather than an entire database instance. For more information, see Setting up PostgreSQL logical replication with Multi-AZ DB clusters for Amazon RDS.	July 6, 2023

Amazon RDS for PostgreSQL now supports PostgreSQL version 16 Beta 2 in the database preview environment	PostgreSQL version 16 Beta 2 is now available in the database preview environment in the US East (Ohio) AWS Region. For more information, see Working with the database preview environment.	July 6, 2023
Update to AWS managed policy permissions	The `AmazonRDSCustomServiceRolePolicy` of the `AWSServiceRoleForRDSCustom` service-linked role has new permissions that allow RDS Custom for Oracle to use snapshots. For more information, see Amazon RDS updates to AWS managed policies.	June 23, 2023
RDS supports MariaDB 10.6.14, 10.5.21, and 10.4.30	You can now create Amazon RDS DB instances running MariaDB version 10.6.14, 10.5.21, and 10.4.30. For more information, see MariaDB on Amazon RDS versions.	June 22, 2023
RDS supports MySQL 8.0.33 and 5.7.42	You can now create Amazon RDS DB instances running MySQL version 8.0.33 and 5.7.42. For more information, see MySQL on Amazon RDS versions.	June 15, 2023

RDS supports MariaDB 10.6.13, 10.5.20, 10.4.29, and 10.3.39	You can now create Amazon RDS DB instances running MariaDB version 10.6.13, 10.5.20, 10.4.29, and 10.3.39. For more information, see MariaDB on Amazon RDS versions.	June 15, 2023
RDS for Oracle supports transportable tablespaces	You can migrate data from an on-premises Oracle database into an RDS for Oracle DB instance using transportable tablespaces. This technique doesn't require additional licensing and is the migration technique with the least downtime. For more information, see Migrating using Oracle transportable tablespaces.	June 15, 2023
Amazon RDS supports RDS Proxy with RDS for MariaDB version 10.6	You can now create an RDS Proxy with an RDS for MariaDB version 10.6 database. For more information about RDS Proxy, see Using Amazon RDS Proxy.	June 15, 2023
RDS Custom for SQL Server supports Bring Your Own Media (BYOM)	You can now create a Custom Engine Version (CEV) using your own SQL Server media. For more information, see Bring Your Own Media with RDS Custom for SQL Server.	June 8, 2023

RDS for Oracle can convert an Oracle Database 19c non-CDB to a CDB	If your DB instance runs Oracle Database 19c with the April 2021 or higher RU, you can convert a non-CDB to a CDB (container database) . After you convert the architecture, you can upgrade your 19c CDB to a 21c CDB. This step is necessary bec ause you can't upgrade your database and convert the architecture using a single command. For more informati on, see Converting an RDS for Oracle non-CDB into a CDB.	May 31, 2023
Multi-AZ DB clusters available in the China Regions	Multi-AZ DB clusters are now available in the AWS Regions China (Beijing) and China (Ningxia). For more informati on, see Supported Regions and DB engines for Multi-AZ DB clusters in Amazon RDS.	May 30, 2023
Amazon RDS Optimized Reads support for Multi-AZ DB clusters	Amazon RDS Optimized Reads now supports Multi-AZ DB clusters. For more information, see Improving query performance for RDS for MySQL with Amazon RDS Optimized Reads and Improving query performance for RDS for PostgreSQL with Amazon RDS Optimized Re ads.	May 30, 2023

RDS Custom for Oracle supports the Asia Pacific (Jakarta) Region	For more information, see Supported Regions and DB engines for RDS Custom for Oracle.	May 29, 2023
Create a DB instance read replica with a source RDS for PostgreSQL Multi-AZ DB cluster	You can now create a DB instance read replica with an RDS for PostgreSQL Multi-AZ DB cluster as the source. Previously, only RDS for MySQL was supported. For more information, see Creating a DB instance read replica from a Multi-AZ DB cluster.	May 24, 2023
Amazon RDS provides combined Performance Insights and CloudWatch metrics in the Performance Insights dashboard	Amazon RDS now provides a consolidated view of Performance Insights and CloudWatch metrics in the Performance Insights dash board. For more informati on, see Viewing combined metrics with the Performance Insights dashboard.	May 24, 2023

Amazon RDS Optimized Reads available in the China Regions	Amazon RDS Optimized Reads is now available in the AWS Regions China (Beijing) and China (Ningxia). For more information, see Improving query performance for RDS for MariaDB with Amazon RDS Optimized Reads and Improving query performance for RDS for MySQL with Amazon RDS Optimized Reads.	April 24, 2023
Amazon RDS support for AWS Secrets Manager in the China Regions	Amazon RDS supports Secrets Manager in the China (Beijing) and China (Ningxia) Regions. For more information, see Password management with Amazon RDS and AWS Secrets Manager.	April 20, 2023
RDS Custom for Oracle supports reuse of AMI IDs for new CEVs	When you create a custom engine version (CEV), RDS Custom for Oracle defaults to the most recent Amazon Machine Image (AMI) available. Now you can specify an AMI ID that was used in a previous CEV. For more information, see Creating a CEV.	April 19, 2023

Amazon RDS supports publishing events with tags to topic subscribers	Amazon RDS event notifications sent to Amazon Simple Notification Service (Amazon SNS) or Amazon EventBridge now contain event tags in the message body. These tags provide the resource data that was affected by the service event. For more information, see Amazon RDS event notification tags and attributes.	April 17, 2023
Purchase reserved instances for a Multi-AZ DB cluster	You can now purchase reserved DB instances for a Multi-AZ DB cluster. For more information, see Reserved DB instances for a Multi-AZ DB cluster.	April 12, 2023
Amazon RDS supports the db.m7g and db.r7g instance classes	You can now use the db.m7g and db.r7g instance classes for RDS for MySQL, RDS for MariaDB, and RDS for PostgreSQL DB instances. For more information, see Supported DB engines for DB instance classes.	April 12, 2023

Update to Amazon RDS Custom service-linked role permissions	The `AmazonRDSCustomServiceRolePolicy` now grants additional permissions to allow RDS Custom for SQL Server to use Amazon SQS and create snapshots. For more information, see Updates to AWS managed policies.	April 6, 2023
Migrate to an RDS for MySQL Multi-AZ DB cluster using a read replica	You can now use a read replica to migrate an RDS for MySQL Single-AZ deployment or Multi-AZ DB instance deployment to an RDS for MySQL Multi-AZ DB cluster deployment with reduced downtime. For more information, see Migrating to a Multi-AZ DB cluster using a read replica.	April 6, 2023
Create a DB instance read replica from a Multi-AZ DB cluster	You can now create a DB instance read replica from a Multi-AZ DB cluster in order to scale beyond the compute capacity of the source cluster. For more information, see Creating a DB instance read replica from a Multi-AZ DB cluster.	April 6, 2023

Amazon RDS Custom for SQL Server supports Multi-AZ	You can create a Multi-AZ deployment with RDS Custom for SQL Server. For more information, see Managing a Multi-AZ deployment for RDS Custom for SQL Server.	April 6, 2023
Update to AWS managed policy permissions	The `AmazonRDSFullAccess` and `AmazonRDSReadOnlyAccess` policies now grants additional permissions to allow the display of Amazon DevOps Guru findings in the RDS console. For more information, see Amazon RDS updates to AWS managed policies.	March 30, 2023
Amazon RDS supports Oracle APEX version 22.2.v1	You can use Oracle APEX 22.2.v1 with all supported versions of Oracle Database. For more information, see Oracle Application Express.	March 30, 2023

Amazon DevOps Guru available for RDS for PostgreSQL	RDS for PostgreSQL alerts you to recent anomalies detected by Amazon DevOps Guru. The database details page of the console alerts you to current and anomalies that occurred in the past 24 hours. DevOps Guru publishes proactive insights with recommendations to help address issues in your RDS for PostgreSQL databases before they are predicted to happen. For more information, see How DevOps Guru for RDS works.	March 30, 2023
RDS Custom supports the Amazon EBS gp3 storage volume	RDS Custom for Oracle and RDS Custom for SQL Server both support the io1, gp2, and gp3 SSD-based EBS volumes. For more informati on, see General requirements for RDS Custom for Oracle and General requirements for RDS Custom for SQL Server.	March 29, 2023

Update to AWS managed policy permissions	The `AmazonRDSFullAccess` and `AmazonRDSReadOnlyAccess` policies now grants additional permissions to Amazon CloudWatch. For more information, see Amazon RDS updates to AWS managed policies.	March 16, 2023
RDS Proxy is available in the China Regions	RDS Proxy is now available in the China (Beijing) and China (Ningxia) regions. For more information about RDS Proxy, see Using Amazon RDS Proxy.	March 15, 2023
RDS Proxy is available in the Asia Pacific (Jakarta) Region	RDS Proxy is now available in the Asia Pacific (Jakarta) Region. For more information about RDS Proxy, see Using Amazon RDS Proxy.	March 8, 2023
Amazon RDS Optimized Writes improves the performance of write transactions for RDS for MariaDB	You can improve the performance of write transactions for RDS for MariaDB DB instances with Amazon RDS Optimized Writes. For more information, see Improving write performance with Amazon RDS Optimized Writes for MariaDB.	March 7, 2023

Amazon RDS for PostgreSQL versions 15.2	New features in Amazon RDS for PostgreSQL 15.2 include the SQL standard "MERGE" command for conditional SQL queries, performance improvements for both in-memory and disk-based sorting, and support for two-phase commit and row/column filtering for logical replication.	February 27, 2023
RDS Custom for Oracle is available in the Canada (Central) Region and South America (São Paulo) Regions	For a table that shows all supported AWS Regions, see Supported Regions and DB engines for RDS Custom for Oracle.	February 22, 2023
Amazon RDS supports cross-Region automated backups for RDS for MariaDB and RDS for MySQL	You can now replicate DB snapshots and transaction logs between AWS Regions for RDS for MariaDB and RDS for MySQL DB instances. For more information, see Replicating automated backups to another AWS Region.	February 22, 2023

Amazon RDS for Oracle supports advance notice of automatic minor version upgrades	RDS notifies you in advance of the date when a new minor version of the RDS for Oracle engine will become available . RDS begins scheduling automatic minor version upgrades of your RDS for Oracle DB instances on the availability date. For more information, see Before an automatic minor version upgrade is scheduled.	February 21, 2023
Amazon RDS for SQL Server supports Database Activity Streams	You can now monitor a SQL Server DB instance using Database Activity Streams. A SQL Server database instance has the server audit which is managed by Amazon RDS. You can define the policies to record server events in the server audit specification. You can create a database audit specification and define the policies to record databa se events. The stream of activity is collected and transmitted to Amazon Kinesis. From Kinesis, you can monitor the activity stream for further analysis. For more information, see Monitoring Amazon RDS with Database Activity Streams.	February 15, 2023

RDS supports MySQL 8.0.32 and 5.7.41	You can now create Amazon RDS DB instances running MySQL version 8.0.32 and 5.7.41. For more information, see MySQL on Amazon RDS versions.	February 7, 2023
Amazon RDS for Oracle supports new cipher suites for SSL	If you run Oracle Database 19c or 21c, you can specify six new cipher suites in the SSL option for RDS for Oracle. These suites support FIPS and are FedRAMP-complaint. For more information, see Oracle Secure Sockets Layer.	February 3, 2023
Amazon RDS for Oracle supports new cipher suites for Oracle Enterprise Manager	You can use four new FedRAMP-compliant cipher suites for the OEM option. For more information, see Oracle Management Agent for Enterprise Manager Cloud Control.	February 3, 2023
RDS for Oracle supports Database Activity Streams in the Asia Pacific (Hyderabad), Europe (Spain), and Middle East (UAE) Regions	For more information, see Supported Regions and DB engines for database activity streams in Amazon RDS.	January 27, 2023

Migrate to an RDS for PostgreSQL Multi-AZ DB cluster using a read replica	By using a read replica, you can migrate an RDS for PostgreSQL Single-AZ deployment or Multi-AZ DB instance deployment to an RDS for PostgreSQL Multi-AZ DB cluster deployment with reduced downtime. For more information, see Migrating to a Multi-AZ DB cluster using a read replica.	January 23, 2023
Amazon RDS is available in the Asia Pacific (Melbourne) Region	Amazon RDS is now available in the Asia Pacific (Melbourne) Region. For more information, see Regions and Availability Zones.	January 23, 2023
RDS for MariaDB supports enforcing SSL/TLS connections	RDS for MariaDB now supports enforcing SSL/TLS connections by setting the `require_secure_transport` parameter to `ON`. For more information, see Requiring SSL/TLS for all connections to a MariaDB DB instance.	January 19, 2023
Amazon RDS Optimized Reads improves query performance for RDS for MariaDB	You can achieve faster query processing for RDS for MariaDB DB instances with Amazon RDS Optimized Reads. For more information, see Improving query performance for RDS for MariaDB with Amazon RDS Optimized Reads.	January 11, 2023

Restore a Multi-AZ DB cluster snapshot to a DB instance	You can now restore a Multi-AZ DB cluster snapshot to a Single-AZ deployment or Multi-AZ DB instance deployment. For more information, see Restoring from a Multi-AZ DB cluster snapshot to a DB instance.	January 10, 2023
Specify certificate authority (CA) during DB instance creation	You can now specify which CA to use for a DB instance's server certificate during DB instance creation. For more information, see Certificate authorities.	January 5, 2023
RDS Custom for SQL Server supports custom engine versions	A custom engine version (CEV) for RDS Custom for SQL Server is an Amazon Machine Image (AMI) with Microsoft SQL Server pre-installed. You choose an Amazon EC2 Windows AMI to use as a base image and can install other software on the operating system (OS). You can customize the configuration of the OS and SQL Server to meet your enterprise needs. For more information, see Working with custom engine versions for RDS Custom for SQL Server.	December 28, 2022

Use Amazon RDS Blue/Green Deployments available in additional AWS Regions	The Blue/Green Deployments feature is now available in the China (Beijing) and China (Ningxia) Regions. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.	December 22, 2022
Update to IAM service-linked role permissions	The AmazonRDSServiceRolePolicy policy now grants additional permissions to AWS Secrets Manager. For more information, see Amazon RDS updates to AWS managed policies.	December 22, 2022
Amazon RDS supports renaming a Multi-AZ DB cluster	You can now rename a Multi-AZ DB cluster. For more information, see Renaming a Multi-AZ DB cluster.	December 22, 2022
Amazon RDS integrates with AWS Secrets Manager for password management	Amazon RDS can manage the master user password for a DB instance or Multi-AZ DB cluster in Secrets Manager. For more information, see Password management with Amazon RDS and AWS Secrets Manager.	December 22, 2022

Amazon RDS Optimized Writes supports the db.r6g and db.r6gd DB instance classes	Amazon RDS Optimized Writes now supports the db.r6g and db.r6gd DB instance classes. For more information, see Improving write performance with Amazon RDS Optimized Writes.	December 22, 2022
Amazon RDS Custom for Oracle supports new AWS Regions	You can create RDS Custom for Oracle DB instances in the Asia Pacific (Seoul) and Asia Pacific (Osaka) Regions. For more information, see Supported Regions and DB engines for RDS Custom for Oracle.	December 21, 2022
Amazon RDS on AWS Outposts supports read replicas	You can now create a read replica from an RDS on Outposts MySQL or PostgreSQL DB instance. For more information, see Creating read replicas for Amazon RDS on AWS Outposts.	December 19, 2022
RDS Custom for Oracle supports modifying the DB instance class	You can now change the instance class of your RDS Custom for Oracle DB instance. For more information, see Modifying your RDS Custom for Oracle DB instance.	December 16, 2022

RDS for MySQL and RDS for PostgreSQL support db.x2iedn DB instance classes	You can now use the db.x2iedn DB instance classes for RDS for MySQL and RDS for PostgreSQL DB instances . For more information, see Supported DB engines for DB instance classes.	December 14, 2022
Amazon RDS Optimized Writes supports db.x2iedn DB instance classes	Amazon RDS Optimized Writes now supports db.x2iedn DB instance classes. For more information, see Improving write performance with Amazon RDS Optimized Writes.	December 14, 2022
Amazon RDS supports copying DB option groups when copying DB snapshots	You can now copy an option group across AWS accounts as part of a snapshot copy request on RDS for Oracle databases. For more information, see Option group considerations.	December 13, 2022
Amazon RDS supports RDS Proxy with RDS for PostgreSQL version 14	You can now create an RDS Proxy with an RDS for PostgreSQL version 14 database. For more information about RDS Proxy, see Using Amazon RDS Proxy.	December 13, 2022

Amazon RDS for Oracle supports db.x2idn, db.x2iedn, and db.x2iezn instance classes	You can now use the db.x2idn, db.x2iedn, and db.x2iezn instance classes for Amazon RDS for Oracle DB instances . For more information, see Supported DB engines for DB instance classes and Supported RDS for Oracle instance classes.	December 12, 2022
RDS for PostgreSQL DB instances support Trusted Language Extensions for PostgreSQL	Trusted Language Extension s for PostgreSQL is an open source development kit that allows you to build high performance PostgreSQ L extensions and safely run them on your RDS for PostgreSQL DB instance. For more information, see Working with Trusted Language Extensions for PostgreSQL.	November 30, 2022

Use Amazon RDS Blue/Green Deployments for database updates	You can make changes to a DB instance in a staging environment and test the changes without affecting your production DB instance. When you are ready, you can promote the staging environment to be the new production environment, with minimal downtime. For more information, see Using Amazon RDS Blue/Green Deployments for database updates.	November 27, 2022
Amazon RDS Optimized Writes improves the performance of write transactions for RDS for MySQL	You can improve the performance of write transactions for RDS for MySQL DB instances with Amazon RDS Optimized Writes. For more information, see Improving write performance with Amazon RDS Optimized Writes for MySQL.	November 27, 2022
Amazon RDS Optimized Reads improves query performance for RDS for MySQL	You can achieve faster query processing for RDS for MySQL DB instances with Amazon RDS Optimized Reads. For more information, see Improving query performance with Amazon RDS Optimized Reads.	November 27, 2022

Amazon RDS is available in the Asia Pacific (Hyderabad) Region	Amazon RDS is now available in the Asia Pacific (Hyderabad) Region. For more information, see Regions and Availability Zones.	November 22, 2022
RDS supports MariaDB 10.6.11, 10.5.18, 10.4.27, and 10.3.37	You can now create Amazon RDS DB instances running MariaDB versions 10.6.11, 10.5.18, 10.4.27, and 10.3.37. For more informati on, see MariaDB on Amazon RDS versions.	November 18, 2022
RDS Custom for Oracle supports setting nondefault installation parameters in a custom engine version (CEV)	When you create a CEV, you can set nondefault values for the Oracle base, Oracle home, UNIX user name and ID, and UNIX group name and ID. In this way, you gain more control over the database installation on your RDS Custom for Oracle DB instance. For more informati on, see Preparing the CEV manifest.	November 18, 2022
Amazon RDS supports Oracle APEX version 22.1.v1	You can use Oracle APEX 22.1.v1 with all supported versions of Oracle Database. For more information, see Oracle Application Express.	November 18, 2022

RDS for SQL Server supports cross-Region read replicas	You can now create a cross-Region read replica to enhance disaster recovery capability, reduce application read latency, and offload read workloads from the primary DB instance. For more information, see Creating a read replica in a different AWS Region.	November 16, 2022
Amazon RDS is available in the Europe (Spain) Region	Amazon RDS is now available in the Europe (Spain) Region. For more information, see Regions and Availability Zones.	November 16, 2022
RDS for SQL Server supports linked servers for Oracle database	You can now create a linked server to access external Oracle databases to read data and execute SQL commands. For more information, see Linked Servers with Oracle OLEDB with RDS for SQL Server.	November 15, 2022

RDS Custom for Oracle supports Oracle Multitenant	You can create an RDS Custom for Oracle DB instance as a container database (CDB). After creation, the CDB contains the CDB root, PDB seed, and one PDB. You can add more PDBs manually using Oracle SQL. For more information, see Overview of Amazon RDS Custom for Oracle architecture.	November 15, 2022
Amazon RDS for Oracle supports Amazon EFS integration	If you add the `EFS_INTEGRATION` option to your option group, you can transfer files between your RDS for Oracle DB instance and an Amazon EFS file system. For more information, see Amazon EFS.	November 15, 2022
RDS supports MySQL 8.0.31 and 5.7.40	You can now create Amazon RDS DB instances running MySQL version 8.0.31 and 5.7.40. For more information, see MySQL on Amazon RDS versions.	November 10, 2022
Amazon RDS is available in the Europe (Zurich) Region	Amazon RDS is now available in the Europe (Zurich) Region. For more information, see Regions and Availability Zones.	November 9, 2022

Access to transaction log backups is now available for RDS for SQL Server	You can now view and copy database transaction log backups to an Amazon S3 bucket. For more informati on, see Access to transaction log backups.	November 7, 2022
Multi-AZ DB clusters supported in additional AWS Regions	Multi-AZ DB clusters are now available in additional AWS Regions. For more information, see Supported Regions and DB engines for Multi-AZ DB clusters in Amazon RDS.	November 4, 2022
Amazon RDS supports gp3 storage	You can now create Amazon RDS DB instances that use Amazon EBS General Purpose SSD (gp3) storage volumes, which let you customize storage performan ce independently of storage capacity. For more informati on, see General Purpose SSD storage.	November 4, 2022
Amazon RDS supports a new event for operating system updates	Amazon RDS now supports a new DB instance event, RDS-EVENT-0230, under the event category of security patching. This new event alerts you when an operating system update is available for your DB instance. For more information, see Monitorin g Amazon RDS events and Working with operating system updates.	October 28, 2022

Amazon RDS for Oracle supports preconfigured r5b memory optimized instance classes	The db.r5b Oracle DB instance classes are optimized for workloads that require additional memory, storage, and I/O per vCPU. For example, db.r5b.4xlarge.tpc2.mem2x has multithreading turned on and provides twice as much memory as db.r5b.4xlarge. For more information, see RDS for Oracle instance classes.	October 27, 2022
Amazon RDS supports 15 read replicas for RDS for MariaDB, MySQL, and PostgreSQL DB instances	You can now create up to 15 read replicas for RDS for MariaDB, MySQL, and PostgreSQL DB instances. For more information about read replicas, see Working with read replicas.	October 20, 2022
Amazon RDS for PostgreSQL now supports PostgreSQL version 15 RC 3 in the database preview environment	PostgreSQL version 15 Beta 3 is now available in the database preview environment in the US East (Ohio) AWS Region. For more information, see Working with the database preview environment.	October 18, 2022

Amazon RDS supports automatically setting up connectivity between an RDS database and an EC2 instance	You can use the AWS Management Console to set up connectivity between an existing RDS DB instance or Multi-AZ DB cluster and an EC2 instance. For more information, see Connecting an EC2 instance and an RDS database automatically.	October 14, 2022
AWS JDBC Driver for PostgreSQL generally available	The AWS JDBC Driver for PostgreSQL is a client driver designed for RDS for PostgreSQL. The AWS JDBC Driver for PostgreSQL is now generally available. For more information, see Connecting with the AWS JDBC Driver for PostgreSQL.	October 6, 2022
Amazon RDS for Oracle supports Oracle APEX version 21.2.v1	Oracle APEX 21.2 includes patch 33420059. For more information, see Oracle APEX version requirements.	October 3, 2022
RDS supports MySQL 5.7.39	You can now create Amazon RDS DB instances running MySQL versions 5.7.39. For more information, see MySQL on Amazon RDS versions.	September 29, 2022

RDS supports MariaDB 10.6.10	You can now create Amazon RDS DB instances running MariaDB versions 10.6.10. For more information, see MariaDB on Amazon RDS versions.	September 29, 2022
RDS Proxy supports RDS for SQL Server	You can now create an RDS Proxy for an RDS DB instance that runs Microsoft SQL Server version 2014 or higher. For more information about RDS Proxy, see Using Amazon RDS Proxy.	September 19, 2022
RDS supports MariaDB 10.5.17, 10.4.26, and 10.3.36	You can now create Amazon RDS DB instances running MariaDB versions 10.5.17, 10.4.26, and 10.3.36. For more information, see MariaDB on Amazon RDS versions.	September 15, 2022

Amazon RDS for Oracle supports local instance storage for temporary data	You can now launch Amazon RDS for Oracle on Amazon EC2 db.r5d and db.m5d instance types with the temporary tablespace and Database Smart Flash Cache (the flash cache) configured to use an instance store. By storing temporary data locally, you can achieve lower read and write latencies when compared to standard storage based on Amazon EBS. For more information, see Storing temporary Oracle data in the instance store.	September 14, 2022
Performance Insights shows the top 25 SQL queries	In the Performance Insights dashboard, the **Top SQL** tab shows the 25 SQL queries that are contributing the most to DB load. For more information, see Overview of the Top SQL tab.	September 13, 2022
RDS supports MySQL 8.0.30	You can now create Amazon RDS DB instances running MySQL version 8.0.30. For more information, see MySQL on Amazon RDS versions.	September 9, 2022
Amazon RDS is available in the Middle East (UAE) Region	Amazon RDS is now available in the Middle East (UAE) Region. For more information, see Regions and Availability Zones.	August 30, 2022

Amazon RDS for SQL Server supports SSRS Email subscriptions	You can now use the SQL Server Reporting Services (SSRS) Email extension to send reports to users and subscribe to reports on the report server. For more information, see Support for SQL Server Reporting Services in RDS for SQL Server.	August 26, 2022
RDS for Oracle supports read replica backups	You can turn on automatic backups and create manual snapshots of RDS for Oracle replicas. For more information, see Working with RDS for Oracle replica backups.	August 23, 2022
RDS for Oracle supports Oracle Data Guard switchover	A switchover is a role reversal between a primary database and a mounted or open Oracle replica. During a switchover, the original primary database transitions to a standby role, while the original standby database transitions to the primary role. For more information, see Performing an Oracle Data Guard switchover.	August 23, 2022

Amazon RDS supports automatically setting up connectivity with an EC2 instance	When you create a DB instance or Multi-AZ DB cluster, you can use the AWS Management Console to s et up connectivity between an Amazon Elastic Compute Cloud instance and the new DB instance or DB c luster. For more informati on, see Configure automatic network connectivity with an EC2 instance for a new DB instance and Configure automatic network connectiv ity with an EC2 instance for a new DB cluster.	August 22, 2022
RDS Custom for Oracle supports promotion of Oracle replicas	If you use RDS Custom for Oracle, you can promote your managed Oracle replicas by using the `promote-read-replica` CLI command. Also, you can delete your primary DB instance, which causes RDS Custom for Oracle to promote your managed Oracle replicas to standalone instances. For more informati on, see Working with Oracle replicas for RDS Custom for Oracle.	August 5, 2022

RDS for MySQL supports enforcing SSL/TLS connections	RDS for MySQL now supports enforcing SSL/TLS connections by setting the `require_secure_transport` parameter to ON. For more information, see Requiring an SSL/TLS connection to a MySQL DB instance.	August 1, 2022
Amazon RDS has deprecated support for Oracle Database 12c Release 1 (12.1.0.2)	Support for version 12.1.0.2 is deprecated for both the BYOL and LI licensing models. On August 1, 2022, RDS for Oracle begins automatic upgrades of 12c Release 1 (12.1.0.2) DB instances and restored 12.1.0.2 snapshots to Oracle Database 19c. For more information, see the end of support timeline on AWS re:Post.	August 1, 2022
RDS Proxy supports RDS for MariaDB	You can now create an RDS Proxy for an RDS DB instance that runs MariaDB version 10.2, 10.3, 10.4, or 10.5. The MariaDB support is included under the MySQL engine family. For more information about RDS Proxy, see Using Amazon RDS Proxy.	July 26, 2022

RDS for MariaDB supports the db.r5b DB instance classes	You can now create RDS for MariaDB DB instances that use the db.r5b DB instance classes. For more information, Supported DB engines for DB instance classes.	July 25, 2022
RDS for Oracle supports modifying database activity streams	If you use RDS for Oracle, you can change the audit policy state of a database acti vity stream to either locked (default) or unlocked. Instead of stopping an activity stream, you can unlock its policy state, customize your audit policy, and then relock the policy state. For more information, see Modifying a database activity stream.	July 22, 2022
Performance Insights supports the Asia Pacific (Jakarta) Region	Formerly, you couldn't use Performance Insights in the Asia Pacific (Jakarta) Reg ion. This restriction has been removed. For more informati on, see Supported Regions and DB engines for Performan ce Insights in Amazon RDS.	July 21, 2022

Microsoft SQL Server 2012 has reached its end of support on Amazon RDS	Microsoft SQL Server 2012 has reached its end of support, coinciding with the Microsoft plan to end extended support for this version on July 12, 2022. Any existing Microsoft SQL Server 2012 instances are to be automatically upgraded to the latest minor version of Microsoft SQL Server 2014 starting on June 1, 2022. For more information, see Microsoft SQL Server 2012 support on Amazon RDS.	July 12, 2022
RDS supports MariaDB 10.6.8, 10.5.16, 10.4.25, 10.3.35, and 10.2.44	You can now create Amazon RDS DB instances running MariaDB versions 10.6.8, 10.5.16, 10.4.25, 10.3.35, and 10.2.44. For more information, see Supported MariaDB versions on Amazon RDS.	July 8, 2022

RDS Performance Insights supports additional retention periods	Previously, Performance Insights offered only two retention periods: 7 days (default) or 2 years (731 days). Now, if you need to retain your performance d ata for longer than 7 days, you can specify from 1–24 months. For more informat ion, see Pricing and data retention for Performance Insights.	July 1, 2022
RDS Custom supports the Asia Pacific (Mumbai) and Europe (London) Regions	You can create RDS Custom for Oracle and RDS Custom for SQL Server DB instances in two new AWS Regions: Asia Pacific (Mumbai) and Europe (London). For more informati on, see AWS Region support for RDS Custom for Oracle and AWS Region support for RDS Custom for SQL Server.	June 21, 2022
RDS Custom for Oracle supports Oracle Database 18c and 12c Release 2 (12.2)	You can now create a CEV for RDS Custom for Oracle using installation files for Oracle Database 18c and 12c Release 2 (12.2). You can use this these CEVs to create an RDS Custom for Oracle DB instance. For more informati on, see Working with custom engine versions for Amazon RDS Custom for Oracle.	June 21, 2022

Multi-AZ DB clusters support the db.m5d and db.r5d DB instance classes	You can now create Multi-AZ DB clusters that use the db.m5d and db.r5d DB instance classes. For more information, see Multi-AZ DB cluster deployments and DB instance class types.	June 21, 2022
Multi-AZ DB clusters available in additional AWS Regions	You can now create Multi-AZ DB clusters in the following Regions: Europe (Frankfurt) and Europe (Stockholm). For more information, see Multi-AZ DB cluster deployments.	June 21, 2022
RDS for Microsoft SQL Server supports migration of databases that use Transparent Data Encryption (TDE)	RDS for SQL Server now supports migrating Microsoft SQL Server databases with TDE turned on, using native backup and restore. For more information, see Support for Transparent Data Encryption in SQL Server.	June 14, 2022
Amazon RDS supports publishing events to encrypted Amazon SNS topics	Amazon RDS can now publish events to Amazon Simple Notification Service (Amazon SNS) topics that have server-side encryption (SSE) enabled, for additional protection of events that carry sensitive data. For more information, see Subscribing to Amazon RDS event notification.	June 1, 2022

RDS supports MySQL 5.7.38	You can now create Amazon RDS DB instances running MySQL version 5.7.38. For more information, see MySQL on Amazon RDS versions.	May 31, 2022
RDS for PostgreSQL supports cascading read replicas	You can now use cascading read replicas with RDS for PostgreSQL version 14.1 and higher releases. For more information, see Working with PostgreSQL read replicas in Amazon RDS.	May 4, 2022
Amazon RDS on AWS Outposts supports scale storage and autoscaling operations	You can now change the storage sizes of DB instances on your Outpost and use storage autoscaling. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	May 2, 2022
Multi-AZ DB clusters available in additional AWS Regions	You can now create Multi-AZ DB clusters in the following Regions: Asia Pacific (Singapore) and Asia Pacific (Sydney). For more informati on, see Multi-AZ DB cluster deployments.	April 29, 2022

Amazon RDS supports dual-stack mode	DB instances can now run in dual-stack mode. In dual-stack mode, resources can communicate with the DB instance over IPv4, IPv6, or both. For more informati on, see Amazon RDS IP addressing.	April 29, 2022
Amazon RDS publishes usage metrics to Amazon CloudWatc h	The AWS/Usage namespace in Amazon CloudWatch includes account-level usage metrics for your Amazon RDS service quotas. For more information, see Amazon CloudWatch usage metrics for Amazon RDS.	April 28, 2022
Amazon RDS for MySQL supports the db.m6i and db.r6i DB instance classes	You can now use the db.m6i and db.r6i DB instance classes for Amazon RDS DB instances running MySQL. For more information, see Supported DB engines for DB instance classes.	April 28, 2022
Amazon RDS for PostgreSQ L supports the db.m6i and db.r6i DB instance classes	You can now use the db.m6i and db.r6i DB instance classes for Amazon RDS DB instances running Postg reSQL. For more information, see Supported DB engines for DB instance classes.	April 27, 2022

Amazon RDS for MariaDB supports the db.m6i and db.r6i DB instance classes	You can now use the db.m6i and db.r6i DB instance classes for Amazon RDS DB instances running MariaDB. For more information, see Supported DB engines for DB instance classes.	April 26, 2022
Amazon RDS on AWS Outposts supports Multi-AZ deployments	You can now create a standby DB instance on a different Outpost. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	April 19, 2022
Amazon RDS for Oracle supports the db.m6i and db.r6i instance classes	If you run Oracle Database 19c, you can use the db.m6i and db.r6i instance classes. The db.m6i classes are general-purpose instance classes that are well suited for a broad range of workloads. For more information, see RDS for Oracle instance classes.	April 8, 2022
Amazon RDS for SQL Server supports SQL Server Agent job replication	When you turn on this feature, SQL Server Agent jobs created, modified, or deleted on the primary host are automatically synchronized to the secondary host in a Multi-AZ configuration. For more information, see Using SQL Server Agent.	April 7, 2022

Amazon RDS supports RDS Proxy with RDS for PostgreSQL version 13	You can now create an RDS Proxy with an RDS for PostgreSQL version 13 database. For more information about RDS Proxy, see Using Amazon RDS Proxy.	April 4, 2022
Amazon RDS plans to deprecate Oracle Database 12c	Oracle Database 12c is on a deprecation path. Oracle Corporation will no longer provide patches for Oracle Database 12c releases after the end-of-support dates. Amazon RDS plans to begin automatically upgrading Oracle Database 12c DB instances to Oracle Database 19c.	March 22, 2022
Amazon RDS for PostgreSQL Release Notes	There is now a separate guide for the Amazon RDS for PostgreSQL release notes. For more information, see *Amazon RDS for PostgreSQL Release Notes*.	March 22, 2022
Amazon RDS for Oracle Release Notes	There is now a separate guide for the Amazon RDS for Oracle release notes. For more information, see *Amazon RDS for Oracle Release Notes*.	March 22, 2022

Multi-AZ DB clusters available in additional AWS Regions	You can now create Multi-AZ DB clusters in the following Regions: US East (Ohio) and Asia Pacific (Tokyo). For more information, see Multi-AZ DB cluster deployments.	March 15, 2022
Amazon RDS for PostgreSQL versions 14.2, 13.6, 12.10, 11.15, and 10.20	RDS for PostgreSQL now supports versions 14.2, 13.6, 12.10, 11.15, and 10.20. Version 14.2 and 13.6 add support for two new foreign data wrappers. The `mysql_fdw` extension lets PostgreSQL work with data stored in MySQL, MariaDB, and Aurora MySQL databases. The `tds_fdw` extension lets PostgreSQL work with data stored in SQL Server databases. For more information, see Supported PostgreSQL database versions.	March 12, 2022
RDS supports MySQL 5.7.37	You can now create Amazon RDS DB instances running MySQL version 5.7.37. For more information, see MySQL on Amazon RDS versions.	March 11, 2022

Amazon RDS for SQL Server supports new DB instance classes	You can now create Amazon RDS DB instances running Microsoft SQL Server that use the db.m6i and db.r6i DB instance classes. For more information, see DB instance class support for Microsoft SQL Server.	March 9, 2022
Amazon RDS for Oracle supports Oracle Database 21c	You can now create Amazon RDS DB instances running Oracle Database 21c (21.0.0.0). This is the first Oracle Database release that supports only the multitena nt (CDB) architecture. For more information, see Oracle Database 21c with Amazon RDS.	March 7, 2022
RDS supports MariaDB 10.6.7, 10.5.15, 10.4.24, 10.3.34, and 10.2.43	You can now create Amazon RDS DB instances running MariaDB versions 10.6.7, 10.5.15, 10.4.24, 10.3.34, and 10.2.43. For more information, see MariaDB on Amazon RDS versions.	March 3, 2022
AWS JDBC Driver for MySQL generally available	The AWS JDBC Driver for MySQL is a client driver designed for RDS for MySQL. The AWS JDBC Driver for MySQL is now generally available. For more informat ion, see Connecting with the Amazon Web Services JDBC Driver for MySQL.	March 2, 2022

Multi-AZ DB clusters generally available	A Multi-AZ DB cluster deployment is a high availability deployment mode of Amazon RDS with two readable standby DB instances. Multi-AZ DB clusters are now generally available. For more information, see Multi-AZ DB cluster deployments.	March 1, 2022
RDS supports MySQL 8.0.28	You can now create Amazon RDS DB instances running MySQL version 8.0.28. For more information, see MySQL on Amazon RDS versions.	February 28, 2022
Amazon RDS for Oracle supports new settings for native network encryption (NNE)	To control whether clients can connect with non-secure encryption and checksumming methods, set `SQLNET.ALLOW_WEAK_CRYPTO_CLIENTS` and `SQLNET.ALLOW_WEAK_CRYPTO` in the NNE option. Examples of insecure methods include DES, 3DES, RC4, and MD5. For more information, see NNE option settings.	February 25, 2022

Amazon RDS for SQL Server supports Always On Availability Groups for Microsoft SQL Server 2017 Standard Edition	When you create a DB instance using the Multi-AZ configuration on SQL Server 2017 Standard Edition 14.00.3401.7 and higher versions, RDS automatically uses Availability Groups. For more information, see Multi-AZ deployments for Microsoft SQL Server.	February 18, 2022
RDS for Oracle supports Database Activity Streams in the Asia Pacific (Jakarta) Region	For more information, see Support for AWS Regions for database activity streams.	February 16, 2022
Amazon RDS Custom for Oracle support for Oracle Database 12.1	You can now create custom engine versions for RDS Custom for Oracle that use Oracle Database 12.1 Enterprise Edition. For more information, see Working with custom engine versions for Amazon RDS Custom for Oracle.	February 4, 2022
Amazon RDS for MariaDB supports a new major version	You can now create Amazon RDS DB instances running MariaDB version 10.6. For more information, see MariaDB 10.6 support on Amazon RDS.	February 3, 2022

Performance Insights supports plan capture for Oracle queries	The Performance Insights console supports a new plan dimension for top SQL. When you slice by plan, you can see which plans your top Oracle queries are using. If a query uses multiple plans, you can compare the plans side by side in the console and determine which plan is most efficient. You can also drill down to see which steps in a plan have the highest cost. For more information, see Analyzing Oracle execution plans using the Performance Insights dashboard.	January 27, 2022
Performance Insights supports new APIs	Performance Insights supports the following APIs: `GetResourceMetadata`, `ListAvailableResourceDimensions`, and `ListAvailableResourceMetrics`. For more information, see Retrieving metrics with the Performance Insights API in this manual and the *Amazon RDS Performance Insights API Reference*.	January 12, 2022

RDS Proxy supports events	RDS Proxy now generates events that you can subscribe to and view in CloudWatch Events or configure to send to Amazon EventBridge. For more information, see Working with RDS Proxy events.	January 11, 2022
Amazon RDS for SQL Server supports SSAS Multidimensional mode	RDS for SQL Server supports running SQL Server Analysis Services (SSAS) in Tabular or Multidimensional mode. For more information, see Support for SQL Server Analysis Services in RDS for SQL Server.	January 7, 2022
RDS Proxy available in additional AWS Regions	RDS Proxy is now available in the following Regions: Africa (Cape Town), Asia Pacific (Hong Kong), Asia Pacific (Osaka), Europe (Milan), Europe (Paris), Europe (Stockholm), Middle East (Bahrain), and South America (São Paulo). For more information about RDS Proxy, see Using Amazon RDS Proxy.	January 5, 2022
RDS supports MySQL 8.0.27	You can now create Amazon RDS DB instances running MySQL version 8.0.27. For more information, see MySQL on Amazon RDS versions.	December 21, 2021

Amazon RDS is available in the Asia Pacific (Jakarta) Region	Amazon RDS is now available in the Asia Pacific (Jakarta) Region. For more information, see Regions and Availability Zones.	December 13, 2021
Amazon RDS supports MariaDB 10.5.13, 10.4.22, 10.3.32, and 10.2.41	You can now create Amazon RDS DB instances running MariaDB versions 10.5.13, 10.4.22, 10.3.32, and 10.2.41. For more informati on, see MariaDB on Amazon RDS versions.	December 8, 2021
Amazon RDS Custom for SQL Server	Amazon RDS Custom is a managed database service for legacy, custom, and packaged applications that require access to the underlying operating system and database environment. With Amazon RDS Custom, you get the automation of Amazon RDS and the flexibili ty of Amazon EC2. For more information, see Working with Amazon RDS Custom.	December 1, 2021

Multi-AZ DB clusters (preview)	You can now create Multi-AZ DB clusters for RDS for MySQL and RDS for PostgreSQL. A Multi-AZ DB cluster deployment is a high availability deployment mode of Amazon RDS with two readable standby DB instances. Multi-AZ DB clusters are in preview. For more information, see Multi-AZ DB cluster deployments (preview).	November 23, 2021
Amazon RDS supports RDS Proxy with RDS for PostgreSQL version 12	You can now create an RDS Proxy with an RDS for PostgreSQL version 12 database. For more information about RDS Proxy, see Using Amazon RDS Proxy.	November 22, 2021
Amazon RDS on AWS Outposts supports local backups	You can store automated backups and manual snapshots in your AWS Region or locally on your Outpost. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	November 22, 2021
Amazon RDS support for cross-account AWS KMS keys	You can use a KMS key from a different AWS account for encryption when exporting DB snapshots to Amazon S3. For more information, see Exporting DB snapshot data to Amazon S3.	November 3, 2021

Amazon RDS on AWS Outposts supports publishing database engine logs to CloudWatch Logs	RDS on Outposts now supports publishing database engine logs to CloudWatch Logs. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	November 2, 2021
Amazon RDS Custom for Oracle	Amazon RDS Custom is a managed database service for legacy, custom, and packaged applications that require access to the underlying operating system and database environment. With Amazon RDS Custom, you get the automation of Amazon RDS and the flexibility of Amazon EC2. For more information, see Working with Amazon RDS Custom.	October 26, 2021
Support for delayed replication for RDS for MySQL version 8.0	Starting with RDS for MySQL version 8.0.26, you can configure delayed replication for RDS for MySQL version 8.0 DB instances. For more information, see Configuring delayed replication with MySQL.	October 25, 2021
Support for MySQL 8.0.26	You can now create Amazon RDS DB instances running MySQL version 8.0.26. For more information, see MySQL on Amazon RDS versions.	October 25, 2021

Support for GTID-based replication for RDS for MySQL version 8.0	Starting with RDS for MySQL version 8.0.26, you can configure GTID-based re plication for RDS for MySQL version 8.0 DB instances. For more information, see Using GTID-based replication for RDS for MySQL.	October 25, 2021
Amazon RDS supports RDS Proxy with RDS for MySQL 8.0	You can now create an RDS Proxy for an RDS for MySQL 8.0 database instance. For more information, see Using Amazon RDS Proxy.	October 21, 2021
Amazon RDS on AWS Outposts supports additional RDS for MySQL versions	RDS on Outposts now supports RDS for MySQL versions 8.0.23 and 8.0.25. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	October 20, 2021
Amazon RDS for PostgreSQ L now supports PostgreSQ L version 14 RC 1 in the database preview environme nt	PostgreSQL version 14 RC 1 is now available in the database preview environment in the US East (Ohio) AWS Region. For more information, see Working with the database preview environment.	October 19, 2021

Amazon RDS supports Performance Insights in additional AWS Regions	Performance Insights is available in the Middle East (Bahrain), Africa (Cape Town), Europe (Milan), and Asia Pacific (Osaka) Regions. For more information, see Supported Regions and DB engines for Performance Insights in Amazon RDS.	October 5, 2021
Performance Insights supports digest-level statistics for Oracle	When you use Performance Insights, you can view SQL statistics both at the statement and digest level for Amazon RDS for Oracle. For more information, see Analyzing running queries in Oracle.	October 4, 2021
Amazon RDS on AWS Outposts supports additional RDS for PostgreSQL versions	RDS on Outposts now supports RDS for PostgreSQL versions 12.8 and 13.4. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	October 1, 2021
Amazon RDS supports Oracle APEX version 21.1.v1	You can use Oracle APEX 21.1.v1 with all supported versions of Oracle Database. For more information, see Oracle Application Express.	September 24, 2021

Amazon RDS for Oracle supports client-side encryption for NNE	When you configure NNE, you might want to avoid forcing encryption on the server side. For example, you might not want to force all client communications to use encryption because the server requires it. In this case, you can force encryption on the client side using the `SQLNET.*CLIENT` options. For more information, see Oracle native network encryption.	September 24, 2021
Amazon RDS for MySQL and RDS for PostgreSQL support new DB instance classes	You can now use the db.r5b, db.t4g, and db.x2g instance classes to create Amazon RDS DB instances running MySQL or PostgreSQL. For more information, see Supported DB engines for DB instance classes.	September 15, 2021
Amazon RDS for Microsoft SQL Server supports Java Database Connectivity (JDBC) with Microsoft Distributed Transaction Coordinator (MSDTC)	JDBC XA transactions are now supported with MSDTC for SQL Server 2017 version 14.00.3223.3 and higher, and SQL Server 2019. For more information, see Support for Microsoft Distributed Transaction Coordinator in RDS for SQL Server.	September 7, 2021

Amazon RDS supports MariaDB 10.5.12, 10.4.21, 10.3.31, and 10.2.40	You can now create Amazon RDS DB instances running MariaDB versions 10.5.12, 10.4.21, 10.3.31, and 10.2.40. For more informati on, see MariaDB on Amazon RDS versions.	September 2, 2021
Amazon RDS has ended support for Oracle Database 18c	You can create DB instances only for Oracle Database 12c and Oracle Database 19c. If you have Oracle Database 18c snapshots, upgrade them to a later release. For more information, see Upgrading an Oracle DB snapshot.	August 17, 2021
Amazon RDS for PostgreSQ L now supports PostgreSQ L version 14 beta 2 in the database preview environme nt	For more about PostgreSQ L version 14 beta 1, see PostgreSQL 14 beta 1 release notes. For more about PostgreSQL version 14 beta 2, see PostgreSQL 14 beta 2 release notes. For informati on on the Database Preview Environment, see Working with the database preview environment.	August 9, 2021
Amazon RDS supports RDS Proxy in a shared VPC	You can now create an RDS Proxy in a shared VPC. For more information about RDS Proxy, see "Managing Connections with Amazon RDS Proxy" in the Amazon RDS User Guide or the Aurora User Guide.	August 6, 2021

Amazon RDS supports MariaDB 10.2.39	You can now create Amazon RDS DB instances running MariaDB versions 10.2.39. For more information, see MariaDB on Amazon RDS versions.	August 4, 2021
Amazon RDS for Oracle adds the `TIMEZONE_FILE_AUTOUPGRADE` option	With this option, you can upgrade the current time zone file to the latest version on your Oracle DB instance. For more information, see Oracle time zone file autoupgrade.	July 30, 2021
Amazon RDS extends support for cross-Region automated backups	You can now replicate DB snapshots and transaction logs between more AWS Regions. For more information, see Replicating automated backups to another AWS Region.	July 19, 2021
Support for MySQL 5.7.34	You can now create Amazon RDS DB instances running MySQL version 5.7.34. For more information, see MySQL on Amazon RDS versions.	July 8, 2021
Amazon RDS on AWS Outposts supports additional RDS for PostgreSQL versions	RDS on Outposts now supports RDS for PostgreSQL versions 12.7 and 13.3. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	July 8, 2021

Amazon RDS for PostgreSQL supports oracle_fdw	You can now use the oracle_fdw extension to provide a foreign data wrapper for access to Oracle databases. For more informati on, see Accessing external data with the oracle_fdw extension.	July 8, 2021
Amazon RDS supports Oracle Management Agent (OMA) version 13.5	You can use Oracle Management Agent (OMA) version 13.5 with Oracle Enterprise Manager (OEM) Cloud Control 13c Release 5 and higher. Amazon RDS for Oracle installs OMA, which then communicates with your Oracle Management Service (OMS) to provide monitorin g information. If you run OMS 13.5, you can manage databases by installing OMA 13.5. For more information, see Oracle Management Agent for Enterprise Manager Cloud Control.	July 7, 2021

Amazon RDS for Oracle supports downloading logs from Amazon S3	If archived redo logs aren't on your instance but are protected by your backup retention period, you can use `rdsadmin.rdsadmin_archive_log_download` to download them from Amazon S3. RDS for Oracle saves the logs to the `/rdsdbdata/log/arch` directory on your DB instance. For more information, see Downloading archived redo logs from Amazon S3.	July 2, 2021
Amazon RDS supports MariaDB 10.4.18 and 10.5.9	You can now create Amazon RDS DB instances running MariaDB versions 10.4.18 and 10.5.9. For more information, see MariaDB on Amazon RDS versions.	June 30, 2021

Amazon RDS for Oracle supports Database Activity Streams	You can now monitor an Oracle DB instance using Database Activity Streams. An Oracle database writes audit records to the unified audit trail. When you start a database activity stream on an Oracle DB instance, Amazon Kinesis streams all activities that match the Oracle Database audit policies. For more information, see Monitoring Amazon RDS with Database Activity Streams.	June 23, 2021
Amazon RDS for Oracle introduces memory optimized instance classes	New Oracle DB instance classes are optimized for workloads that require additional memory, storage, and I/O per vCPU. For more information, see RDS for Oracle instance classes.	June 23, 2021
Support for MySQL 8.0.25	You can now create Amazon RDS DB instances running MySQL version 8.0.25. For more information, see MySQL on Amazon RDS versions.	June 18, 2021

Amazon RDS on AWS Outposts supports additional RDS for PostgreSQL versions	RDS on Outposts now supports RDS for PostgreSQL versions 12.5, 12.6, 13.1, and 13.2. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	May 28, 2021
Amazon RDS supports MariaDB 10.2.37 and 10.3.28	You can now create Amazon RDS DB instances running MariaDB versions 10.2.37 and 10.3.28. For more information, see MariaDB on Amazon RDS versions.	May 27, 2021
Amazon RDS for Oracle supports multitenant container database (CDB)	A multitenant architecture enables an Oracle database to be a CDB. In Oracle Database 19c, your CDB can include a single PDB. The user experience with a PDB is mostly identical to the user experience with a non-CDB. For more information, see RDS for Oracle architecture.	May 25, 2021
Amazon RDS on AWS Outposts supports Amazon RDS for SQL Server	RDS on Outposts now supports Amazon RDS for SQL Server. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	May 11, 2021

Amazon RDS extends support for cross-Region automated backups	You can now configure Amazon RDS database instances running Microsoft SQL Server to replicate DB snapshots and transacti on logs to a different AWS Region. For more informati on, see Replicating automated backups to another AWS Region.	May 7, 2021
Amazon RDS supports cross-Region automated backups for encrypted DB instances	You can now replicate DB snapshots and transacti on logs to a different AWS Region for encrypted Amazon RDS database instances running Oracle or PostgreSQ L. For more information, see Replicating automated backups to another AWS Region.	May 3, 2021
Amazon RDS on AWS Outposts supports Amazon CloudWatch monitoring	RDS on Outposts now supports Amazon CloudWatc h monitoring. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	April 21, 2021
RDS for PostgreSQL supports AWS Lambda functions	You can now invoke AWS Lambda functions for your RDS for PostgreSQL DB instances. For more informati on, see Invoking an AWS Lambda function from an RDS for PostgreSQL DB instance.	April 13, 2021

RDS for SQL Server supports extended events	You can use SQL Server extended events to capture debugging and troublesh ooting information. For more information, see Using extended events with Amazon RDS for Microsoft SQL Server.	April 8, 2021
Support for MySQL 8.0.23, 5.7.33, and 5.6.51	You can now create Amazon RDS DB instances running MySQL version 8.0.23, 5.7.33, and 5.6.51. For more information, see MySQL on Amazon RDS versions.	March 31, 2021
Automatic rollback on failed Amazon RDS for MySQL upgrade	If a DB instance upgrade from MySQL version 5.7 to MySQL version 8.0 fails, Amazon RDS rolls back the changes performed for the upgrade automatic ally. After the rollback, the MySQL DB instance is running MySQL version 5.7. For more information, see Rollback after failure to upgrade from MySQL 5.7 to 8.0.	March 18, 2021
Amazon RDS supports cross-Region read replicas in opt-in Regions	You can now replicate DB instances to opt-in Regions. For more information, see Creating a read replica in a different AWS Region.	March 18, 2021

Amazon RDS plans to deprecate Oracle Database 18c	Oracle Database 18c (18.0.0.0) is on a deprecation path. Oracle Corporation will no longer provide patches for Oracle Database 18c after the end-of-support date. On July 1, 2021, Amazon RDS plans to begin automatically upgrading Oracle Database 18c instances to Oracle Database 19c. Before the automatic upgrades begin, we highly recommend that you manually upgrade your existing Oracle Database 18c instances to Oracle Database 19c. For more information, see Preparing for the automatic upgrade of Oracle Database 18c.	March 11, 2021
Amazon RDS has ended support for Oracle Database 11g	You can only create DB instances for Oracle Database 12c Release 1 (12.1.0.2) and later. If you have Oracle Database 11g snapshots, upgrade them to a later release. For more information, see Upgrading an Oracle DB snapshot.	March 11, 2021

Amazon RDS supports continuous backups of DB instances in AWS Backup	You can now create automated backups in AWS Backup and restore DB instances from these backups to a specified time. For more information, see Using AWS Backup to manage automated backups.	March 10, 2021
Amazon RDS supports Oracle Management Agent (OMA) version 13.4	You can use Oracle Management Agent (OMA) version 13.4 with Oracle Enterprise Manager (OEM) Cloud Control 13c Release 4 Update 9. Amazon RDS for Oracle installs OMA, which then communicates with your Oracle Management Service (OMS) to provide monitoring information. If you run OMS 13.4, you can manage databases by installing OMA 13.4. For more information, see Oracle Management Agent for Enterprise Manager Cloud Control.	March 10, 2021

RDS Proxy endpoint enhancements	You can create additional endpoints associated with each RDS proxy. Creating an endpoint in a different VPC enables cross-VPC access for the proxy. Proxies for Aurora MySQL clusters can also have read-only endpoints. These reader endpoints connect to reader DB instances in the clusters and can improve read scalability and availability for query-intensive applications. For more information about RDS Proxy, see "Managing Connections with Amazon RDS Proxy" in the Amazon RDS User Guide or the Aurora user guide.	March 8, 2021
Amazon RDS extends supports for cross-Region automated backups	You can now configure Amazon RDS database instances running PostgreSQL to replicate DB snapshots and transaction logs to a different AWS Region. For more information, see Replicating automated backups to another AWS Region.	March 8, 2021

Replication filters for Amazon RDS for MariaDB and MySQL supported in the China (Beijing) Region and China (Ningxia) Region	Replication filtering is now supported in the China (Beijing) Region and China (Ningxia) Region. For more information, see Configuring replication filters with MariaDB and Configuring replication filters with MySQL.	March 5, 2021
Amazon RDS supports cross-Region DB snapshot copy in opt-in Regions	You can now copy DB snapshots to and from opt-in AWS Regions. For more information, see Copying snapshots across AWS Regions.	March 4, 2021
Amazon RDS for SQL Server supports Always On Availability Groups for Standard Edition	When you create a DB instance using the Multi-AZ configuration on SQL Server 2019 for the Standard Edition database engine, RDS automatically uses Availability Groups. For more information, see Multi-AZ deployments for Microsoft SQL Server.	February 23, 2021

Amazon RDS for Oracle introduces advisor-related procedures	The `rdsadmin_util` package includes the procedures `advisor_task_set_parameter` , `advisor_task_drop` , and `dbms_stats_init` . You can use these procedures to modify, stop, and re-enable advisor tasks such as `AUTO_STATS_ADVISOR_TASK` . For more information, see Setting parameters for advisor tasks.	February 23, 2021
Amazon RDS provides failover reasons for Multi-AZ DB instances	You can now see more detailed explanations when a Multi-AZ DB instance fails over to a standby replica. For more information, see Failover process for Amazon RDS.	February 18, 2021
Amazon RDS extends support for exporting snapshots to Amazon S3	You can now export DB snapshot data to Amazon S3 in China. For more information, see Exporting DB snapshot data to Amazon S3.	February 17, 2021

Replication filters for Amazon RDS for MariaDB and MySQL	You can configure replication filters for MySQL and MariaDB instances. Replication filters specify which databases and tables are replicated in a read replica. You can create lists of databases and tables to include or exclude for each read replica. For more information, see Configuring replication filters with MariaDB and Configuring replication filters with MySQL.	February 12, 2021
RDS for Oracle supports Oracle APEX 20.2v1	You can use Oracle APEX 20.2.v1 with all supported versions of Oracle Database. For more information, see Oracle Application Express.	February 2, 2021
Amazon RDS for SQL Server supports local instance storage for the tempdb database	You can now launch Amazon RDS for SQL Server on Amazon EC2 db.r5d and db.m5d instance types with the tempdb database configured to use an instance store. By placing tempdb data files and log files locally, you can achieve lower read and write latencies when compared to standard storage based on Amazon EBS. For more information, see Instance store support for the tempdb database on Amazon RDS for SQL Server.	January 27, 2021

Amazon RDS for PostgreSQL supports pg_partman and pg_cron	Amazon RDS for PostgreSQL now supports the pg_partman and pg_cron extensions. For more information on the pg_partman extension, see Managing PostgreSQL partitions with the pg_partman extension. For more information on the pg_cron extension, see Scheduling maintenance with the PostgreSQL pg_cron extension.	January 12, 2021
Amazon RDS supports publishing the Oracle Management Agent log to Amazon CloudWatch Logs	The Oracle Management Agent log consists of emctl.log, emdctlj.log, gcagent.log, gcagent_errors.log, emagent.nohup, and secure.log. Amazon RDS publishes each of these logs as a separate CloudWatch log stream. For more information, see Publishing Oracle logs to Amazon CloudWatch Logs.	December 28, 2020
Amazon RDS on AWS Outposts supports additional database versions	RDS on Outposts now supports additional MySQL and PostgreSQL versions. For more information, see Amazon RDS on AWS Outposts support for Amazon RDS features.	December 23, 2020

Amazon RDS on AWS Outposts supports CoIPs	RDS on Outposts now supports customer-owned IP addresses (CoIPs). CoIPs provide local or external connectivity to resources in your Outpost subnets through your on-premises network. For more information, see Customer-owned IP addresses for RDS on Outposts.	December 22, 2020
Amazon RDS for Oracle plans upgrade of 11g BYOL instances to 19c	On January 4, 2021, we plan to begin automatically upgrading all editions of Oracle Database 11g instances on the Bring Your Own License (BYOL) model to Oracle Database 19c. All Oracle Database 11g instances, including reserved instances, will move to the latest available Release Update (RU). For more information, see Preparing for the automatic upgrade of Oracle Database 11g BYOL.	December 11, 2020

Amazon RDS supports replicating automated backups to another AWS Region	You can now configure your Amazon RDS database instances to replicate snapshots and transacti on logs to a destination AWS Region of your choice. For more information, see Replicating automated backups to another AWS Region.	December 4, 2020
Amazon RDS for Oracle and Microsoft SQL Server support a new DB instance class	You can now use the db.r5b instance class to create Amazon RDS DB instances running Oracle or SQL Server. For more information, see Supported DB engines for DB instance classes.	December 4, 2020
Support for MariaDB 10.2.32	You can now create Amazon RDS DB instances running MariaDB version 10.2.32. For more information, see MariaDB on Amazon RDS versions.	November 25, 2020
Amazon RDS for SQL Server supports the Microsoft Business Intelligence Suite on SQL Server 2019	You can now run SQL Server Analysis Services, SQL Server Integration Services, and SQL Server Reporting Services on DB instances using the latest major version. For more information, see Options for the Microsoft SQL Server database engine.	November 24, 2020

Amazon RDS for PostgreSQL version 13 in the database preview environment	Amazon RDS for PostgreSQL now supports PostgreSQL version 13 in the database preview environment. For more information, see PostgreSQL 13 versions.	November 24, 2020
Amazon RDS Performance Insights introduces new dimensions	You can group database load according to the dimension groups for database (PostgreSQL, MySQL, and MariaDB), application (PostgreSQL), and session type (PostgreSQL). Amazon RDS also supports the dimensions db.name (PostgreSQL, MySQL, and MariaDB), db.application.name (PostgreSQL), and db.session_type.name (PostgreSQL). For more information, see Top load table.	November 24, 2020
Amazon RDS for MariaDB supports a new major version	You can now create Amazon RDS DB instances running MariaDB version 10.5. For more information, see MariaDB on Amazon RDS versions.	November 23, 2020
Support for MySQL 5.6.49	You can now create Amazon RDS DB instances running MySQL version 5.6.49. For more information, see MySQL on Amazon RDS versions.	November 20, 2020

Support for MySQL 5.5.62	You can now create Amazon RDS DB instances running MySQL version 5.5.62. For more information, see MySQL on Amazon RDS versions.	November 20, 2020
Performance Insights supports analyzing statistics for running PostgreSQL queries	You can now analyze statistics for running queries with Performance Insights for PostgreSQL DB instances. For more information, see Statistics for PostgreSQL.	November 18, 2020
Amazon RDS extends support for storage autoscaling	You can now enable storage autoscaling when creating a read replica, restoring a DB instance to a specified time, or restoring a MySQL DB instance from an Amazon S3 backup. For more information, see Managing capacity automatically with Amazon RDS storage autoscaling.	November 18, 2020
Amazon RDS for SQL Server supports Database Mail	With Database Mail you can send email messages from your Amazon RDS for SQL Server database instance. After specifying the email recipients, you can add files or query results to the message you send. For more information, see Using Database Mail on Amazon RDS for SQL Server.	November 4, 2020

Support for MySQL 8.0.21	You can now create Amazon RDS DB instances running MySQL version 8.0.21. For more information, see MySQL on Amazon RDS versions.	October 22, 2020
Amazon RDS extends support for exporting snapshots to Amazon S3	You can now export DB snapshot data to Amazon S3 in all commercial AWS Regions. For more information, see Exporting DB snapshot data to Amazon S3.	October 22, 2020
Amazon RDS for PostgreSQL supports read replica upgrades	With Amazon RDS for PostgreSQL, when you do a major version upgrade of the primary DB instance, read replicas are also automatically upgraded. For more information, see Upgrading the PostgreSQL DB engine.	October 15, 2020
Amazon RDS for MariaDB, MySQL and PostgreSQL support the Graviton2 DB instance classes	You can now use the Graviton2 DB instance classes db.m6g.x and db.r6g.x to create Amazon RDS DB instances running MariaDB, MySQL or PostgreSQL. For more information, see Supported DB Engines for All Available DB instance Classes.	October 15, 2020

Amazon RDS for SQL Server supports upgrades to SQL Server 2019	You can upgrade your SQL Server DB instances to SQL Server 2019. For more information, see Upgrading the Microsoft SQL Server DB Engine.	October 6, 2020
Amazon RDS for Oracle supports specifying the national character set	The national character set, also called the NCHAR character set, is used in the NCHAR, NVARCHAR2 , and NCLOB data types. When you create a database, you can specify either AL16UTF16 (default) or UTF8 as the NCHAR character set. For more information, see Oracle character sets supported in Amazon RDS.	October 2, 2020
Support for MySQL 5.7.31	You can now create Amazon RDS DB instances running MySQL version 5.7.31. For more information, see MySQL on Amazon RDS versions.	October 1, 2020
Amazon RDS for PostgreSQL supports exporting data to Amazon S3	You can query data from a PostgreSQL DB instance and export it directly into files stored in an Amazon S3bucket. For more information, see Exporting data from an RDS for PostgreSQL DB instance to Amazon S3.	September 24, 2020

Amazon RDS for MySQL 8.0 supports Percona XtraBackup	You can now use Percona XtraBackup to restore a backup into an Amazon RDS for MySQL 8.0 DB instance. For more information, see Restoring a backup into a MySQL DB instance.	September 17, 2020
Amazon RDS for SQL Server supports native backup and restore on DB instances with read replicas	You can restore a SQL Server native backup onto a DB instance that has read replicas configured. For more information, see Importing and exporting SQL Server databases.	September 16, 2020
Amazon RDS for SQL Server supports additional time zones	You can match your DB instance time zone with your chosen time zone. For more information, see Local time zone for Microsoft SQL Server DB instances.	September 11, 2020
Amazon RDS for PostgreSQL version 13 beta 3 in the database preview environment	Amazon RDS for PostgreSQL now supports PostgreSQL Version 13 Beta 3 in the Database Preview Environment. For more information, see PostgreSQL 13 versions.	September 9, 2020

Amazon RDS for SQL Server supports trace flag 692	You can now use trace flag 692 as a startup parameter using DB parameter gro ups. Enabling this trace flag disables fast inserts while bulk loading data into heap or clustered indexes. For more information, see Disabling fast inserts during bulk loading.	August 27, 2020
Amazon RDS for SQL Server supports Microsoft SQL Server 2019	You can now create RDS DB instances that use SQL Server 2019. For more informati on, see Microsoft SQL Server versions on Amazon RDS.	August 26, 2020
RDS for Oracle supports mounted replica database	When creating or modifying an Oracle replica, you can place it in mounted mode. Because the replica database doesn't accept user connectio ns, it can't serve a read-only workload. The mounted replica deletes archived redo log files after it applies them. The primary use for mounted replicas is cross-Region disaster recovery. For more information, see Overview of Oracle replicas.	August 13, 2020

RDS for Oracle plans upgrade of 11g SE1 LI instances	On November 1, 2020, we plan to begin automatically upgrading Oracle Database 11g SE1 License Included (LI) instances to Oracle Database 19c for Amazon RDS for Oracle. All 11g instances, including reserved instances, will move to the latest available Oracle Release Update (RU). For more information, see Preparing for the automatic upgrade of Oracle Database 11g SE1.	July 31, 2020
Amazon RDS supports new Graviton2 DB instance classes in preview release for PostgreSQL and MySQL	You can now create Amazon RDS DB instances running PostgreSQL or MySQL that use the db.m6g.x and db.r6g.x DB instance classes. For more information, see Supported DB engines for all available DB instance classes.	July 30, 2020
RDS for Oracle supports Oracle APEX 20.1v1	You can use Oracle APEX 20.1v1 with all supported versions of Oracle Database. For more information, see Oracle application Express.	July 28, 2020
Support for MySQL 8.0.20	You can now create Amazon RDS DB instances running MySQL version 8.0.20. For more information, see MySQL on Amazon RDS versions.	July 23, 2020

Amazon RDS for MariaDB and MySQL support new DB instance classes	You can now create Amazon RDS DB instances running MariaDB and MySQL that use the db.m5.16xlarge, db.m5.8xlarge, db.r5.16x large, and db.r5.8xlarge DB instance classes. For more information, see Supported DB engines for all available DB instance classes.	July 23, 2020
RDS for SQL Server supports disabling old versions of TLS and ciphers	You can turn certain security protocols and ciphers on and off. For more information, see Configuring security protocols and ciphers.	July 21, 2020
RDS supports Oracle Spatial on SE2	You can use Oracle Spatial in Standard Edition 2 (SE2) for all versions of 12.2, 18c, and 19c. For more information, see Oracle Spatial.	July 9, 2020
Amazon RDS supports AWS PrivateLink	Amazon RDS now supports creating Amazon VPC endpoints for Amazon RDS API calls to keep traffic between applications and Amazon RDS in the AWS network. For more informat ion, see Amazon RDS and interface VPC endpoints (AWS PrivateLink).	July 9, 2020

Amazon RDS for PostgreSQL versions 9.4.x has reached its end of support.	Amazon RDS for PostgreSQL no longer supports versions 9.4.x. For supported versions, see Supported PostgreSQL database versions.	July 8, 2020
Support for MariaDB 10.3.23 and 10.4.13	You can now create Amazon RDS DB instances running MariaDB version 10.3.23 and 10.4.13. For more informati on, see MariaDB on Amazon RDS versions.	July 6, 2020
Amazon RDS on AWS Outposts	You can create Amazon RDS DB instances on AWS Outposts. For more informati on, see Working with Amazon RDS on AWS Outposts.	July 6, 2020
Amazon RDS for Oracle creates inventory files automatically	To open service requests for BYOL customers, Oracle Support requests inventory files generated by Opatch. Amazon RDS for Oracle automatically creates inventory files every hour in the BDUMP directory. For more information, see Accessing Opatch files.	July 6, 2020
Support for MySQL 5.7.30 and 5.6.48	You can now create Amazon RDS DB instances running MySQL version 5.7.30 and 5.6.48. For more information, see MySQL on Amazon RDS versions.	June 25, 2020

Amazon RDS for Oracle supports ADRCI	The Automatic Diagnostic Repository Command Interpreter (ADRCI) utility is an Oracle command-line tool that you use to manage diagnostic data. By using the functions in the Amazon RDS package `rdsadmin_adrci_util`, you can list and package problems and incidents, and also show trace files. For more information, see Common DBA diagnostic tasks for Oracle DB instances.	June 17, 2020
Support for MySQL 8.0.19	You can now create Amazon RDS DB instances running MySQL version 8.0.19. For more information, see MySQL on Amazon RDS versions.	June 2, 2020
MySQL 8.0 supports lower case table names	You can now set the `lower_case_table_names` parameter to 1 for Amazon RDS DB instances running MySQL version 8.0.19 and higher 8.0 versions. For more information, see MySQL parameter exceptions for Amazon RDS DB instances.	June 2, 2020

Amazon RDS for Microsoft SQL Server supports SQL Server Integration Services (SSIS)	SSIS is a platform for data integration and workflow applications. You can enable SSIS on existing or new DB instances. It's installed on the same DB instance as your database engine. For more information, see Support for SQL Server Integration Services in SQL Server.	May 19, 2020
Amazon RDS for Microsoft SQL Server supports SQL Server Reporting Services (SSRS)	SSRS is a server-based application used for report generation and distribut ion. You can enable SSRS on existing or new DB instances . It's installed on the same DB instance as your database engine. For more information, see Support for SQL Server Reporting Services in SQL Server.	May 15, 2020
Amazon RDS for Microsoft SQL Server supports S3 integration on Multi-AZ instances	You can now use Amazon S3 with SQL Server features such as bulk insert on Multi-AZ DB instances. For more information, see Integrati ng an Amazon RDS for SQL Server DB instance with Amazon S3.	May 15, 2020

Amazon RDS for Oracle supports purging the recycle bin	The `rdsadmin.rdsadmin_util.purge_dba_recyclebin` procedure purges the recycle bin. For more information, see Purging the recycle bin.	May 13, 2020
Amazon RDS for Oracle improves manageability of Automatic Workload Repository (AWR)	The `rdsadmin.rdsadmin_diagnostic_util` procedures generate AWR reports and extract AWR data into dump files. For more information, see Generating performance reports with Automatic Workload Repository (AWR).	May 13, 2020
Amazon RDS for Microsoft SQL Server supports Microsoft Distributed Transaction Coordinator (MSDTC)	Amazon RDS for SQL Server supports distributed transactions between hosts. For more information, see Support for Microsoft Distributed Transaction Coordinator in SQL Server.	May 4, 2020
Amazon RDS for Microsoft SQL Server supports new versions	You can now create Amazon RDS DB instances running SQL Server versions 2017 CU19 14.00.3281.6, 2016 SP2 CU11 13.00.5598.27, 2014 SP3 CU4 12.00.6329.1, and 2012 SP4 GDR 11.0.7493.4 for all editions. For more information, see Microsoft SQL Server versions on Amazon RDS.	April 28, 2020

Amazon RDS available in the Europe (Milan) Region	Amazon RDS is now available in the Europe (Milan) Region. For more information, see Regions and Availability Zones.	April 28, 2020
Amazon RDS support for Local Zones	You can now launch DB instances into a Local Zone subnet. For more information, see Regions, Availability Zones, and Local Zones.	April 23, 2020
Amazon RDS available in the Africa (Cape Town) Region	Amazon RDS is now available in the Africa (Cape Town) Region. For more information, see Regions and Availability Zones.	April 22, 2020
Amazon RDS for Microsoft SQL Server supports SQL Server Analysis Services (SSAS)	SSAS is an online analytical processing (OLAP) and data mining tool that is installed within SQL Server. You can enable SSAS on existing or new DB instances. It's installed on the same DB instance as your database engine. For more information, see Support for SQL Server Analysis Services in SQL Server.	April 17, 2020

Amazon RDS proxy for PostgreSQL	Amazon RDS Proxy is now available for PostgreSQL. You can use RDS Proxy to reduce the overhead of connection management on your DB instance and also the chance of "too many connections" errors. The RDS Proxy is currently in public preview for PostgreSQL. For more information, see Managing connections with Amazon RDS proxy (preview).	April 8, 2020
Amazon RDS for Oracle supports Oracle APEX version 19.2.v1	Amazon RDS for Oracle now supports Oracle Application Express (APEX) version 19.2.v1. For more information, see Oracle application Express.	April 8, 2020
Amazon RDS for MariaDB supports a new major version	You can now create Amazon RDS DB instances running MariaDB version 10.4. For more information, see MariaDB on Amazon RDS versions.	April 6, 2020
Amazon RDS Performance Insights is available for Amazon RDS for MariaDB 10.4	Amazon RDS Performance Insights is now available for Amazon RDS for MariaDB version 10.4. For more information, see Using Amazon RDS performance insights.	April 6, 2020

Amazon RDS for PostgreSQL versions 9.3.x has reached its end of support	Amazon RDS for PostgreSQL no longer supports versions 9.3.x. For supported versions, see Supported PostgreSQL database versions.	April 3, 2020
Amazon RDS for Microsoft SQL Server supports read replicas	You can now create read replicas for SQL Server DB instances. For more informati on, see Working with read replicas.	April 3, 2020
Amazon RDS for Microsoft SQL Server supports multifile backups	You can now back up databases to multiple files using SQL Server native backup and restore. For more information, see Backing up a database.	April 2, 2020
Amazon RDS for Oracle integration with AWS License Manager	Amazon RDS for Oracle is now integrated with AWS License Manager. If you use the Bring Your Own License model, AWS License Manager integration makes it easier to monitor your Oracle license usage within your organizat ion. For more information, see Integrating with AWS License Manager.	March 23, 2020

Support for 64 TiB on db.r5 instances in Amazon RDS for MariaDB and MySQL	You can now create Amazon RDS DB instances for MariaDB and MySQL that use the db.r5 DB instance class with up to 64 TiB of storage. For more information, see Factors that affect storage performance.	March 18, 2020
Support for MySQL 8.0.17	You can now create Amazon RDS DB instances running MySQL version 8.0.17. For more information, see MySQL on Amazon RDS versions.	March 10, 2020
Amazon RDS Performance Insights is available for Amazon RDS for MySQL 8.0	Amazon RDS Performance Insights is now available for Amazon RDS for MySQL version 8.0.17 and higher 8.0 versions. For more information, see Using Amazon RDS performance insights.	March 10, 2020
Support for MySQL 5.6.46	You can now create Amazon RDS DB instances running MySQL version 5.6.46. For more information, see MySQL on Amazon RDS versions.	February 28, 2020

Amazon RDS Performance Insights is available for Amazon RDS for MariaDB 10.3	Amazon RDS Performance Insights is now available for Amazon RDS for MariaDB version 10.3.13 and higher 10.3 versions. For more information, see Using Amazon RDS performance insights.	February 26, 2020
Support for MySQL 5.7.28	You can now create Amazon RDS DB instances running MySQL version 5.7.28. For more information, see MySQL on Amazon RDS versions.	February 20, 2020
Support for MariaDB 10.3.20	You can now create Amazon RDS DB instances running MariaDB version 10.3.20. For more information, see MariaDB on Amazon RDS versions.	February 20, 2020
Amazon RDS for Microsoft SQL Server supports a new DB instance class	You can now create Amazon RDS DB instances running SQL Server that use the db.z1d DB instance class. For more information, see DB instance class support for Microsoft SQL Server.	February 19, 2020

Support for cross-account, cross-VPC Active Directory domains in Amazon RDS for SQL Server	Amazon RDS for Microsoft SQL Server now supports associating DB instances with Active Directory domains owned by different accounts and VPCs. For more information, see Using Windows authentication with a Microsoft SQL Server DB instance.	February 13, 2020
Oracle OLAP option	Amazon RDS for Oracle now supports the On-line Analytical Processing (OLAP) option for Oracle DB instances. You can use Oracle OLAP to analyze large amounts of data by creating dimensional objects and cubes in accordance with the OLAP standard. For more information, see Oracle OLAP.	February 13, 2020
FIPS 140-2 support for Oracle	Amazon RDS for Oracle supports the Federal Information Processing Standard Publication 140-2 (FIPS 140-2) for SSL/TLS connections. For more information, see FIPS support.	February 11, 2020

Amazon RDS for PostgreSQL supports new DB instance classes	You can now create Amazon RDS DB instances running PostgreSQL that use the db.m5.16xlarge, db.m5.8xlarge, db.r5.16xlarge, and db.r5.8xlarge DB instance classes. For more information, see Supported DB engines for all available DB instance classes.	February 11, 2020
Performance Insights supports analyzing statistics of running MariaDB and MySQL queries	You can now analyze statistics of running queries with Performance Insights for MariaDB and MySQL DB instances. For more information, see Analyzing statistics of running queries.	February 4, 2020
Support for exporting DB snapshot data to Amazon S3 for MariaDB, MySQL, and PostgreSQL	Amazon RDS supports exporting DB snapshot data to Amazon S3 for MariaDB, MySQL, and PostgreSQL. For more information, see Exporting DB snapshot data to Amazon S3.	January 23, 2020
Amazon RDS for MySQL supports Kerberos authentication	You can now use Kerberos authentication to authenticate users when they connect to your Amazon RDS for MySQL DB instances. For more information, see Using Kerberos authentication for MySQL.	January 21, 2020

Amazon RDS Performance Insights supports viewing more SQL text for Amazon RDS for Microsoft SQL Server	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Amazon RDS for Microsoft SQL Server DB instances. For more information, see Viewing more SQL text in the Performance Insights dashboard.	December 17, 2019
Amazon RDS proxy	You can reduce the overhead of connection management on your cluster, and reduce the chance of "too many connections" errors, by using the Amazon RDS Proxy. You associate each proxy with an RDS DB instance or Aurora DB cluster. Then you use the proxy endpoint in the connection string for your application. The Amazon RDS Proxy is currently in a public preview state. It supports the RDS for MySQL database engine. For more information, see Managing connections with Amazon RDS proxy (preview).	December 3, 2019

Amazon RDS on AWS Outposts (preview)	With Amazon RDS on AWS Outposts, you can create AWS-managed relational databases in your on-premises data centers. RDS on Outposts enables you to run RDS databases on AWS Outposts. For more information, see Amazon RDS on AWS Outposts (preview).	December 3, 2019
Amazon RDS for Oracle supports cross-region read replicas	Amazon RDS for Oracle now supports cross-region read replicas with Active Data Guard. For more information, see Working with read replicas and Working with Oracle read replicas.	November 26, 2019
Performance Insights supports analyzing statistics of running Oracle queries	You can now analyze statistics of running queries with Performance Insights for Oracle DB instances. For more information, see Analyzing statistics of running queries.	November 25, 2019
Amazon RDS for Microsoft SQL Server supports publishing logs to CloudWatch Logs	You can configure your Amazon RDS for SQL Server DB instance to publish log events directly to Amazon CloudWatch Logs. For more information, see Publishing SQL Server logs to Amazon CloudWatch Logs.	November 25, 2019

Amazon RDS for Microsoft SQL Server supports new DB instance classes	You can now create Amazon RDS DB instances running SQL Server that use the db.x1e and db.x1 DB instance classes. For more information, see DB instance class support for Microsoft SQL Server.	November 25, 2019
Amazon RDS for Microsoft SQL Server supports differential and log restores	You can restore differential backups and logs using SQL Server native backup and restore. For more information, see Using native backup and restore.	November 25, 2019
Multi-AZ supported on Amazon RDS for Microsoft SQL Server in new regions	Multi-AZ on SQL Server is now available in China, Middle East (Bahrain), and Europe (Stockholm). For more information, see Multi-AZ deployments for Microsoft SQL Server.	November 22, 2019
Amazon RDS for Microsoft SQL Server now supports bulk insert and S3 integration	You can transfer files between a SQL Server DB instance and an Amazon S3 bucket. Then you can use Amazon S3 with SQL Server features such as bulk insert. For more information, see Integrating an Amazon RDS for SQL Server DB instance with Amazon S3.	November 21, 2019

Performance Insights counters for Amazon RDS for Microsoft SQL Server	You can now add performance counters to your Performance Insights charts for Microsoft SQL Server DB instances. For more information, see Performance Insights counters for Amazon RDS for Microsoft SQL Server.	November 12, 2019
Amazon RDS for Microsoft SQL Server supports new DB instance class sizes	You can now create Amazon RDS DB instances running SQL Server that use the 8xlarge and 16xlarge instance sizes for the db.m5 and db.r5 DB instance classes. Instance sizes ranging from small to 2xlarge are now available for the db.t3 instance class. For more information, see DB instance class support for Microsoft SQL Server.	November 11, 2019
Support for PostgreSQL snapshot upgrades	If you have existing manual DB snapshots of your Amazon RDS PostgreSQL DB instances, you can now upgrade them to a later version of the PostgreSQL database engine. For more information, see Upgrading a PostgreSQL DB snapshot.	November 7, 2019

Amazon RDS for Oracle supports a new major version	You can now create Amazon RDS DB instances running Oracle Database 19c (19.0). For more information, see Oracle Database 19c with Amazon RDS.	November 7, 2019
Amazon RDS for PostgreSQL version 12.0 in the database preview environment	Amazon RDS for PostgreSQL now supports PostgreSQL Version 12.0 in the Database Preview Environment. For more information, see PostgreSQL version 12.0 in the database preview environment.	November 1, 2019
Amazon RDS for PostgreSQL supports Kerberos authentication	You can now use Kerberos authentication to authenticate users when they connect to your Amazon RDS DB instance running PostgreSQL. For more information, see Using Kerberos authentication with Amazon RDS for PostgreSQL.	October 28, 2019
OEM Management Agent database tasks for Oracle DB instances	Amazon RDS for Oracle DB instances now support procedures to invoke certain EMCTL commands on the Management Agent. For more information, see OEM agent database tasks.	October 24, 2019

Amazon RDS for PostgreSQL supports PostgreSQL transportable databases	PostgreSQL Transportable Databases provide an extremely fast method of migrating an RDS PostgreSQL database between two DB instances. For more information, see Transporting PostgreSQL databases between DB instances.	October 8, 2019
Amazon RDS for Oracle supports Kerberos authentication	You can now use Kerberos authentication to authenticate users when they connect to your Amazon RDS DB instance running Oracle. For more information, see Using Kerberos authentication with Amazon RDS for Oracle.	September 30, 2019
Amazon RDS for PostgreSQL version 12 beta 3 in the database preview environment	Amazon RDS for PostgreSQL now supports PostgreSQL Version 12 Beta 3 in the Database Preview Environment. For more information, see PostgreSQL version 12 beta 3 on Amazon RDS in the database preview environment.	August 28, 2019
Support for MySQL 8.0.16	You can now create Amazon RDS DB instances running MySQL version 8.0.16. For more information, see MySQL on Amazon RDS versions.	August 19, 2019

Amazon RDS for Oracle supports a new major version	You can now create Amazon RDS DB instances running Oracle Database 18c (18.0). For more information, see Oracle Database 18c with Amazon RDS.	August 15, 2019
Management Agent for OEM 13c release 3	Amazon RDS for Oracle DB instances now support the Management Agent for Oracle Enterprise Manager (OEM) Cloud Control 13c Release 3. For more information, see Oracle Management Agent for Enterprise Manager cloud control.	August 7, 2019
Amazon RDS for PostgreSQL version 12 beta 2 in the database preview environment	Amazon RDS for PostgreSQL now supports PostgreSQL Version 12 Beta 2 in the Database Preview Environment. For more information, see PostgreSQL version 12 beta 2 on Amazon RDS in the database preview environment.	August 6, 2019
Amazon RDS supports server collations for SQL Server	Amazon RDS for SQL Server supports a selection of collations for new DB instances. For more information, see Collations and character sets for Microsoft SQL Server.	July 29, 2019

Amazon RDS for Oracle supports Oracle APEX version 19.1.v1	Amazon RDS for Oracle now supports Oracle Application Express (APEX) version 19.1.v1. For more information, see Oracle application Express.	June 28, 2019
Amazon RDS for PostgreSQL version 13 beta 1 in the database preview environment	Amazon RDS for PostgreSQL now supports PostgreSQL Version 13 Beta 1 in the Database Preview Environment. For more information, see PostgreSQL 13 versions.	June 22, 2019
Amazon RDS storage autoscaling	Storage autoscaling for Amazon RDS DB instances enables Amazon RDS to automatically expand the storage associated with a DB instance to reduce the chance of out-of-space conditions. For information about storage autoscaling, see Working with storage for Amazon RDS DB instances.	June 20, 2019
Amazon RDS for Oracle supports db.z1d DB instance classes	You can now create Amazon RDS DB instances running Oracle that use the db.z1d DB instance classes. For more information, see DB instance class.	June 13, 2019

Amazon RDS Performance Insights supports viewing more SQL text for Amazon RDS for Oracle	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Amazon RDS for Oracle DB instances. For more information, see Viewing more SQL text in the Performance Insights dashboard.	June 10, 2019
Amazon RDS adds support native restores of SQL Server databases up to 16 TB	You can now do native restores of up to 16 TB from SQL Server to Amazon RDS. For more information, see Amazon RDS for SQL Server: Limitations and recommendations.	June 4, 2019
Amazon RDS adds support for Microsoft SQL Server audit	Using Amazon RDS for Microsoft SQL Server, you can audit server and database level events using SQL Server Audit, and view the results on your DB instance or send the audit log files directly to Amazon S3. For more information, see SQL Server Audit.	May 23, 2019

Improvements to Amazon RDS recommendations	Amazon RDS has improved its automated recommendations for database resources. For example, Amazon RDS now provides recommendations for database parameters. For more information, see Using Amazon RDS recommendations.	May 22, 2019
Support for more databases per DB instance for Amazon RDS for SQL Server	You can create up to 30 databases on each of your DB instances running Microsoft SQL Server. For more information, see Limits for Microsoft SQL Server DB instances.	May 21, 2019
Support for 64 TiB and 80k IOPS of storage for Amazon RDS for MariaDB, MySQL and PostgreSQL	You can now create Amazon RDS DB instances for MariaDB, MySQL and PostgreSQL with up to 64 TiB of storage and up to 80,000 provisioned IOPS. For more information, see DB instance storage.	May 20, 2019
Amazon RDS for MySQL supports upgrade prechecks	When you upgrade a DB instance from MySQL 5.7 to MySQL 8.0, Amazon RDS performs prechecks for incompatibilities. For more information, see Prechecks for upgrades from MySQL 5.7 to 8.0.	May 17, 2019

Support for the MySQL password validation plugin	You can now use the MySQL `validate_password` plugin for improved security of Amazon RDS for MySQL DB instances. For more information, see Using the Password Validation Plugin.	May 16, 2019
Performance Insights counters for Amazon RDS for Oracle	You can now add performance counters to your Performance Insights charts for Oracle DB instances. For more information, see Performance Insights counters for Amazon RDS for Oracle.	May 8, 2019
Support for per-second billing	Amazon RDS is now billed in 1-second increments in all AWS Regions except AWS GovCloud (US) for on-demand instances. For more information, see DB instance billing for Amazon RDS.	April 25, 2019
Support for importing data from Amazon S3 for Amazon RDS for PostgreSQL	You can now import data from Amazon S3 file into a table in an RDS PostgreSQL DB instance. For more information, see Importing Amazon S3 data into an RDS PostgreSQL DB instance.	April 24, 2019

Support for restoring 5.7 backups from Amazon S3	You can now create a backup of your MySQL version 5.7 database, store it on A mazon S3, and then restore the backup file onto a new Amazon RDS DB instance running MySQL. For more information, see Restoring a backup into a MySQL DB instance.	April 17, 2019
Support for multiple major version upgrades for Amazon RDS for PostgreSQL	With Amazon RDS for PostgreSQL, you can now choose from multiple major versions when you upgrade the DB engine. This feature enables you to skip ahead to a newer major version when you upgrade select PostgreSQ L engine versions. For more information, see Upgrading the PostgreSQL DB engine.	April 16, 2019
Support for 64 TiB of storage for Amazon RDS for Oracle	You can now create Amazon RDS DB instances for Oracle with up to 64 TiB of storage and up to 80,000 provisioned IOPS. For more information, see DB instance storage.	April 4, 2019
Support for MySQL 8.0.15	You can now create Amazon RDS DB instances running MySQL version 8.0.15. For more information, see MySQL on Amazon RDS versions.	April 3, 2019

Support for MariaDB 10.3.13	You can now create Amazon RDS DB instances running MariaDB version 10.3.13. For more information, see MariaDB on Amazon RDS versions.	April 3, 2019
Microsoft SQL Server 2008 R2 has reached its end of support on Amazon RDS	Microsoft SQL Server 2008 R2 has reached its end of support, coinciding with the Microsoft plan to end extended support for this version on July 9, 2019. Any existing Microsoft SQL Server 2008 R2 snapshots are to be automatically upgraded to the latest minor version of Microsoft SQL Server 2012 starting on June 1, 2019. For more information, see Microsoft SQL Server 2008 R2 support on Amazon RDS .	April 2, 2019
Always On availability groups supported in Microsoft SQL Server 2017	You can now use Always On Availability Groups in SQL Server 2017 Enterprise Edition 14.00.3049.1 or later. For more information, see Multi-AZ deployments for Microsoft SQL Server.	March 29, 2019

View volume metrics	You can now view metrics for the Amazon Elastic Block Store (Amazon EBS) volumes, which are the physical devices used for database and log storage. For more informati on, see Viewing Enhanced Monitoring.	March 20, 2019
Support for MySQL 5.7.25	You can now create Amazon RDS DB instances running MySQL version 5.7.25. For more information, see MySQL on Amazon RDS versions.	March 19, 2019
Amazon RDS for Oracle supports RMAN DBA tasks	Amazon RDS for Oracle now supports Oracle Recovery Manager (RMAN) DBA tasks, including RMAN backups. For more information, see Common DBA Recovery Manager (RMAN) tasks for Oracle DB instances.	March 14, 2019
Amazon RDS for PostgreSQL supports version 11.1	You can now create Amazon RDS DB instances running PostgreSQL version 11.1. For more information, see PostgreSQL version 11.1 on Amazon RDS.	March 12, 2019
Multiple-file restore is available in Amazon RDS for SQL Server	You can now restore from multiple files with Amazon RDS for SQL Server. For more information, see Restoring a database.	March 11, 2019

MariaDB 10.2.21	You can now create Amazon RDS DB instances running MariaDB version 10.2.21. For more information, see MariaDB on Amazon RDS versions.	March 11, 2019
Amazon RDS for Oracle supports read replicas	Amazon RDS for Oracle now supports read replicas with Active Data Guard. For more information, see Working with read replicas and Working with Oracle read replicas.	March 11, 2019
Amazon RDS Performance Insights is available for Amazon RDS for MariaDB	Amazon RDS Performance Insights is now available for Amazon RDS for MariaDB. For more information, see Using Amazon RDS Performance Insights.	March 11, 2019
MySQL 8.0.13 and 5.7.24	You can now create Amazon RDS DB instances running MySQL versions 8.0.13 and 5.7.24. For more information, see MySQL on Amazon RDS versions.	March 8, 2019
Amazon RDS Performance Insights is available for Amazon RDS for SQL Server	Amazon RDS Performance Insights is now available for Amazon RDS for SQL Server. For more information, see Using Amazon RDS Performance Insights.	March 4, 2019

Amazon RDS for Oracle supports Amazon S3 integration	You can now transfer files between an Amazon RDS for Oracle DB instance and an Amazon S3 bucket. For more information, see Integrating Amazon RDS for Oracle and Amazon S3.	February 26, 2019
Amazon RDS for MySQL and Amazon RDS for MariaDB support db.t3 DB instance classes	You can now create Amazon RDS DB instances running MySQL or MariaDB that use the db.t3 DB instance classes. For more information, see DB instance class.	February 20, 2019
Amazon RDS for MySQL and Amazon RDS for MariaDB support db.r5 DB instance classes	You can now create Amazon RDS DB instances running MySQL or MariaDB that use the db.r5 DB instance classes. For more information, see DB instance class.	February 20, 2019
Performance Insights counters for RDS for MySQL and PostgreSQL	You can now add performance counters to your Performance Insights charts for MySQL and PostgreSQL DB instances. For more information, see Performance Insights dashboard components.	February 19, 2019

Amazon RDS for PostgreSQL now supports adaptive autovacuum parameter tuning	Adaptive autovacuum parameter tuning with Amazon RDS for PostgreSQL helps prevent transaction ID wraparound by adjusting autovacuum parameter values automatically. For more information, see Reducing the likelihood of transaction ID wraparound.	February 12, 2019
Amazon RDS for Oracle supports Oracle APEX versions 18.1.v1 and 18.2.v1	Amazon RDS for Oracle now supports Oracle Application Express (APEX) versions 18.1.v1 and 18.2.v1. For more information, see Oracle application Express.	February 11, 2019
Amazon RDS Performance Insights supports viewing more SQL text for RDS for MySQL	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for MySQL DB instances. For more information, see Viewing more SQL text in the Performance Insights dashboard.	February 6, 2019
Amazon RDS for PostgreSQL supports db.t3 DB instance classes	You can now create Amazon RDS DB instances running PostgreSQL that use the db.t3 DB instance classes. For more information, see DB instance class.	January 25, 2019

Amazon RDS for Oracle supports db.t3 DB instance classes	You can now create Amazon RDS DB instances running Oracle that use the db.t3 DB instance classes. For more information, see DB instance class.	January 25, 2019
Amazon RDS Performance Insights supports viewing more SQL text for Amazon RDS PostgreSQL	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Amazon RDS PostgreSQL DB instances. For more information, see Viewing more SQL text in the Performance Insights dashboard.	January 24, 2019
Amazon RDS for Oracle supports a new version of SQLT	Amazon RDS for Oracle now supports SQLT version 12.2.180725. For more information, see Oracle SQLT.	January 22, 2019
Amazon RDS for PostgreSQL supports db.r5 DB instance classes	You can now create Amazon RDS DB instances running PostgreSQL that use the db.r5 DB instance classes. For more information, see DB instance class.	December 19, 2018

Amazon RDS for PostgreSQL now supports restricted password management	Amazon RDS for PostgreSQL enables you to restrict who can manage user passwords and password expiration changes by using the parameter `rds.restrict_password_commands` and the role `rds_password`. For more information, see Restricting password management.	December 19, 2018
Amazon RDS for PostgreSQL supports uploading database logs to Amazon CloudWatch Logs	Amazon RDS for PostgreSQL supports uploading database logs to CloudWatch Logs. For more information, see Publishing PostgreSQL logs to CloudWatch Logs.	December 10, 2018
Amazon RDS for Oracle supports db.r5 DB instance classes	You can now create Amazon RDS DB instances running Oracle that use the db.r5 DB instance classes. For more information, see DB instance class.	November 20, 2018
Retain backups when deleting a DB instance	Amazon RDS supports retaining automated backups when you delete a DB instance. For more information, see Working with backups.	November 15, 2018

Amazon RDS for PostgreSQL supports db.m5 DB instance classes	You can now create Amazon RDS DB instances running PostgreSQL that use the db.m5 DB instance classes. For more information, see DB instance class.	November 15, 2018
Amazon RDS for Oracle supports a new major version	You can now create Amazon RDS DB instances running Oracle version 12.2.	November 13, 2018
Amazon RDS for SQL Server supports Always On	Amazon RDS for SQL Server supports Always On Availability Groups. For more information, see Multi-AZ deployments for Microsoft SQL Server.	November 8, 2018
Amazon RDS for PostgreSQL supports outbound network access using custom DNS servers	Amazon RDS for PostgreSQL supports outbound network access using custom DNS servers. For more informati on, see Using a custom DNS server for outbound network access.	November 8, 2018
Amazon RDS for MariaDB, MySQL, and PostgreSQL supports 32 TiB of storage	You can now create Amazon RDS DB instances with up to 32 TiB of storage for MySQL, MariaDB, and PostgreSQL. For more information, see DB instance storage.	November 7, 2018

Amazon RDS for Oracle supports extended data types	You can now enable extended data types on Amazon RDS DB instances running Oracle. With extended data types, the maximum size is 32,767 bytes for the VARCHAR2, NVARCHAR2, and RAW data types. For more information, see Using extended data types.	November 6, 2018
Amazon RDS for Oracle supports db.m5 DB instance classes	You can now create Amazon RDS DB instances running Oracle that use the db.m5 DB instance classes. For more information, see DB instance class.	November 2, 2018
Amazon RDS for Oracle migration from SE, SE1, or SE2 to EE	You can now migrate from any Oracle Database Standard Edition (SE, SE1, or SE2) to Oracle Database Enterprise Edition (EE). For more information, see Migrating between Oracle editions.	October 31, 2018
Amazon RDS can now stop Multi-AZ instances	Amazon RDS can now stop a DB instance that is part of a Multi-AZ deployment. Formerly, the stop instance feature had a limitation for multi-AZ instances. For more information, see Stopping an Amazon RDS DB instance temporarily.	October 29, 2018

Amazon RDS Performance Insights is available for Amazon RDS for Oracle	Amazon RDS Performance Insights is now available for Amazon RDS for Oracle. For more information, see Using Amazon RDS Performance Insights.	October 29, 2018
Amazon RDS for PostgreSQL supports PostgreSQL version 11 in the database preview environment	Amazon RDS for PostgreSQL now supports PostgreSQL version 11 in the Database Preview Environment. For more information, see PostgreSQL version 11 on Amazon RDS in the database preview environment.	October 25, 2018
MySQL supports a new major version	You can now create Amazon RDS DB instances running MySQL version 8.0. For more information, see MySQL on Amazon RDS versions.	October 23, 2018
MariaDB supports a new major version	You can now create Amazon RDS DB instances running MariaDB version 10.3. For more information, see MariaDB on Amazon RDS versions.	October 23, 2018
Amazon RDS for Oracle supports Oracle JVM	Amazon RDS for Oracle now supports the Oracle Java Virtual Machine (JVM) option. For more information, see Oracle Java virtual machine.	October 16, 2018

Custom parameter group for restore and point in time recovery	You can now specify a custom parameter group when you restore a snapshot or perform a point in time recovery operation. For more information, see Restoring from a DB snapshot and Restoring a DB instance to a specified time.	October 15, 2018
Amazon RDS for Oracle supports 32 TiB storage	You can now create Oracle RDS DB instances with up to 32 TiB of storage. For more information, see DB instance storage.	October 15, 2018
Amazon RDS for MySQL supports GTIDs	Amazon RDS for MySQL now supports global transaction identifiers (GTIDs), which are unique across all DB instances and in a replication configuration. For more information, see Using GTID-based replication for RDS for MySQL.	October 10, 2018
MySQL 5.7.23, 5.6.41, and 5.5.61	You can now create Amazon RDS DB instances running MySQL versions 5.7.23, 5.6.41, and 5.5.61. For more information, see MySQL on Amazon RDS versions.	October 8, 2018
Amazon RDS for Oracle supports a new version of SQLT	Amazon RDS for Oracle now supports SQLT version 12.2.180331. For more information, see Oracle SQLT.	October 4, 2018

Amazon RDS for PostgreSQL now supports IAM authentication	Amazon RDS for PostgreSQL now supports IAM authentication. For more information see IAM database authentication for MySQL and PostgreSQL.	September 27, 2018
You can enable deletion protection for your Amazon RDS DB instances	When you enable deletion protection for a DB instance, the database cannot be deleted by any user. For more information, see Deleting a DB instance.	September 26, 2018
Amazon RDS for MySQL and Amazon RDS for MariaDB support db.m5 DB instance classes	You can now create Amazon RDS DB instances running MySQL or MariaDB that use the db.m5 DB instance classes. For more information, see DB instance class.	September 18, 2018
Amazon RDS now supports upgrades to SQL Server 2017	You can upgrade your existing DB instance to SQL Server 2017 from any version except SQL Server 2008. To upgrade from SQL Server 2008, first upgrade to one of the other versions first. For information, see Upgrading the Microsoft SQL Server DB engine.	September 11, 2018

Amazon RDS for PostgreSQL now supports PostgreSQL version 11 beta 3 in the database preview environment	In this release, the Write-Ahead Log (WAL) segment size (wal_segment_size) is now set to 64MB. For more about PostgreSQL version 11 Beta 3, see PostgreSQL 11 beta 3 released. For information on the Database Preview Environment, see Working with the database preview environment.	September 7, 2018
Amazon Aurora User Guide	The *Amazon Aurora User Guide* describes all Amazon Aurora concepts and provides instructions on using the various features with both the console and the command line interface. The *Amazon RDS User Guide* now covers non-Aurora database engines.	August 31, 2018
Amazon RDS Performance Insights is available for RDS for MySQL	Amazon RDS Performance Insights is now available for RDS for MySQL. For more information, see Using Amazon RDS Performance Insights.	August 28, 2018
Aurora PostgreSQL-Compatible Edition now supports Aurora Auto Scaling	Auto Scaling of Aurora replicas is now available for Aurora PostgreSQL-Compatible Edition. For more information, see Amazon Aurora auto scaling with Aurora replicas.	August 16, 2018

Aurora Serverless for Aurora MySQL	Aurora Serverless is an on-demand, autoscaling configuration for Amazon Aurora. For more information, see Using Amazon Aurora Serverless.	August 9, 2018
MySQL 5.7.22 and 5.6.40	You can now create Amazon RDS DB instances running MySQL versions 5.7.22 and 5.6.40. For more information, see MySQL on Amazon RDS versions.	August 6, 2018
Aurora is now available in the China (Ningxia) region	Aurora MySQL and Aurora PostgreSQL are now available in the China (Ningxia) region. For more information, see Availability for Amazon Aurora MySQL and Availability for Amazon Aurora PostgreSQL.	August 6, 2018
Amazon RDS for MySQL supports delayed replication	Amazon RDS for MySQL now supports delayed replication as a strategy for disaster recovery. For more information, see Configuring delayed replication with MySQL.	August 6, 2018
Amazon RDS Performance Insights is available for Aurora MySQL	Amazon RDS Performance Insights is now available for Aurora MySQL. For more information, see Using Amazon RDS Performance Insights.	August 6, 2018

Amazon RDS Performance Insights integration with Amazon CloudWatch	Amazon RDS Performance Insights automatically publishes metrics to Amazon CloudWatch. For more information, see Performance Insights metrics published to CloudWatch.	August 6, 2018
Amazon RDS recommendations	Amazon RDS now provides automated recommendations for database resources. For more information, see Using Amazon RDS recommendations.	July 25, 2018
Incremental snapshot copies across AWS Regions	Amazon RDS supports incremental snapshot copies across AWS Regions for both unencrypted and encrypted instances. For more information, see Copying snapshots across AWS Regions.	July 24, 2018
Amazon RDS Performance Insights is available for Amazon RDS for PostgreSQL	Amazon RDS Performance Insights is now available for Amazon RDS for PostgreSQL. For more information, see Using Amazon RDS Performance Insights.	July 18, 2018
Amazon RDS for Oracle supports Oracle APEX version 5.1.4.v1	Amazon RDS for Oracle now supports Oracle Application Express (APEX) version 5.1.4.v1. For more information, see Oracle application Express.	July 10, 2018

Amazon RDS for Oracle supports publishing logs to Amazon CloudWatch Logs	Amazon RDS for Oracle now supports publishing alert, audit, trace, and listener log data to a log group in CloudWatch Logs. For more information, see Publishing Oracle logs to Amazon CloudWatch Logs.	July 9, 2018
MariaDB 10.2.15, 10.1.34, and 10.0.35	You can now create Amazon RDS DB instances running MariaDB versions 10.2.15, 10.1.34, and 10.0.35. For more information, see MariaDB on Amazon RDS versions.	July 5, 2018
Aurora PostgreSQL 1.2 is available and compatible with PostgreSQL 9.6.8	Aurora PostgreSQL 1.2 is now available and is compatible with PostgreSQL 9.6.8. For more information, see Version 1.2.	June 27, 2018
Read replicas for Amazon RDS PostgreSQL support Multi-AZ deployments	RDS read replicas in Amazon RDS PostgreSQL now support multiple Availability Zones. For more information, see Working with PostgreSQL read replicas.	June 25, 2018

Performance Insights available for Aurora PostgreSQL	Performance Insights is generally available for Aurora PostgreSQL, with support for extended retention of performance data. For more information, see Using Amazon RDS performance insights.	June 21, 2018
Aurora PostgreSQL available in western US (northern California) region	Aurora PostgreSQL is now available in the western United States (Northern Cal ifornia) region. For more information, see Availability for Amazon Aurora PostgreSQ L.	June 11, 2018
Amazon RDS for Oracle now supports CPU configuration	Amazon RDS for Oracle supports configuring the number of CPU cores and the number of threads for each core for the processor of a DB instance class. For more information, see Configuri ng the processor of the DB instance class.	June 5, 2018

Earlier updates

The following table describes the important changes in each release of the *Amazon RDS User Guide* before June 2018.

Change	Description	Date changed
Amazon RDS for PostgreSQL now supports PostgreSQL Version 11 Beta 1 in the Database Preview Environment	PostgreSQL version 11 Beta 1 contains several improvements that are described in PostgreSQL 11 beta 1 released! For information on the Database Preview Environment, see Working with the Database Preview environment.	May 31, 2018
Amazon RDS for Oracle now supports TLS versions 1.0 and 1.2	Amazon RDS for Oracle supports Transport Layer Security (TLS) versions 1.0 and 1.2. For more information, see TLS versions for the Oracle SSL option.	May 30, 2018
Aurora MySQL supports publishing logs to Amazon CloudWatch Logs	Aurora MySQL now supports publishing general, slow, audit, and error log data to a log group in CloudWatch Logs. For more information, see Publishing Aurora MySQL to CloudWatch Logs.	May 23, 2018
Database Preview Environment for Amazon RDS PostgreSQL	You can now launch a new instance of Amazon RDS PostgreSQL in a preview mode. For more information about the Database Preview Environment see, Working with the Database Preview environment.	May 22, 2018
Amazon RDS for Oracle DB instances support new DB instance classes	Oracle DB instances now support the db.x1e and db.x1 DB instance classes. For more information, see DB instance classes and RDS for Oracle DB instance classes.	May 22, 2018
Amazon RDS PostgreSQL now supports	You can now use postgres_fdw to connect to a remote server from a read replica. For more information see, Using the postgres_fdw extension to access external data.	May 17, 2018

Change	Description	Date changed
postgres_fdw on a read replica.		
Amazon RDS for Oracle now supports setting sqlnet.ora parameters	You can now set sqlnet.ora parameters with Amazon RDS for Oracle. For more information, see Modifying connection properties using sqlnet.ora parameters.	May 10, 2018
Aurora PostgreSQL available in Asia Pacific (Seoul) region.	Aurora PostgreSQL is now available in the Asia Pacific (Seoul) region. For more information, see Availability for Amazon Aurora PostgreSQL.	May 9, 2018
Aurora MySQL supports backtracking	Aurora MySQL now supports "rewinding" a DB cluster to a specific time, without restoring data from a backup. For more information, see Backtracking an Aurora DB cluster.	May 9, 2018
Aurora MySQL supports encrypted migration and replication from external MySQL	Aurora MySQL now supports encrypted migration and replication from an external MySQL database. For more information, see Migrating data from an external MySQL database to an Amazon Aurora MySQL DB cluster and Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster.	April 25, 2018
Aurora PostgreSQL-Compatible Edition support for the Copy-on-Write protocol.	You can now clone databases in an Aurora PostgreSQL database cluster. For more information see, Cloning databases in an Aurora DB cluster.	April 10, 2018

Change	Description	Date changed
MariaDB 10.2.12, 10.1.31, and 10.0.34	You can now create Amazon RDS DB instances running MariaDB versions 10.2.12, 10.1.31, and 10.0.34. For more information, see MariaDB on Amazon RDS versions.	March 21, 2018
Aurora PostgreSQL Support for new regions	Aurora PostgreSQL is now available in the EU (London) and Asia Pacific (Singapore) regions. For more information, see Availability for Amazon Aurora PostgreSQL.	March 13, 2018
MySQL 5.7.21, 5.6.39, and 5.5.59	You can now create Amazon RDS DB instances running MySQL versions 5.7.21, 5.6.39, and 5.5.59. For more information, see MySQL on Amazon RDS versions.	March 9, 2018
Amazon RDS for Oracle now supports Oracle REST Data Services	Amazon RDS for Oracle supports Oracle REST Data Services as part of the APEX option. For more information, see Oracle Application Express (APEX).	March 9, 2018
Amazon Aurora MySQL-Compatible Edition available in new AWS Region	Aurora MySQL is now available in the Asia Pacific (Singapore) region. For the complete list of AWS Regions for Aurora MySQL, see Availability for Amazon Aurora MySQL.	March 6, 2018
Amazon RDS DB instances running Microsoft SQL Server support change data capture (CDC)	DB instances running Amazon RDS for Microsoft SQL Server now support change data capture (CDC). For more information, see Change data capture support for Microsoft SQL Server DB instances.	February 6, 2018

Change	Description	Date changed
Aurora MySQL supports a new major version	You can now create Aurora MySQL DB clusters running MySQL version 5.7. For more information, see Amazon Aurora MySQL database engine updates 2018-02-06.	February 6, 2018
Publish MySQL and MariaDB logs to Amazon CloudWatch Logs	You can now publish MySQL and MariaDB log data to CloudWatch Logs. For more information, see Publishing MySQL logs to Amazon CloudWatch Logs and Publishing MariaDB logs to Amazon CloudWatch Logs.	January 17, 2018
Multi-AZ support for read replicas	You can now create a read replica as a Multi-AZ DB instance. Amazon RDS creates a standby of your replica in another Availability Zone for failover support for the replica. Creating your read replica as a Multi-AZ DB instance is independent of whether the source database is a Multi-AZ DB instance. For more information, see Working with DB instance read replicas.	January 11, 2018
Amazon RDS for MariaDB supports a new major version	You can now create Amazon RDS DB instances running MariaDB version 10.2. For more information, see MariaDB 10.2 support on Amazon RDS.	January 3, 2018
Amazon Aurora PostgreSQL-Compatible Edition available in new AWS Region	Aurora PostgreSQL is now available in the EU (Paris) region. For the complete list of AWS Regions for Aurora PostgreSQL, see Availability for Amazon Aurora PostgreSQL.	December 22, 2017
Aurora PostgreSQL supports new instance types	Aurora PostgreSQL now supports new instance types. For the complete list of instance types, see Choosing the DB instance class.	December 20, 2017

Change	Description	Date changed
Amazon Aurora MySQL-Com patible Edition available in new AWS Region	Aurora MySQL is now available in the EU (Paris) region. For the complete list of AWS Regions for Aurora MySQL, see Availability for Amazon Aurora MySQL.	December 18, 2017
Aurora MySQL supports hash joins	This feature can improve query performance when you need to join a large amount of data by using an equijoin. For more information, see Working with hash joins in Aurora MySQL.	December 11, 2017
Aurora MySQL supports native functions to invoke AWS Lambda fun ctions	You can call the native functions `lambda_sync` and `lambda_async` when you use Aurora MySQL. For more information, see Invoking a Lambda function from an Amazon Aurora MySQL DB cluster.	December 11, 2017
Added Aurora PostgreSQL HIPAA eligibility	Aurora PostgreSQL now supports building HIPAA compliant applications. For more information, see Working with Amazon Aurora PostgreSQL.	December 6, 2017
Additional AWS Regions available for Amazon Aurora with PostgreSQL compatibility	Amazon Aurora with PostgreSQL compatibility is now available in four new AWS Regions. For more information, see Availability for Amazon Aurora PostgreSQL.	November 22, 2017
Modify storage for Amazon RDS DB instances running Microsoft SQL Server	You can now modify the storage of your Amazon RDS DB instances running SQL Server. For more informati on, see Modifying an Amazon RDS DB instance.	November 21, 2017

Change	Description	Date changed
Amazon RDS supports 16 TiB storage for Linux-based engines	You can now create MySQL, MariaDB, PostgreSQL, and Oracle RDS DB instances with up to 16 TiB of storage. For more information, see Amazon RDS DB instance storage.	November 21, 2017
Amazon RDS supports fast scale up of storage	You can now add storage to MySQL, MariaDB, PostgreSQL, and Oracle RDS DB instances in a few minutes. For more information, see Amazon RDS DB instance storage.	November 21, 2017
Amazon RDS supports MariaDB versions 10.1.26 and 10.0.32	You can now create Amazon RDS DB instances running MariaDB versions 10.1.26 and 10.0.32. For more information, see MariaDB on Amazon RDS versions.	November 20, 2017
Amazon RDS for Microsoft SQL Server now supports new DB instance classes	You can now create Amazon RDS DB instances running SQL Server that use the db.r4 and db.m4.16xlarge DB instance classes. For more information, see DB instance class support for Microsoft SQL Server.	November 20, 2017
Amazon RDS for MySQL and MariaDB now supports new DB instance classes	You can now create Amazon RDS DB instances running MySQL and MariaDB that use the db.r4, db.m4.16xlarge, db.t2.xlarge, and db.t2.2xlarge DB instance classes. For more information, see DB instance classes.	November 20, 2017
SQL Server 2017	You can now create Amazon RDS DB instances running Microsoft SQL Server 2017. You can also create DB instances running SQL Server 2016 SP1 CU5. For more information, see Amazon RDS for Microsoft SQL Server.	November 17, 2017

Change	Description	Date changed
Restore MySQL backups from Amazon S3	You can now create a backup of your on-premises database, store it on Amazon S3, and then restore the backup file onto a new Amazon RDS DB instance running MySQL. For more information, see Restoring a backup into an Amazon RDS for MySQL DB instance.	November 17, 2017
Auto Scaling with Aurora Replicas	Amazon Aurora MySQL now supports Aurora Auto Scaling. Aurora Auto Scaling dynamically adjusts the number of Aurora Replicas based on increases or decreases in connectivity or workload. For more information, see Amazon Aurora Auto Scaling with Aurora replicas.	November 17, 2017
Oracle default edition support	Amazon RDS for Oracle DB instances now supports setting the default edition for the DB instance. For more information, see Setting the default edition for a DB instance.	November 3, 2017
Oracle DB instance file validation	Amazon RDS for Oracle DB instances now supports validating DB instance files with the Oracle Recovery Manager (RMAN) logical validation utility. For more information, see Validating database files in RDS for Oracle.	November 3, 2017
Management Agent for OEM 13c	Amazon RDS for Oracle DB instances now support the Management Agent for Oracle Enterprise Manager (OEM) Cloud Control 13c. For more information, see Oracle Management Agent for Enterprise Manager Cloud Control.	November 1, 2017
Storage reconfiguration for Microsoft SQL Server snapshots	You can now reconfigure the storage when you restore a snapshot to an Amazon RDS DB instance running Microsoft SQL Server. For more information, see Restoring to a DB instance.	October 26, 2017

Change	Description	Date changed
Asynchronous key prefetch for Aurora MySQL-Compatible Edition	Asynchronous key prefetch (AKP) improves the performance of noncached index joins, by prefetching keys in memory ahead of when they are needed. For more information, see Working with asynchronous key prefetch in Amazon Aurora.	October 26, 2017
MySQL 5.7.19, 5.6.37, and 5.5.57	You can now create Amazon RDS DB instances running MySQL versions 5.7.19, 5.6.37, and 5.5.57. For more information, see MySQL on Amazon RDS versions.	October 25, 2017
General availability of Amazon Aurora with PostgreSQL compatibility	Amazon Aurora with PostgreSQL compatibility makes it simple and cost-effective to set up, operate, and scale your new and existing PostgreSQL deployments, thus freeing you to focus on your business and applications. For more information, see Working with Amazon Aurora PostgreSQL.	October 24, 2017
Amazon RDS for Oracle DB instances support new DB instance classes	Amazon RDS for Oracle DB instances now support memory optimized next generation (db.r4) instance classes. Amazon RDS for Oracle DB instances also now support the following new current generation instance classes: db.m4.16xlarge, db.t2.xlarge, and db.t2.2xlarge. For more information, see DB instance classes and RDS for Oracle DB instance classes.	October 23, 2017

Change	Description	Date changed
New feature	Your new and existing Reserved Instances can now cover multiple sizes in the same DB instance class. Size-flexible reserved instances are available for DB instances with the same AWS Region, database engine, and instance family, and across AZ configuration. Size-flexible reserved instances are available for the following database engines: Amazon Aurora, MariaDB, MySQL, Oracle (Bring Your Own License), PostgreSQL. For more information, see Size-flexible reserved DB instances.	October 11, 2017
New feature	You can now use the Oracle SQLT option to tune a SQL statement for optimal performance. For more information, see Oracle SQLT.	September 22, 2017
New feature	If you have existing manual DB snapshots of your Amazon RDS for Oracle DB instances, you can now upgrade them to a later version of the Oracle database engine. For more information, see Upgrading an Oracle DB snapshot.	September 20, 2017
New feature	You can now use Oracle Spatial to store, retrieve, update, and query spatial data in your Amazon RDS DB instances running Oracle. For more information, see Oracle Spatial.	September 15, 2017
New feature	You can now use Oracle Locator to support internet and wireless service-based applications and partner-based GIS solutions with your Amazon RDS DB instances running Oracle. For more information, see Oracle Locator.	September 15, 2017
New feature	You can now use Oracle Multimedia to store, manage, and retrieve images, audio, video, and other heterogeneous media data in your Amazon RDS DB instances running Oracle.	September 15, 2017

Change	Description	Date changed
New feature	You can now export audit logs from your Amazon Aurora MySQL DB clusters to Amazon CloudWatch Logs. For more information, see Publishing Aurora MySQL logs to Amazon CloudWatch Logs.	September 14, 2017
New feature	Amazon RDS now supports multiple versions of Oracle Application Express (APEX) for your DB instances running Oracle. For more information, see Oracle Application Express (APEX).	September 13, 2017
New feature	You can now use Amazon Aurora to migrate an unencrypted or encrypted DB snapshot or MySQL DB instance to an encrypted Aurora MySQL DB cluster. For more information, see Migrating an RDS for MySQL snapshot to Aurora and Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using an Aurora read replica.	September 5, 2017
New feature	You can use Amazon RDS for Microsoft SQL Server databases to build HIPAA-compliant applications. For more information, see Compliance program support for Microsoft SQL Server DB instances.	August 31, 2017
New feature	You can now use Amazon RDS for MariaDB databases to build HIPAA-compliant applications. For more information, see Amazon RDS for MariaDB.	August 31, 2017
New feature	You can now create Amazon RDS DB instances running Microsoft SQL Server with allocated storage up to 16 TiB, and Provisioned IOPS to storage ranges of 1:1–50:1. For more information, see Amazon RDS DB instance storage.	August 22, 2017

Change	Description	Date changed
New feature	You can now use Multi-AZ deployments for DB instances running Microsoft SQL Server in the EU (Frankfurt) region. For more information, see Multi-AZ deployments for Amazon RDS for Microsoft SQL Server.	August 3, 2017
New feature	You can now create Amazon RDS DB instances running MariaDB versions 10.1.23 and 10.0.31. For more information, see MariaDB on Amazon RDS versions.	July 17, 2017
New feature	Amazon RDS now supports Microsoft SQL Server Enterprise Edition with the License Included model in all AWS Regions. For more information, see Licensing Microsoft SQL Server on Amazon RDS.	July 13, 2017
New feature	Amazon RDS for Oracle now supports Linux kernel huge pages for increased database scalability. The use of huge pages results in smaller page tables and less CPU time spent on memory managemen t, increasing the performance of large database instances. You can use huge pages with your Amazon RDS DB instances running all editions of Oracle ver sions 12.1.0.2 and 11.2.0.4. For more information, see Turning on HugePages for an RDS for Oracle instance.	July 7, 2017
New feature	Updated to support encryption at rest (EAR) for db.t2.small and db.t2.medium DB instance classes for all non-Aurora DB engines. For more information, see Availability of Amazon RDS encryption.	June 27, 2017
New feature	Updated to support Amazon Aurora in the Europe (Frankfurt) region. For more information, see Availability for Amazon Aurora MySQL.	June 16, 2017

Change	Description	Date changed
New feature	You can now specify an option group when you copy a DB snapshot across AWS regions. For more information, see Considerations for option groups.	June 12, 2017
New feature	You can now copy DB snapshots created from specialized DB instances across AWS regions. You can copy snapshots from DB instances that use Oracle TDE, Microsoft SQL Server TDE, and Microsoft SQL Server Multi-AZ with Mirroring. For more information, see Copying a DB snapshot.	June 12, 2017
New feature	Amazon Aurora now allows you to quickly and cost-effectively copy all of your databases in an Amazon Aurora DB cluster. For more information, see Cloning databases in an Aurora DB cluster.	June 12, 2017
New feature	Amazon RDS now supports Microsoft SQL Server 2016 SP1 CU2. For more information, see Amazon RDS for Microsoft SQL Server.	June 7, 2017
Preview	Public preview of Amazon Aurora with PostgreSQL Compatibility. For more information, see Working with Amazon Aurora PostgreSQL.	April 19, 2017
New feature	Amazon Aurora now allows you to run an ALTER TABLE *tbl_name* ADD COLUMN *col_name column_definition* operation nearly instantaneously. The operation completes without requiring the table to be copied and without materially impacting other DML statements. For more information, see Altering tables in Amazon Aurora using fast DDL.	April 5, 2017
New feature	We have added a new monitoring command, SHOW VOLUME STATUS, to display the number of nodes and disks in a volume. For more information, see Displaying volume status for an Aurora DB cluster.	April 5, 2017

Change	Description	Date changed
New feature	You can now use your own custom logic in your custom password verification functions for Oracle on Amazon RDS. For more information, see Creating custom functions to verify passwords.	March 21, 2017
New feature	You can now access your online and archived redo log files on your Oracle DB instances on Amazon RDS. For more information, see Accessing online and archived redo logs.	March 21, 2017
New feature	You can now copy both encrypted and unencrypted DB cluster snapshots between accounts in the same region. For more information, see Copying a DB cluster snapshot across accounts.	March 7, 2017
New feature	You can now share encrypted DB cluster snapshots between accounts in the same region. For more information, see Sharing a DB cluster snapshot.	March 7, 2017
New feature	You can now replicate encrypted Amazon Aurora MySQL DB clusters to create cross-region Aurora Replicas. For more information, see Replicating Aurora MySQL DB clusters across AWS Regions.	March 7, 2017
New feature	You can now require that all connections to your DB instance running Microsoft SQL Server use Secure Sockets Layer (SSL). For more information, see Using SSL with a Microsoft SQL Server DB instance.	February 27, 2017
New feature	You can now set your local time zone to one of 15 additional time zones. For more information, see Supported time zones.	February 27, 2017

Change	Description	Date changed
New feature	You can now use the Amazon RDS procedure `msdb.dbo.rds_shrink_tempdbfile` to shrink the tempdb database on your DB instances running Microsoft SQL Server. For more information, see Shrinking the tempdb database.	February 17, 2017
New feature	You can now compress your backup file when you export your Enterprise and Standard Edition Microsoft SQL Server database from an Amazon RDS DB instance to Amazon S3. For more information, see Compressing backup files.	February 17, 2017
New feature	Amazon RDS now supports custom DNS servers to resolve DNS names used in outbound network access on your DB instances running Oracle. For more information, see Setting up a custom DNS server.	January 26, 2017
New feature	Amazon RDS now supports creating an encrypted read replica in another region. For more information, see Creating a read replica in a different AWS Region and CreateDBInstanceReadReplica.	January 23, 2017
New feature	Amazon RDS now supports upgrading a MySQL DB snapshot from MySQL 5.1 to MySQL 5.5.	January 20, 2017
New feature	Amazon RDS now supports copying an encrypted DB snapshot to another region for the MariaDB, MySQL, Oracle, PostgreSQL, and Microsoft SQL Server database engines. For more information, see Copying a DB snapshot and CopyDBSnapshot.	December 20, 2016

Change	Description	Date changed
New feature	Amazon Aurora MySQL now supports spatial indexing. Spatial indexing improves query performance on large datasets for queries that use spatial data. For more information, see Amazon Aurora MySQL and spatial data.	December 14, 2016
New feature	Amazon RDS now supports outbound network access on your DB instances running Oracle. You can use utl_http, utl_tcp, and utl_smtp to connect from your DB instance to the network. For more information, see Configuring UTL_HTTP access using certificates and an Oracle wallet.	December 5, 2016
New feature	Amazon RDS has retired support for MySQL version 5.1. However, you can restore existing MySQL 5.1 snapshots to a MySQL 5.5 instance. For more information, see Supported storage engines for RDS for MySQL.	November 15, 2016
New feature	Amazon RDS now supports Microsoft SQL Server 2016 RTM CU2. For more information, see Amazon RDS for Microsoft SQL Server.	November 4, 2016
New feature	Amazon RDS now supports major version upgrades for DB instances running Oracle. You can now upgrade your Oracle DB instances from 11g to 12c. For more information, see Upgrading the RDS for Oracle DB engine.	November 2, 2016
New feature	You can now create DB instances running Microsoft SQL Server 2014 Enterprise Edition. Amazon RDS now supports SQL Server 2014 SP2 for all editions and all regions. For more information, see Amazon RDS for Microsoft SQL Server.	October 25, 2016

Change	Description	Date changed
New feature	Amazon Aurora MySQL now integrates with other AWS services: You can load text or XML data into a table from an Amazon S3 bucket, or invoke an AWS Lambda function from database code. For more information, see Integrating Aurora MySQL with other AWS services.	October 18, 2016
New feature	You can now access the tempdb database on your Amazon RDS DB instances running Microsoft SQL Server. You can access the tempdb database by using Transact-SQL through Microsoft SQL Server Management Studio (SSMS), or any other standard SQL client application. For more information, see Accessing the tempdb database on Microsoft SQL Server DB instances on Amazon RDS.	September 29, 2016
New feature	You can now use the UTL_MAIL package with your Amazon RDS DB instances running Oracle. For more information, see Oracle UTL_MAIL.	September 20, 2016
New features	You can now set the time zone of your new Microsoft SQL Server DB instances to a local time zone, to match the time zone of your applications. For more information, see Local time zone for Microsoft SQL Server DB instances.	September 19, 2016
New feature	You can now use the Oracle Label Security option to control access to individual table rows in your Amazon RDS DB instances running Oracle Database 12c. With Oracle Label Security, you can enforce regulatory compliance with a policy-based administration model, and ensure that an access to sensitive data is restricted to only users with the appropriate clearance level. For more information, see Oracle Label Security.	September 8, 2016

Change	Description	Date changed
New feature	You can now connect to an Amazon Aurora DB cluster using the reader endpoint, which load-balances connections across the Aurora Replicas that are available in the DB cluster. As clients request new connections to the reader endpoint, Aurora distributes the connection requests among the Aurora Replicas in the DB cluster. This functionality can help balance your read workload across multiple Aurora Replicas in your DB cluster. For more information, see Amazon Aurora endpoints.	September 8, 2016
New feature	You can now support the Oracle Enterprise Manager Cloud Control on your Amazon RDS DB instances running Oracle. You can enable the Management Agent on your DB instances, and share data with your Oracle Management Service (OMS). For more information, see Oracle Management Agent for Enterprise Manager Cloud Control.	September 1, 2016
New feature	This release adds support to get an ARN for a resource. For more information, see Getting an existing ARN for Amazon RDS.	August 23, 2016
New feature	You can now assign up to 50 tags for each Amazon RDS resource, for managing your resources and tracking costs. For more information, see Tagging Amazon RDS resources.	August 19, 2016

Change	Description	Date changed
New feature	Amazon RDS now supports the License Included model for Oracle Standard Edition Two. For more information, see Creating an Amazon RDS DB instance. You can now change the license model of your Amazon RDS DB instances running Microsoft SQL Server and Oracle. For more information, see Licensing Microsoft SQL Server on Amazon RDS and RDS for Oracle licensing options.	August 5, 2016
New feature	Amazon RDS now supports native backup and restore for Microsoft SQL Server databases using full backup files (.bak files). You can now easily migrate SQL Server databases to Amazon RDS, and import and export databases in a single, easily-portable file, using Amazon S3 for storage, and AWS KMS for encryption. For more information, see Importing and exporting SQL Server databases using native backup and restore.	July 27, 2016
New feature	You can now copy the source files from a MySQL database to an Amazon Simple Storage Service (Amazon S3) bucket, and then restore an Amazon Aurora DB cluster from those files. This option can be considerably faster than migrating data using `mysqldump` . For more information, see Migrating data from an external MySQL database to an Aurora MySQL DB cluster.	July 20, 2016

Change	Description	Date changed
New feature	You can now restore an unencrypted Amazon Aurora DB cluster snapshot to create an encrypted Amazon Aurora DB cluster by including an AWS Key Management Service (AWS KMS) encryption key during the restore operation. For more information, see Encrypting Amazon RDS resources.	June 30, 2016
New feature	You can use the Oracle Repository Creation Utility (RCU) to create a repository on Amazon RDS for Oracle. For more information, see Using the Oracle Repository Creation Utility on RDS for Oracle.	June 17, 2016
New feature	Adds support for PostgreSQL cross-region read replicas. For more information, see Creating a read replica in a different AWS Region.	June 16, 2016
New feature	You can now use the AWS Management Console to easily add Multi-AZ with Mirroring to a Microsoft SQL Server DB instance. For more information, see Adding Multi-AZ to a Microsoft SQL Server DB instance.	June 9, 2016
New feature	You can now use Multi-AZ Deployments Using SQL Server Mirroring in the following additional regions: Asia Pacific (Sydney), Asia Pacific (Tokyo), and South America (São Paulo). For more information, see Multi-AZ deployments for Amazon RDS for Microsoft SQL Server.	June 9, 2016
New feature	Updated to support MariaDB version 10.1. For more information, see Amazon RDS for MariaDB.	June 1, 2016
New feature	Updated to support Amazon Aurora cross-region DB clusters that are read replicas. For more informati on, see Replicating Aurora MySQL DB clusters across AWS Regions.	June 1, 2016

Change	Description	Date changed
New feature	Enhanced Monitoring is now available for Oracle DB instances. For more information, see Monitoring OS metrics with Enhanced Monitoring and Modifying an Amazon RDS DB instance.	May 27, 2016
New feature	Updated to support manual snapshot sharing for Amazon Aurora DB cluster snapshots. For more information, see Sharing a DB cluster snapshot.	May 18, 2016
New feature	You can now use the MariaDB Audit Plugin to log database activity on MariaDB and MySQL database instances. For more information, see Options for MariaDB database engine and Options for MySQL DB instances.	April 27, 2016
New feature	In-place, major version upgrades are now available for upgrading from MySQL version 5.6 to version 5.7. For more information, see Upgrades of the RDS for MySQL DB engine.	April 26, 2016
New feature	Enhanced Monitoring is now available for Microsoft SQL Server DB instances. For more information, see Monitoring OS metrics with Enhanced Monitoring.	April 22, 2016
New feature	Updated to provide an Amazon Aurora **Clusters** view in the Amazon RDS console. For more information, see Viewing an Aurora DB cluster.	April 1, 2016
New feature	Updated to support SQL Server Multi-AZ with mirroring in the Asia Pacific (Seoul) region. For more information, see Multi-AZ deployments for Amazon RDS for Microsoft SQL Server.	March 31, 2016

Change	Description	Date changed
New feature	Updated to support Amazon Aurora Multi-AZ with mirroring in the Asia Pacific (Seoul) region. For more information, see Availability for Amazon Aurora MySQL.	March 31, 2016
New feature	PostgreSQL DB instances have the ability to require connections to use SSL. For more information, see Using SSL with a PostgreSQL DB instance.	March 25, 2016
New feature	Enhanced Monitoring is now available for PostgreSQL DB instances. For more information, see Monitoring OS metrics with Enhanced Monitoring.	March 25, 2016
New feature	Microsoft SQL Server DB instances can now use Windows Authentication for user authentication. For more information, see Working with AWS Managed Active Directory with RDS for SQL Server.	March 23, 2016
New feature	Enhanced Monitoring is now available in the Asia Pacific (Seoul) region. For more information, see Monitoring OS metrics with Enhanced Monitoring.	March 16, 2016
New feature	You can now customize the order in which Aurora Replicas are promoted to primary instance during a failover. For more information, see Fault tolerance for an Aurora DB cluster.	March 14, 2016
New feature	Updated to support encryption when migrating to an Aurora DB cluster. For more information, see Migrating data to an Aurora DB cluster.	March 2, 2016
New feature	Updated to support local time zone for Aurora DB clusters. For more information, see Local time zone for Aurora DB clusters.	March 1, 2016
New feature	Updated to add support for MySQL version 5.7 for current generation Amazon RDS DB instance classes.	February 22, 2016

Change	Description	Date changed
New feature	Updated to support *db.r3* and *db.t2* DB instance classes in the AWS GovCloud (US-West) region.	February 11, 2016
New feature	Updated to support encrypting copies of DB snapshots and sharing encrypted DB snapshots. For more information, see Copying a DB snapshot for Amazon RDS and Sharing a DB snapshot for Amazon RDS.	February 11, 2016
New feature	Updated to support Amazon Aurora in the Asia Pacific (Sydney) region. For more information, see Availability for Amazon Aurora MySQL.	February 11, 2016
New feature	Updated to support SSL for Oracle DB instances. For more information, see Using SSL with an RDS for Oracle DB instance.	February 9, 2016
New feature	Updated to support local time zone for MySQL and MariaDB DB instances. For more information, see Local time zone for MySQL DB instances and Local time zone for MariaDB DB instances.	December 21, 2015
New feature	Updated to support Enhanced Monitoring of OS metrics for MySQL and MariaDB instances and Aurora DB clusters. For more information, see Viewing metrics in the Amazon RDS console.	December 18, 2015
New feature	Updated to support db.t2, db.r3, and db.m4 DB instance classes for MySQL version 5.5. For more information, see DB instance classes.	December 4, 2015
New feature	Updated to support modifying the database port for an existing DB instance.	December 3, 2015

Change	Description	Date changed
New feature	Updated to support major version upgrades of the database engine for PostgreSQL instances. For more information, see Upgrades of the RDS for PostgreSQL DB engine.	November 19, 2015
New feature	Updated to support modifying the public accessibi lity of an existing DB instance. Updated to support db.m4 standard DB instance classes.	November 11, 2015
New feature	Updated to support manual DB snapshot sharing. For more information, see Sharing a DB snapshot for Amazon RDS.	October 28, 2015
New feature	Updated to support Microsoft SQL Server 2014 for the Web, Express, and Standard editions.	October 26, 2015
New feature	Updated to support the MySQL-based MariaDB database engine. For more information, see Amazon RDS for MariaDB.	October 7, 2015
New feature	Updated to support Amazon Aurora in the Asia Pacific (Tokyo) region. For more information, see Availabil ity for Amazon Aurora MySQL.	October 7, 2015
New feature	Updated to support db.t2 burst-capable DB instance classes for all DB engines and the addition of the db.t2.large DB instance class. For more information, see DB instance classes.	September 25, 2015
New feature	Updated to support Oracle DB instances on R3 and T2 DB instance classes. For more information, see DB instance classes.	August 5, 2015
New feature	Microsoft SQL Server Enterprise Edition is now available with the License Included service model. For more information, see Licensing Microsoft SQL Server on Amazon RDS.	July 29, 2015

Change	Description	Date changed
New feature	Amazon Aurora has officially released. The Amazon Aurora DB engine supports multiple DB instances in a DB cluster. For detailed information, see What is Amazon Aurora?.	July 27, 2015
New feature	Updated to support copying tags to DB snapshots.	July 20, 2015
New feature	Updated to support increases in storage size for all DB engines and an increase in Provisioned IOPS for SQL Server.	June 18, 2015
New feature	Updated options for reserved DB instances.	June 15, 2015
New feature	Updated to support using Amazon CloudHSM with Oracle DB instances using TDE.	January 8, 2015
New feature	Updated to support encrypting data at rest and new API version 2014-10-31.	January 6, 2015
New feature	Updated to include the new Amazon DB engine: Aurora. The Amazon Aurora DB engine supports multiple DB instances in a DB cluster. Amazon Aurora is currently in preview release and is subject to change. For detailed information, see What is Amazon Aurora?.	November 12, 2014
New feature	Updated to support PostgreSQL read replicas.	November 10, 2014
New API and features	Updated to support the GP2 storage type and new API version 2014-09-01. Updated to support the ability to copy an existing option or parameter group to create a new option or parameter group.	October 7, 2014
New feature	Updated to support InnoDB Cache Warming for DB instances running MySQL version 5.6.19 and later.	September 3, 2014

Change	Description	Date changed
New feature	Updated to support SSL certificate verification when connecting to MySQL version 5.6, SQL Server, and PostgreSQL database engines.	August 5, 2014
New feature	Updated to support the db.t2 burstable DB instance classes.	August 4, 2014
New feature	Updated to support the db.r3 memory optimized DB instance classes for use with the MySQL (version 5.6), SQL Server, and PostgreSQL database engines.	May 28, 2014
New feature	Updated to support SQL Server Multi-AZ deployments using SQL Server Mirroring.	May 19, 2014
New feature	Updated to support upgrades from MySQL version 5.5 to version 5.6.	April 23, 2014
New feature	Updated to support Oracle GoldenGate.	April 3, 2014
New feature	Updated to support the M3 DB instance classes.	February 20, 2014
New feature	Updated to support the Oracle Timezone option.	January 13, 2014
New feature	Updated to support replication between MySQL DB instances in different regions.	November 26, 2013
New feature	Updated to support the PostgreSQL DB engine.	November 14, 2013
New feature	Updated to support SQL Server transparent data encryption (TDE).	November 7, 2013
New API and new feature	Updated to support cross region DB snapshot copies; new API version, 2013-09-09.	October 31, 2013
New features	Updated to support Oracle Statspack.	September 26, 2013

Change	Description	Date changed
New features	Updated to support using replication to import or export data between instances of MySQL running in Amazon RDS and instances of MySQL running on-premises or on Amazon EC2.	September 5, 2013
New features	Updated to support the db.cr1.8xlarge DB instance class for MySQL 5.6.	September 4, 2013
New feature	Updated to support replication of read replicas.	August 28, 2013
New feature	Updated to support parallel read replica creation.	July 22, 2013
New feature	Updated to support fine-grained permissions and tagging for all Amazon RDS resources.	July 8, 2013
New feature	Updated to support MySQL 5.6 for new instances , including support for the MySQL 5.6 memcached interface and binary log access.	July 1, 2013
New feature	Updated to support major version upgrades from MySQL 5.1 to MySQL 5.5.	June 20, 2013
New feature	Updated DB parameter groups to allow expressions for parameter values.	June 20, 2013
New API and new feature	Updated to support read replica status; new API version, 2013-05-15.	May 23, 2013
New features	Updated to support Oracle Advanced Security features for native network encryption and Oracle Transparent Data Encryption.	April 18, 2013
New features	Updated to support major version upgrades for SQL Server and additional functionality for Provisioned IOPS.	March 13, 2013
New feature	Updated to support VPC By Default for RDS.	March 11, 2013

Change	Description	Date changed
New API and feature	Updated to support log access; new API version 2013-02-12	March 4, 2013
New feature	Updated to support RDS event notification subscriptions.	February 4, 2013
New API and feature	Updated to support DB instance renaming and the migration of DB security group members in a VPC to a VPC security group.	January 14, 2013
New feature	Updated for AWS GovCloud (US-West) support.	December 17, 2012
New feature	Updated to support m1.medium and m1.xlarge DB instance classes.	November 6, 2012
New feature	Updated to support read replica promotion.	October 11, 2012
New feature	Updated to support SSL in Microsoft SQL Server DB instances.	October 10, 2012
New feature	Updated to support Oracle micro DB instances.	September 27, 2012
New feature	Updated to support SQL Server 2012.	September 26, 2012
New API and feature	Updated to support provisioned IOPS. API version 2012-09-17.	September 25, 2012
New features	Updated for SQL Server support for DB instances in VPC and Oracle support for Data Pump.	September 13, 2012
New feature	Updated for support for SQL Server Agent.	August 22, 2012
New feature	Updated for support for tagging of DB instances.	August 21, 2012

Change	Description	Date changed
New features	Updated for support for Oracle APEX and XML DB, Oracle time zones, and Oracle DB instances in a VPC.	August 16, 2012
New features	Updated for support for SQL Server Database Engine Tuning Advisor and Oracle DB instances in VPC.	July 18, 2012
New feature	Updated for support for option groups and first option, Oracle Enterprise Manager Database Control.	May 29, 2012
New feature	Updated for support for read replicas in Amazon Virtual Private Cloud.	May 17, 2012
New feature	Updated for Microsoft SQL Server support.	May 8, 2012
New features	Updated for support for forced failover, Multi-AZ deployment of Oracle DB instances, and nondefault character sets for Oracle DB Instances.	May 2, 2012
New feature	Updated for Amazon Virtual Private Cloud (VPC) Support.	February 13, 2012
Updated content	Updated for new Reserved Instance types.	December 19, 2011
New feature	Updated for Oracle engine support.	May 23, 2011
Updated content	Console updates.	May 13, 2011
Updated content	Edited content for shortened backup and maintenance windows.	February 28, 2011
New feature	Added support for MySQL 5.5.	January 31, 2011
New feature	Added support for read replicas.	October 4, 2010
New feature	Added support for AWS Identity and Access Management (IAM).	September 2, 2010

Change	Description	Date changed
New feature	Added DB engine Version Management.	August 16, 2010
New feature	Added Reserved DB instances.	August 16, 2010
New Feature	Amazon RDS now supports SSL connections to your DB instances.	June 28, 2010
New Guide	This is the first release of the Amazon RDS User Guide.	June 7, 2010

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.